ADAPTIVE MODELLING OF SIGNAL DATA WITH APPLICATION TO IMAGE COMPRESSION

by

Mark Trumbo

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE
in the School
of
Engineering Science

SIMON FRASER UNIVERSITY
August 1994

APPROVAL

Name:	Mark Trumbo
Degree:	Master of Applied Science
Title of thesis:	Adaptive Modelling of Signal Data with Application to Image Compression

Date Approved:

Abstract

Traditionally, Markov models have not been successfully used for compression of signal data other than binary image data. Due to the fact that exact substring matches in non-binary signal data are rare, using full resolution conditioning information generally tends to make Markov models learn slowly, yielding poor compression. However, such models can be successfully used for non-binary signal data compression by varying both the resolution and the order of the conditioning information. In contrast to variable order methods, the overall model complexity (the number of states) is a function not just of the order but also the resolution. In the adaptive Markov algorithms proposed, the resolution and order are continually adjusted to minimize the codelength of the past samples in the hope that this choice will best compress the future samples as well, a technique inspired by Rissanen's Minimum Description Length (MDL) principle. Performance of this method meets or exceeds current approaches. Traditional techniques for adapting order 0 models to non-stationary inputs can be used in the order 0 component of the variable resolution/variable order models, or a new method of order 0 adaptation, which is presented, could be used. Splitting a non-binary input signal into a number of lower resolution subgroups is also explored as a technique to accelerate learning in Markov models. The common non-binary image compression technique of Gray coding the input data before splitting into bit planes for subsequent coding is analyzed and a non-binary pseudo-Gray code is proposed which can yield slightly better performance than the standard binary Gray code method.

Acknowledgments

I would like to thank my senior supervisor, Dr. J. Vaisey, for his support and guidance during the preparation of this thesis. I would also like to thank my friend, Caroline Barrière, for her inspiration and encouragement.

Contents

List of Tables

List of Figures

List of Symbols

symbol	description
A	the alphabet, symbols of which are elements in a string
M	the size of the alphabet
x_t	element t of the sequence $x = x_1 x_2 \cdots x_N$
x^t	all symbols from x_1 through x_t
$H(p)$	entropy of the pmf p
$D(p\|\|q)$	relative entropy, cross entropy, or divergence of p and q
δ_t	histogram decay rate at time t
h	half-life
W	the width or horizontal dimension of an image
Q	the number of states or contexts in a Markov model
$P(X = x)$	probability the random variable X will take on the value
	x; the shorthand notation $P(x)$ will also be used
$C(\zeta)$	the number of times the event ζ occurred in the current experiment

List of Abbreviations

abbreviation	description
ADPCM	adaptive differential pulse code modulation
AR	autoregressive
DPCM	differential pulse code modulation
DC-DPCM	distortion-constrained differential pulse code modulation
ECVQ	entropy constrained vector quantization
FOFR	fixed order/fixed resolution
FOVR	fixed order/variable resolution
FSM	finite state machine
IIR	infinite impulse response
JBIG	Joint Bi-level Image Experts Group
LPC	linear predictive coding
MDL	minimum description length
MMDC	multi-modal data compression
PPMI	prediction by partial matching for images
RLE	run-length encoding
SNR	signal to noise ratio
PSNR	peak signal to noise ratio
REC	relative efficiency counter
UMC	universal Markov coding
VOFR	variable order/fixed resolution
VOVR	variable order/variable resolution

Chapter 1

Introduction

Data compression is becoming more and more important. In view of the growing multimedia industry, compression of still images, video, and audio now merits extra attention because the classical techniques fall far short of the performance that can be obtained using a system optimized for compression of these data.

This thesis explores adaptive Markov modelling of signal data. For concreteness, particular attention is given to the task of lossless still image compression (or, equivalently, lossless intra-frame video coding) but is also applicable to the compression of all kinds of signal data.

1.1 Data

First, a definition of what is meant by data. One way to categorize data is into two groups, text data and signal data.

Text data appears in the form of machine- or human-readable strings, one example being a sequence of integers, each of which corresponds to a letter or symbol in a finite alphabet. Another example is the machine language used by a computer, perhaps in the form of an executable object code file. Much of the classical research in data compression was done with the goal of text compression in mind.

Signal data, on the other hand, is assumed to be formed by quantizing samples of a physical process. Examples of signal data are digitized images or sounds, or the digitized output of a pressure gauge.

One notable difference between signal data and text data is the absence of a reasonable distance measure for the latter. Whereas one can say conclusively that the two real numbers 1.2 and 1.3 are separated on R^1 by 0.1, it is more difficult to agree the distance between two words "airplane" and "planar", or even the two letters "a" and "p". Consequently, there is no conventionally accepted signal-to-noise ratio for text data.

Williams (1991, pages 95–98) puts it another way, saying that the difference between a signal compressor and a text compressor is that signal data assumes an ordering. If a deterministic one-to-one permutation were applied to the source, a text compressor would perform identically, whereas a signal compressor would perform worse. For example, if everywhere in a sample of text data, we substituted the character "e" for the character "k" and vice versa, an adaptive text compressor would compress the resulting sample to the same number of bits as the original. If, however, given a signal data source, we substituted the value 10 for the value 1000 and vice versa, we would expect a signal data compressor to react quite differently. This is because conventional signal compressors depend on the input being a slowly moving process, that is, one with a high correlation between temporally or spatially adjacent symbols.

1.2 Compression

The goal of data compression is to reduce the number of bits needed to either store or transmit data, either losslessly or with some loss.

1.2.1 Statistical *vs* Dictionary Methods

There are probably hundreds of different methods of compression, but most can be classified into two methods: statistical or dictionary.

Statistical coding is the class of data compression algorithms in which explicit statistical models of the data source are either assumed or are determined from the source and are used to assign probabilities to symbols in the data to be compressed. The overall optimal codelength for an input *with respect to a particular model* is the sum over all the input symbols of the negative logarithm[1] of the probability assigned

[1] All logarithms in this thesis are assumed to be base-2 logarithms and will be denoted by log.

to each symbol. Arithmetic coding (Langdon and Rissanen 1981) is a well-known technique used to convert the sequence of probabilities into a sequence of bits that can be then either transmitted or stored. The central task in statistical coding, therefore, is the construction of models that represent the source data well. One example of a statistical method is the Universal Markov Code (UMC) (Rissanen 1986a). Additionally, techniques like differential pulse code modulation (DPCM) and adaptive DPCM (ADPCM) can also be considered statistical methods because the coefficients of the (adaptive) linear predictor are determined from the statistics (the empirical autocorrelation function) of the source being coded or from the statistics of a set of sources of which the source being coded is thought to be a member.

Dictionary methods replace sequences of characters with indices into a dictionary and so form a class of algorithms distinct from those that build statistical models. Whereas dictionary methods exploit the same redundancy in the input data as do statistical methods, they do so without estimating symbol probabilities or source statistics. However, relationships exist between the two classes, and statistical methods can be made to emulate dictionary methods at the cost of higher computational burden (Rissanen 1983b). The most well-known dictionary method is the Ziv-Lempel compression algorithm (Ziv and Lempel 1978). This method is used in the popular UNIX program `compress`. Also, perhaps one of the earliest data compression methods, run-length encoding, can be considered a member of this group.

1.2.2 Lossless *vs* Lossy Methods

Another way to classify compression methods is by the fidelity of the reproduction.

Lossless compression means that if the input is compressed and then subsequently decompressed, the resulting output is exactly the same, bit for bit, as the input. In general, for text data, we are interested in lossless coding because, as mentioned previously, there is no conventionally accepted difference measure for text data. In some signal compression applications, as in the compression of medical imagery or other medical sensor data, no amount of loss, measurable or not, is acceptable. Thus, there is a need for lossless coding of both signal and text data.

However, in a large number of applications of compression of signal data, some amount of loss is acceptable. Such loss is incurred either when the channel strictly limits the rate or when it is simply desirable to achieve the highest compression at a given acceptable fidelity. Among such applications are the compression of non-critical consumer signal data (still images, video and audio, for example) for storage or transmission. Lossy methods are not really applicable to text data compression.

1.3 Performance

Two types of performance measures will be used in this thesis. One relates to the compression of the input, and the other to the fidelity of the reproduction of the input.

1.3.1 Compression Performance Measure

Because we will be dealing primarily with images, the compression measure used will be bits/pixel (bpp). For an input image consisting of N r-bit pixels and an output of size β (in bits), the compression will be reported as β/N. This performance measure is equally applicable to lossy or lossless coding.[2]

1.3.2 Fidelity Performance Measure

Despite the well-known fact that the signal to noise ratio (SNR) is not a perfect fidelity measure, it is used prevalently. Especially for images, some visual artifacts do not significantly alter the SNR but are quite obvious to the viewer. Conversely, other artifacts are virtually imperceptible but cause a large change in SNR. Nonetheless, for lack of a better measure, and to use a measure with which we are all comfortable, fidelity will be expressed in terms of SNR. Situations in which this measure is inappropriate will be noted. Of course, this performance measure is meaningful only in the realm of lossy coding.

[2]Another measure is the compression ratio, β/Nr. This measure is less common but is actually more useful for comparing methods because it maps the performance into a positive number independent of the entropy of the source. Using this metric, smaller values represent better compression, a value of 1 represents no compression, and percentage differences between methods can be more easily read. In this thesis, however, we adhere to the convention in the image compression literature and will report compression results in bits/pixel.

1.4 Goals of this Thesis

The primary goal of the work presented in this thesis is the compression of signal data, specifically images, using variable order and *variable resolution* adaptive Markov models.

Adaptive Markov models start with no information about the source and gradually build up estimates of the conditional probability distribution by maintaining histograms. Care must be taken to choose the right conditioning information and the right order model as a function of time because the accuracy of the probability distribution estimate varies with the number of sample processed and the order of the conditioning information. We will call this the *learning problem* for adaptive Markov models. Variable order adaptive Markov models have been used quite successfully for text data compression because they overcome the learning problem by starting with low order models and by gradually working up to higher order models.

However, adaptive Markov models have not been the method of choice for compression of signal data except for the compression of black and white images (Langdon and Rissanen 1981). Because the conditional probabilities are estimated based on the frequency of exact matches in the input data, and since exact matches are harder to find in signal data, these techniques do not work as well on signal data as do other approaches.

Indeed, signal data is usually losslessly compressed by subtracting a prediction of the signal from the signal and transmitting the residual, as in DPCM (Gersho and Gray 1992, pages 206-211). Linear prediction is designed to minimize the error between the signal and the prediction, so that the transmitted signal has lower energy than the original signal. The prediction need only be *close* to the actual value to get coding gain. Hence, this method works well for signal data because it does not depend on finding exact matches.

As will be shown in this thesis, however, if we do not look for exact matches, but rather look at reduced resolution matches, the effectiveness of adaptive Markov models on signal data is greatly improved. It will be shown that using reduced resolution conditioning information is similar in intent to variable order adaptive Markov modelling in that it speeds up learning the source statistics. By exploiting the fact that there is a distance measure for signal data, we can vary the resolution of the conditioning information giving us an even finer control over the number of states in the model.

A secondary goal is to investigate and extend the technique of splitting a non-binary input into separate sources called bit planes, each of which has one bit of resolution. This technique involves pre-coding the data with a binary Gray code prior to splitting into bit planes. The effect of this pre-coding on subsequent Markov modelling is analyzed and we propose a non-binary pseudo-Gray code that allows us to split an input into planes with arbitrary resolution, preserving relationships between bits that may be useful for increasing compression performance.

Most of the emphasis will be placed on lossless coding, as this presents itself as one of the more natural results of Markov modelling. Additionally, we will be concerned almost exclusively with natural image data, as this is currently one of the major applications of signal coders.

1.5 Organization of this Thesis

This thesis is arranged into 8 Chapters. Chapter 2 provides information on some theoretical background of signal data compression. The concept of adaptive Markov modelling is presented and the learning problem for both text and signal data is described. The chapter concludes with a discussion of the minimum description length (MDL) principle which is central in overcoming the learning problem and understanding both existing techniques and those described in this thesis. Chapter 3 gives a brief survey of current work in the field of Markov modelling for data compression, focusing on how variable order techniques have attempted to solve the learning problem for text data and how these methods fall short when modelling signal data. Before describing methods for overcoming the learning problem for Markov models and signal data, Chapter 4 presents a new method of order 0 adaptation and gives a simple first look at how useful it is for a model to monitor its own performance and adjust itself accordingly. Chapter 5 presents the concepts of adaptive variable resolution/variable order modelling for lossless coding of signal data, showing that adaptive Markov models can be effectively used on signal data. Chapter 6 presents one of the traditional methods of overcoming the learning problem in Markov modelling, namely, splitting a signal into bit planes, and extends this idea to bit groups of arbitrary size. Chapter 7 presents applications of the work in adaptive Markov modelling to the task of signal prediction and then describes distortion-constrained DPCM (DC-DPCM), which can be considered a recursive Markov model. Chapter 8 summarizes the work contained in this thesis and gives suggestions for further research.

1.6 Contributions of this Thesis

The main contributions of this thesis are

- coding a non-binary signal data input by simultaneously varying the resolution and order of the conditioning information in a Markov model, using as a guide the local performance of the sub-models
- splitting a non-binary signal data input into planes containing more than one bit using a non-binary pseudo-Gray code, and passing the resulting bit groups to a variable order model
- using a function of the derivative of the per-symbol codelength of an order zero histogram model as the histogram count decay to counteract order zero non-stationarity

Chapter 2

Background

This chapter gives background on some basic concepts required for a discussion of statistical data compression. We will discuss the decoupling of the model from the coder, and the justification for doing so, namely, arithmetic coding. Background on Markov models will be presented including a description of two forms of adaptation, the construction of adaptive Markov models from a source, and most importantly, the Markov model learning problem. The specific problems of using Markov models for signal data will be presented as motivation for the work in this thesis. Finally, a section will be devoted to the minimum description length (MDL) principle and to the idea of stochastic complexity, both of which will be useful in the following chapters.

2.1 Adaptive Modelling and Arithmetic Coding

Before proceeding, some notational conventions need to be explained. First, a string x of length N is composed of a sequence of symbols x_i for $1 \leq i \leq N$:

$$x = x_1 x_2 x_3 \cdots x_N \ , \tag{2.1}$$

and the substring consisting of the first t symbols of x is denoted by x^t. With this definition, x^N can be used interchangeably with x. Second, we will follow the convention of using upper case letters for random variables and lower case for specific values of those variables where necessary for clarity. Usually we will just use the lower case letters and assume the reader understands when we are referring to random variables and when we are referring to specific instances of them.

Given two random variables X and Y, we are interested in quantifying the amount of information conveyed by a particular outcome y of Y about the outcome x of X. If X and Y are independent, then we get no information about X given the outcome of Y. If, however, the random variables are completely dependent, then the outcome of Y tells us exactly the outcome of X, and so the only information conveyed by that outcome is whatever information would have been conveyed by x. One way of quantifying the amount of information is then

$$I(x;y) = \log \frac{P(x|y)}{P(x)} \ , \tag{2.2}$$

which is called the mutual information.

As desired, when the random variables are independent, $P(X = x|Y = y) = P(X = x)$ and $I(x;y)$ is zero. Whenever $P(X = x|Y = y)$ is different from $P(X = x)$ some information conveyed. Finally, when the random variables are completely dependent, $P(X = x|Y = y) = 1$ and the mutual information is $\log 1/P(x)$, the information conveyed by the outcome x alone. This special case does not depend on y, and is called the self information,

$$I(x) = \log \frac{1}{p(x)} = -\log p(x) \ . \tag{2.3}$$

With this definition of information, a high probability event conveys less information than a low probability event. While the choice of the measure is somewhat arbitrary, it is intuitively attractive and it is this definition that forms the basis of statistical data compression.

In statistical compression, we are able to decouple the model from the coder as in Figure 2.1. The only means by which the model and the coder communicate is via symbols and symbol distributions. The model accepts a stream of source symbols and at each time t generates the distribution

$$P(x_{t+1}|(z_t = \sigma(x^t))) \tag{2.4}$$

for all $x_{t+1} \in A$, where A is the source alphabet and σ is a function that chooses some set of symbols from the entire past sequence as the conditioning information or the *context* z_t.[1] The coder accepts this distribution and a source symbol and codes the

[1]The notion of a function σ that generates the conditioning information is based on (Rissanen 1986a) in which it is called the "sorting function" (see Section 3.1.4 for more details).

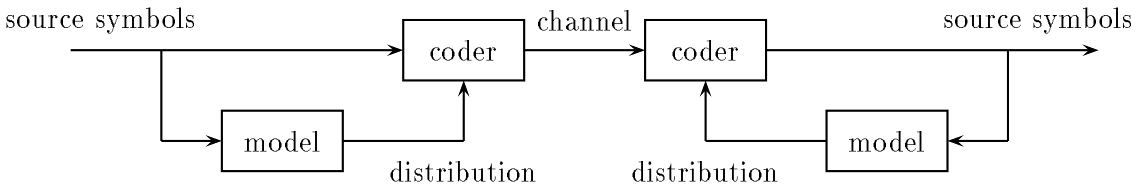

Figure 2.1: Statistical data compression

symbol with as close to log $P(x_{t+1}|z_t)$ bits as possible, per the quantitative definition of information (2.3). The coder produces a stream of channel symbols. Let us consider each part separately.

2.1.1 Arithmetic Coding

Arithmetic coding is the technique that allows the decoupling of the model and coder. Given a string x^t composed of symbols taken from a finite alphabet A of size M, and a model that predicts the probability of each symbol in the alphabet, arithmetic coding represents the string as an interval, I, of real numbers in the range $[0, 1)$. The size of the interval is the probability of the string

$$P(x^t) = \prod_{i=0}^{t-1} P(x_{i+1}|z_i) \tag{2.5}$$

given by the model. The message, C, is the sum of the probability of all strings of the same length that precede the current string in the lexical order of the strings (Langdon and Rissanen 1981) with the probability of the null string being zero.

The algorithm starts with an interval of size 1.0 and, as each symbol from the string is coded, the interval gets smaller in proportion to the probability of the symbol as given by the model. More probable symbols reduce the size of the interval less than less probable symbols, and hence add fewer bits to the message, C.

As an example, consider all possible 3 bit sequences generated by a source that emits zeros with probability $p_0 = 1/3$ and ones with probability $p_1 = 2/3$. In Figure 2.2, the range [0,1] is broken into eight intervals, each one corresponding to a unique 3 bit source sequence. The sequence "111" is the most probable ($p = 8/27$)

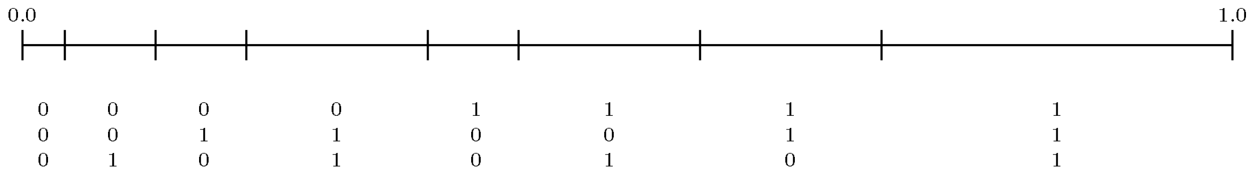

Figure 2.2: An example of arithmetic coding

and its interval is the largest of all the intervals. On the other hand, the sequence "000" is the least probable ($p = 1/27$) and its interval is the smallest. Sequences with equal probability have equal-sized intervals.

Since both the encoder and decoder have the same probability model for the source, only the interval corresponding to the source sequence needs to be transmitted. This is done by transmitting a value that falls in the interval. Any value would do, but since the goal is compression, we of course transmit the value with the fewest digits in its binary representation. Smaller intervals will require more bits to specify and larger intervals will require fewer bits to specify, so arithmetic coding achieves compression by assigning long messages to infrequent source sequences and short messages to frequent source sequences.

To define arithmetic coding more rigorously, let

$$C(\emptyset) = 0 \tag{2.6}$$

$$I(\emptyset) = 1 \tag{2.7}$$

indicate the initial conditions of the message and the interval. For any new symbol x_{t+1} and previous string x^t

$$I(x^t x_{t+1}) = I(x^t) P(x_{t+1}|z_t) \tag{2.8}$$

$$C(x^t x_{t+1}) = C(x^t) + \sum_{i=0}^{x_{t+1}-1} I(x^t i) \ . \tag{2.9}$$

We see that each new symbol x_{t+1} reduces the interval in proportion to the probability of the symbol, $P(x_{t+1})$. As stated previously, any number in the interval is a valid message, and in this formal definition, the message C is taken to be the lower bound of the interval corresponding to the source sequence. Finally, for any string x^t

$$I(x^t) = \sum_{i=0}^{M-1} x^t i \ , \tag{2.10}$$

which is required for decodability as described in (Rissanen and Langdon 1981).

Note that, as with Huffman coding (Huffman 1952), the effectiveness of arithmetic coding is only as good as the model of the source. However in contrast to Huffman coding, because arithmetic coding does not require integral bit allocation, it can closely approach the entropy of the source model even if one symbol has a probability approaching unity. Huffman coding performs very poorly in this case because at least one bit has to be transmitted for each symbol. Also, if there are several such low probability symbols, the code length for each can grow unmanageably large.

Despite the advantages of arithmetic coding over Huffman coding, in the simple formulation given above, problems arise in its implementation. Specifically, the entire string x^t must be coded prior to transmission and the precision required to represent the message C grows with the length of x^t. However, these problems are not unsurmountable. For example, Witten, Neal, and Cleary (1987) have implemented a practical fixed-precision arithmetic codec with incremental transmission and reception. There also exist variants that avoid the multiplication operations inherent in arithmetic coding (Rissanen and Mohiuddin 1989; Langdon and Rissanen 1981), reducing the computational resources required. A final disadvantage of arithmetic coding (and all non-immediately decodable codes, for that matter) is that a single bit error in the coded stream makes the rest of the data irrecoverable. Piecewise arithmetic coding (Teuhola and Raita 1991) avoids this problem by sending codewords of a fixed length. However, due to the associated overhead, this method may in fact be equivalent in performance (both bit error rate and compression ratio) to an arithmetic coder followed by an error correcting coder.

2.1.2 Adaptive Modelling

Now that we know that an arithmetic coder can come arbitrarily close to the entropy of the source model, the real problem in statistical data compression becomes determining the model. The following sections will describe one class of models, Markov models, which can be effectively used to generate low conditional entropy distributions.

Markov Models

A *finite, discrete-time Markov chain* (see, for example, (Feller 1957)) consists of a set of q states numbered 1 through q. At each point in (quantized) time t, the chain is at a particular state s, and at the next moment, the chain changes state according to a set of probabilities $P_{i,j}$. That is, if the chain is in state i it will change to state j with probability $P_{i,j}$. Of course, the probabilities are constrained so that $\sum_{i=1}^{q} P_{i,j} = \sum_{j=1}^{q} P_{i,j} = 1$.

A *Markov source* is essentially a Markov chain that emits a symbol at each state change, a unique symbol being associated with each transition i, j.

A *Markov model* is a stochastic deterministic finite state machine. Like a Markov source, it has a set of transition probabilities $P_{i,j}$. We use a Markov model as an approximation to an unknown Markov source, or to a completely unknown source which can be approximated by a Markov source. In this thesis, as in many other works on Markov modelling (Williams 1991; Langdon and Rissanen 1981), we make the simplifying assumption that the state of a Markov source is determined by the previous n symbols it has emitted. This technically makes the Markov source a finite context automaton, but for convenience, in this thesis we will use the terms state and context interchangeably. The order of the corresponding Markov model is n.

The first problem in Markov modelling is determining the number of states in the Markov model. The next problem is determining the transition probabilities. Assuming that the source is in fact a Markov source, solving these two problems allows us to compress the sequence of symbols emitted by the Markov source down to the source entropy. Even if the data generating mechanism is not a Markov source, modelling it as such may allow us to effectively compress the data.

Adaptive Models

Adaptive algorithms accept data sequentially, are presented with each symbol in the sample only once, and modify the way they compress in response to the history. Adaptive algorithms are attractive in that it is unnecessary to perform a pre-scan of the data, and no side information need be sent. This is appropriate for compressors that may be part of a larger system (an image communication system, for example) in which only a certain amount of delay is acceptable.

Williams (1991) devotes much effort to rigorously defining adaptation. As it will be used in this thesis, a summary of his terminology, tailored for application in particular to Markov models, follows.

Context Adaptation A model can use counts of the number of times each symbol in the alphabet has appeared in a particular context, and from these make an estimate of the probability that the source being modelled will emit each symbol when in that context. If the model changes these counts as the input is processed, it is said to be *contextually adaptive.* If the count for a symbol is simply incremented each time the symbol appears in the context, the model is said to be *asymptotically adaptive.* If, on the other hand, the counts in a context only represent the frequency of occurrence during the last k symbols, the model is *locally adaptive.* Local adaptation can also be achieved by decaying the counts so that more recent samples have more influence on the model than less recent samples. One method is to increment the count of each new sample while simultaneously decrementing that of the k most recent symbols. Alternatively, each time a new sample arrives, all the counts are first multiplied by a constant $\delta = e^{\ln 0.5/h}$, and then the count for the current symbol is incremented. The parameter h is the half-life of the sample, which is the time (in samples) before the effect of a sample becomes half that of the current sample. This method requires a lot of calculation, which may not be practical, so the decay could be done at a larger interval than every sample, but with a smaller half-life. Alternatively, the decay operation could be done only when a count reaches some threshold.

Structural Adaptation A model that alters its structure (its order, for example) based on the nature of the input is said to be structurally adaptive. One example of a structurally adaptive algorithm is UMC because it is a tree-structured algorithm and has defined growth rules based on the performance of the model on the input. (See Appendix B for details of the operation of the UMC algorithm.)

Context adaptation and structural adaptation are interrelated. The goal of context adaptation is to counteract any non-stationarity within a context, but the contexts are part of the model structure. It may be that the structure is correct but that the data generating mechanism is non-stationary. Or it may be that the data generating mechanism is stationary, but that the structure of the model is incorrect. In both cases, the statistics at each context *appear* non-stationary, necessitating the use of context adaptation.

Constructing Markov Models

The most important factor to consider in constructing a Markov model is the choice of the conditioning information; i.e., choosing the function σ in (2.4). Choosing the function σ corresponds to choosing the number of states q in the Markov model and also to what those states correspond. For example, if the function σ always maps the previous string x^t to the previous symbol x_t so that the context $z_t = x_t$, then the model would realize the conditional probability $P(x_{t+1}|x_t)$; this would be the standard order 1 Markov model.

An estimator for the probability $P(x_{t+1}|z(x^t))$ is the number of times each symbol in A occurred in each of the Q contexts

$$P(x_{t+1}|z_t)) = \frac{P(x_{t+1}, z_t)}{P(z_t)} \tag{2.11}$$

$$\approx \frac{C(x_{t+1}, z_t)}{C(z_t)} \tag{2.12}$$

where the notation $C(\zeta)$ is the number of times the event ζ has occurred in the past string x^t. Hence, the estimate of the probability distribution for a symbol x_{t+1} in a particular context z_t is the number of times the symbol x_{t+1} occurred in the same context z_t in the past string, divided by the number of times the context z_t occurred in the past string. The approximation is based on the assumption that from one sample to the next, the data generation mechanism does not change much and so the conditional distribution in the past is a good guess for the conditional distribution for the sample to be coded. Note that given the counts $C(x_{t+1}, z_t)$ and $C(z_t)$, there are several methods of estimating the distribution $P(x_{t+1}|z_t)$, some of which are described in Appendix C. The main concerns are that sum of the probabilities in the resulting distribution is unity, as required by the definition of a random variable, and that no symbol or context gets assigned zero probability, as required by statistical data compression. Unless otherwise noted, the non-linear pmf estimator described in Section C.3.1 will be used in all models in this thesis.

To make the previous discussion a bit more concrete let us return to the order 1 model just described. For each symbol in the alphabet we would maintain a count of the number of times that symbol appeared in every one of the A possible contexts. Based on these counts and using one of the estimation techniques we could generate a probability distribution for each symbol x_{t+1} conditioned on the previous symbol x_t.

The Learning Problem

As previously mentioned, the choice of the function σ an important one. Let us consider for the moment only how the model order choice affects the accuracy of the adaptive Markov model constructed from the source. We will assume that σ merely chooses the last n symbols from the past string as the context. This particular function will be called the *identity permutation* of order n.

Consider adaptively modelling a sample generated by an unknown stationary source. We present the same sample to an order 0 model and to an order 1 model. The order 0 model approximates the order 0 statistics

$$P(x_{t+1}|\emptyset) = P(x_{t+1}) \approx \frac{C(x_{t+1})}{t} \tag{2.13}$$

where $\emptyset$ indicates no conditioning information. Similarly, the order 1 model approximates the order 1 statistics by

$$P(x_{t+1}|x_t) \approx \frac{C(x_{t+1}, x_t)}{C(x_t)} \ . \tag{2.14}$$

A histogram bin for an event ζ that occurs with probability $p = P(\zeta)$, counts the number of times the event has occurred in t trials. Hence the count in the histogram is binomially distributed,

$$P(C(\zeta) = k) = \binom{t}{k} p^k (1-p)^{t-k} \ , \tag{2.15}$$

and has an expected value $E(C) = tp$.

For the order 0 and order 1 models described above, the counters $C(x_{t+1})$ and $C(x_t)$ are binomial random variables with probabilities $p = P(x_{t+1})$ and $p_0 = P(x_t)$, respectively, and the counter $C(x_{t+1}, x_t)$ is binomial with $p_1 = P(x_{t+1}, x_t)$. It is significant to note that, despite the fact that the order 1 model approximates the conditional distribution, it must do so by first approximating the joint distribution.

After t samples, the expected values of the counters $C(x_{t+1}, x_t)$ and $C(x_t)$ are tp_1 and tp_0, respectively. Since the value of the counters are integers, it is possible that p_1 is small enough that after t samples the expected value of the counter tp_1 is less than one. In fact, it may be that it takes many more than t samples before the even one instance of the pair (x_{t+1}, x_t) occurs. However, in the same number of samples, it is possible that tp_0 is greater than one. Continuing with this reasoning and remembering that, for any events ζ_1 and ζ_2, $P(\zeta_1, \zeta_2) \leq P(\zeta_1)$, it will always be the case that the histogram bins corresponding to joint probability will never have greater counts than the bins of the corresponding marginals. Because this is the case, the lower order joint probabilities will usually be more accurate than the higher order ones.

But what do we mean by accurate? A measure of the difference between two distributions is their cross entropy

$$D(p_r||p_a) = -\sum_{\zeta} p_r(\zeta) \log p_a(\zeta) \tag{2.16}$$

where p_r is the "real" or actual distribution and p_a is the approximation. If the distributions are identical, the cross entropy is the entropy of the distribution. However, if the distributions are different, the cross entropy is always higher than the real distribution. We of course want the approximation to be as close as possible to the real distribution, so that cross entropy is the same as the source entropy. This means the approximation is doing as well as possible.

In the context of the current problem, let us define then $p_{0,r}$ and $p_{0,a}$ as the real and approximated order 0 distributions and $p_{1,r}$ and $p_{1,a}$ as the real and approximated order 1 distributions. We will assume that the source is stationary so that the real distributions do not change with time. The approximated distributions, however, are recalculated at each time t based on the past symbols. If we can determine the expected value of cross entropy at any time t, we can determine how well a histogram is modelling the distribution at that time, and with this information make a sensible choice when to use the order 0 model and when to use the order 1 model.

If we let $\bar{D}(t) = E(D(p_r||p_a))$ be the expected value of the cross entropy at time t, then the expected value of the *codelength* at time t is given by

$$\bar{L}(t) = \sum_{i=0}^{t} \bar{D}(i) \ . \tag{2.17}$$

As before, the approximation is based on a histogram, and is really a random variable $p_a(\zeta) = \frac{C(\zeta)}{t}$, where t is the number of samples seen so far. $C(\zeta)$ is a binomial with probability $p_r(\zeta)$,

$$P(C = k) = \binom{t}{k} p_r(\zeta)^k (1 - p_r(\zeta))^{t-k} \ . \tag{2.18}$$

Substituting this expression into (2.16) and taking the expected value results in

$$\begin{aligned} E(D(p_r||p_a)) &= E(-\sum_{\zeta} p_r(\zeta) \log \frac{C(\zeta)}{t}) && (2.19) \\ &= -\sum_{\zeta} p_r(\zeta) E(\log \frac{C(\zeta)}{t}) && (2.20) \\ &= -\sum_{\zeta} p_r(\zeta) \sum_{k=0}^{t} \binom{t}{k} p_r(\zeta)^k (1 - p_r(\zeta))^{t-k} \log \frac{k}{t} \ . && (2.21) \end{aligned}$$

The cross entropy approaches the entropy with increasing t. As $t \to \infty$, $E(\log \frac{C(\zeta)}{t}) \to \log \frac{t p_r(\zeta)}{t} = \log p_r(\zeta)$.

The optimal model order, n, that should be used to code the input after t samples have been seen depends on the distribution and t. This author knows of no distribution-independent function for determining the optimal order. Solving this problem is what motivated the development of the variable order techniques described in Chapter 3. For instance, the UMC algorithm tends to use low order models at the beginning of a sample and work up to higher order models as more samples are seen. This works better than just using a high order model because high order models yield uniform predictions (which are bad unless, of course, the underlying distributions are uniform) until enough symbols have been processed.

Another traditional method for dealing with this problem is splitting the input into bit planes and coding them separately (Joint Bi-level Image Experts Group 1992). Due to the smaller alphabet size, the probability is distributed among fewer symbols and the counters for those symbols tend to accurately represent the actual probability more quickly. However, the probability distribution that is being estimated is for the bit plane only, and ignores any relationships between the planes. So, while such models learn quickly, their resulting codelength can be higher than a model that takes into account the relationship between the planes, if such a relationship in fact exists.

Both the variable order approach and the bit plane approach will be extended in this thesis for the specific task of signal data compression. The variable order approach will be augmented by variable resolution conditioning information and the bit plane approach will be extended to planes of higher resolution.

Now that we understand the learning problem in terms of model order, let us consider the learning problem in terms of the resolution of the input data.

Markov Models and Signal Data

As previously noted in the introduction, the current opinion (Williams 1991, page 96) is that Markov models are inappropriate for all but binary signal data. This opinion is based on the very real fact that when we are doing Markov modelling, we are looking for exact matches in the data, having made the assumption that the state of the Markov source can be determined by the past n symbols it emitted. However signal data is the result of quantizing measurements of physical processes. Quantization adds noise, and measurements are inherently noisy representations of physical processes that have infinite resolution. The learning problem described in the previous section is exacerbated because the alphabet size A is essentially increased, but no information is added to the signal. Consider the case of binary data corrupted by white Gaussian noise; for example, if one were to digitize to 256 levels of grey a black and white photograph of a text-only facsimile. Suppose we know that really there are only two levels in the generating mechanism, but due to the measurement we end up with many more, up to 256. Each histogram in each context will thus see correspondingly fewer samples and will take longer to generate accurate probability estimates.

Traditionally, DPCM (Section 3.1.7) has been used in an attempt to separate the input into signal and noise. Only the residual is coded, so that the model only has to deal with information that cannot be easily represented by a linear model. In later chapters we will see that there are other methods, and this thesis presents some new methods.

2.2 Complexity

The following sections take us through the derivation of the minimum description length and stochastic complexity. In the sections that follow, the MDL principle will be seen to be the basis for several good data compression algorithms from which the new techniques described in this thesis are derived.

2.2.1 Minimum Description Length (MDL) Principle

The Minimum Description Length (MDL) principle (Rissanen 1983b) states that the best theory to explain a set of data is the one that minimizes the sum of the length of the description of the theory (the parameters) and the length of the data when encoded using the theory. Li and Vitanyi (1992) derive the MDL principle starting from Bayes' rule:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)} \, , \tag{2.22}$$

where H is a random variable that represents possible hypotheses to explain the data. The distribution associated with H, $P(H)$, is the prior distribution and represents what we know about the truth of each of the hypotheses that form the support of H. D is the observed data that is explained by the hypothesis H. We want to choose the hypothesis H such that the *a priori* probability $P(H|D)$ is maximized. Taking the logarithm of both sides results in

$$-\log([(H|D)] = -\log[P(D|H)] - \log[P(H)] + \log[P(D)] \, . \tag{2.23}$$

The term $P(D)$ is some constant we don't know, and can't know, and can be ignored for the purposes of the maximization. Maximizing $P(H|D)$ is the same as minimizing $-\log[P(H|D)]$ and an equivalent problem is thus to find the minimum of

$$-\log[P(D|H)] - \log[P(H)] \, . \tag{2.24}$$

The first term, $-\log[P(D|H)]$, is readily seen to be the codelength that would be assigned to the data using the hypothesis H. Applying the same logic, the second term would then be the codelength of the hypothesis H. But what does this mean?

In the original approach by Solmonoff (1978), H is, in general, a Turing machine; however, to get a computable result, Rissanen (1986b) assumed that H is some model

$H(\Theta)$ with a set of k parameters, $\Theta = \{\theta_1, \theta_2, \ldots \theta_k\}$, where the number of parameters may vary and influences the codelength of the model description.

2.2.2 Stochastic Complexity

If we replace the hypothesis H with the parameter vector Θ in (2.24), the quantity to be minimized becomes

$$-\log[P(D|\Theta)] - \log[P(\Theta)] \; . \tag{2.25}$$

The second term corresponds to the model cost, and (Rissanen 1983a) determined that to encode the k parameters in an optimal way requires $\frac{k}{2}\log n$ bits, where n is the number of observations in D. The minimization then becomes

$$\mathrm{MDL} = \min_{\Theta,k}\left[-\log[P(D|\Theta)] + \frac{k}{2}\log n\right] \; . \tag{2.26}$$

This form of the minimum description length is also called the non-predictive stochastic complexity, $l_{np}(x)$ (Rissanen 1986b).

To summarize, the stochastic complexity is the shortest codelength for the entire observation sequence relative to a model class and, as such, is an approximation to the algorithmic complexity (Kolmogorov 1965), a more involved notion of which represents the length of the smallest input tape that will produce the string when applied to a universal computer. For the stochastic complexity, the model class is not the set of all possible programs for a universal computer, but rather a set of distributions. This makes the stochastic complexity computable, unlike the algorithmic complexity (Li and Vitanyi 1992).

However, Rissanen (1989) warns that choosing the model family, a step that cannot be automated if the non-computability is to be avoided, has a significant effect on the resulting stochastic complexity. For some problems, the choice is evident, but this is not so for others. On one hand, we would like the model class to be large so that we test many different models; however, the more models we have in the class, the greater the danger that the best model may fit only the sample and not the data source (which may not be adequately represented by the sample). This is similar to the situation where a neural network remembers the training set but cannot generalize. On the other hand, choosing too small a family will result in not being able to characterize the data well, unless we have prior knowledge of the data generating mechanism.

Predictive coding means we model the conditional density for the possible values of the next observation x_{t+1} as

$$f_{k,\hat{\Theta}(t)}(x_{t+1}|x^t) \ , \tag{2.27}$$

where $\hat{\Theta}(t)$ is the k component estimate of Θ based on the string x^t. Since the estimate for the parameters is only based on the past string, the parameters do not need to be coded and sent as side information. Adding all the codelengths for the string x of length n results in

$$L(x|k) = -\sum_{t=0}^{n-1} \log[f_{k,\hat{\Theta}(t)}(x_{t+1}|x^t)] \ , \tag{2.28}$$

which can be minimized with respect to k to get the optimal model order $\hat{k}(n)$.

For each value of k above, the estimate of the best $\hat{\Theta}(t)$ for encoding the next observation is the value that minimizes

$$-\sum_{i=0}^{t-1} \log[f_{k,\Theta}(x_{i+1}|x^i)] \ . \tag{2.29}$$

That is, it minimizes the codelength of the previously seen string. This is based on the hope that the predicted distribution is like it was in the past. This minimization requires an estimate for $\hat{\Theta}(0)$ that is equivalent to the specification of a density function, $f(x_1)$, for the first data point.

The minimized codelength is not complete because the decoder does not know the value k chosen by the coder using (2.28). This parameter can be coded using a prefix code, which requires a length of $\log^* k + \log c_K$ bits (see Appendix D).

The semi-predictive stochastic complexity of the string x is then

$$l_{sp}(x) = \min_k \{L(x|k) + \log^* k + \log c_K\} \ . \tag{2.30}$$

It is called semi-predictive because the estimate for the parameter vector at sample $t+1$ is made based only on the past string, x^t and so does not need to be transmitted. Thus, the term $\frac{k}{2}\log n$ does not appear in this expression. However, the order estimate k is based on the entire string x, and hence needs to be transmitted as side information.

If we let $\hat{k}(t)$ be the value of k that minimizes

$$L(x^t|k) = -\sum_{i=0}^{t-1} \log[f_{k,\hat{\Theta}(i)}(x_{i+1}|x^i)] \ , \tag{2.31}$$

then the predictive stochastic complexity is

$$l_p(x) = -\sum_{t=0}^{n-1} \log[f_{\hat{k}(t),\hat{\Theta}(t)}(x_{t+1}|x^t)] \ , \tag{2.32}$$

with $\hat{k}(0) = 0$ and $\hat{\Theta}(0) = \lambda$, where λ is some initial set of model parameters. In the predictive case, we are given x one sample at a time and we gradually build up the estimates for both the optimal model order and the optimal parameters. Since no side information is needed, predictive stochastic complexity is a particularly useful concept in coding. The predictive stochastic complexity can be shown to be an asymptotic approximation to the stochastic complexity for ergodic sources (Rissanen 1989).

Instead of minimizing based on the entire past, Furlan (1991) lets $\hat{k}(t)$ be the value of k that minimizes

$$L(x^t|k) = -\sum_{i=t-q}^{t-1} \log[f_{k,\hat{\Theta}(i)}(x_{i+1}|x^i)] \tag{2.33}$$

for some value of the history size q. This results in a form of local adaptation.

To make this more concrete, let us consider the special but common case of a Markov source. Given an alphabet of M symbols, an order 0 Markov source is completely defined by its M transition probabilities. This source has only one state, or, equivalently, it has only one context, $\emptyset$. Similarly, an order 1 Markov source, in which the state is defined as the previous symbol, is characterized by M^2 conditional probabilities. For the general order j Markov source with the state defined as the j previous symbols, the source is characterized by M^j conditional probabilities. In all these cases, the parameter vector we seek to estimate is the set of conditional probabilities that characterize the source. For the general order j Markov source, Θ would be a M^j element vector.

Since the definition of the stochastic complexity assumes that a model class has been chosen, the maximum likelihood estimate for the codelength at a particular model order is a known function. For a Markov source, the maximum likelihood estimate for the set of conditional probabilities at a particular state is known to be the frequency each symbol was emitted at that state divided by the number of times the source was in that state.

Chapter 3

Survey of Current Methods

This chapter is a survey of the current data compression methods relevant to this thesis. First the set of methods will be described and then in the following section a comparison of the performance of the methods on the test set (see Appendix A) is presented.

3.1 Description of the Methods

3.1.1 Ziv-Lempel

While the Ziv-Lempel family of compressors (Ziv and Lempel 1978) is in the category of dictionary techniques rather than statistical techniques, it is useful to include it in this discussion because it is currently the most popular method for general purpose data compression.

The algorithm maintains a dictionary of substrings and compresses by transmitting an index into the dictionary instead of the substring itself.

As noted previously, since this method, like many current statistical methods, looks for exact matches in the previously seen input, it is necessarily ill-suited for compression of signal data. This can be seen in Section 3.2.

3.1.2 Fixed Order Markov Modelling

In their landmark paper, Langdon and Rissanen (1981) describe a method for compressing binary images. A binary Markov model is described in which the state is defined as set of pixel values above and to the left of the next pixel, x_{t+1}, to be encoded. If there are M pixels per row, x_t is the pixel just encoded, x_{t-M} is the pixel in the row above, and x_{t-M-1} is the pixel above and left of the pixel just encoded. A seven pixel template is defined as the set of pixels

$$\{x_{t-W-1}, x_{t-W}, x_{t-W+1}, x_{t-W+2}, x_{t-W+3}, x_{t-1}, x_t\} \ . \tag{3.1}$$

The context identifier z is defined as the binary number that results from concatenating these pixel values, resulting in 128 possible contexts. A ten pixel template was also defined, resulting in 1024 contexts. The order of the model implemented corresponds to the number of pixels in the template.

The statistics $P(x = 0|z)$ are maintained for all context identifiers. Pixels are coded using the statistics corresponding to the context identifier of their context.

While the approach in (Langdon and Rissanen 1981) was developed for binary image compression, we are also interested in greyscale image compression. There are several possible methods for splitting a greyscale image into bit planes. The current consensus (Joint Bi-level Image Experts Group 1992) is to first Gray code the input and then pass it to the binary model/coder. We will follow this convention for generating the performance data in a later section.

3.1.3 Prediction by Partial Match for Images (PPMI)

The PPM algorithm (Cleary and Witten 1984) runs $n+1$ Markov models of order 0 to order n simultaneously. The probability estimate for a particular symbol is formed based on a weighted sum of the probability estimates from each model. This technique is called *blending*.

PPMI (Howard and Vitter 1991) is the PPM-inspired image data compression scheme that is closest to the work that will be presented in this thesis[1] but which differs in some significant ways. These differences will be described in a later section.

[1]Incidentally, the work in this thesis was unfortunately largely completed before the author became aware of PPMI.

In PPMI, each pixel in the image is first predicted using a causal order 3 linear predictor. This prediction is subtracted from the pixel and the residual is coded using an arithmetic coder. The probability model used by arithmetic coder is a parameterized Laplacian distribution. The error context is the set of error values in the region surrounding the current pixel, i.e., the error for the pixel above, to the left, and above and to the left of the current pixel. A search is made for matches between this error context in the set of error contexts. If a matching context has occurred an insufficient number of times (the authors chose a default threshold of 12), the search for a matching context is done again but at one bit less resolution. This searching and resolution reduction procedure is repeated until a matching context is found that has occurred frequently enough, at which point the set of resulting error values in that matching error context are used to form an estimate for the Laplacian parameters. Two error contexts are updated, the one that was selected for coding and the one with one more bit of precision than the selected context.

3.1.4 Universal Markov Coding (UMC)

A universal data compression system is one in which "assigning messages in order of decreasing probability to the codewords gives an average codeword length within a constant of the optimal average length" (Bell, Cleary, and Witten 1990).

The Ziv-Lempel algorithm is a powerful example of a universal code, but it does not compress well the important class of strings generated by stationary random *fields* that often arise in image compression. Why is this? Rissanen (1983b, page 658) explains that a good model for an image should take into account the pixels above the current pixel as well as those to the left. For a 512×512 image, the linear parsing of Ziv-Lempel requires that the parse tree grow to a depth of 512 before any dependency on the pixel just above the current pixel appears. Clearly, this is impractical.

Also, since the dictionary can grow without bounds, the input can only be taken in blocks of a certain size. The problem with such block models is that they only capture random dependencies between symbols that fall within the same block. Everything that is learned in the previous block is unavailable to the subsequent block.

The "context" algorithm described by Rissanen (1983b)[2] gathers the contexts in

[2]Williams (1991) assigns this technique the name UMC for Universal Markov Code. A catchy

which each symbol in the source occurs and the associated occurrence counts. The contexts are subsets of the past string, are of varying size, and overlap each other.

What is unique about this algorithm is the sorting function, $\sigma(x^t)$, which permutes the past string x^t into a string z whose elements are in decreasing order of importance to the prediction of the following symbol. Each string z thus defines a unique context. This feature differentiates the algorithm context from block models and allows one to best take advantage of any knowledge of the structure of the input data. For example, the identity permutation of the string x^t is the string $z = x_{t-1}x_{t-2}x_{t-3}\cdots$. This is an appropriate sorting function for a one-dimensional sequence whose autocorrelation function decreases monotonically. On the other hand, a potentially good sorting function for image data, where the image horizontal dimension is W, maps the string x^t into $z = x_{t-1}x_{t-W}x_{t-W-1}x_{t-W+1}x_{t-2}x_{t-2W}\cdots$. This mapping sorts the past pixels into order of increasing Euclidean distance from the current pixel x_t. This idea really goes all the way back to (Langdon and Rissanen 1981), where the term "templates" was used. A template could be considered a type of function that maps the past string into a fixed order permutation of the recent symbols.

The algorithm context stores the information it learns about the source in a tree. (For a detailed explanation of the algorithm, including an example, see Appendix B.1.) Each node s in the tree represents a unique context, and has a set of M counts,

$$\{c(x_1, s), c(x_2, s), \cdots, c(x_d, s)\} \ , \tag{3.2}$$

representing the number of times each symbol x_i was seen in the context s.

Essentially, the algorithm context runs all possible Markov models on the source simultaneously. Starting with the root node, which corresponds to an order 0 model, successively deeper levels of the tree, which correspond to higher order models, are examined.

Each level j of the context tree (i.e., each order j Markov model) assigns a probability

$$P(x|j) = \prod_{i,s} \frac{c(i,s)!(M-1)!}{(c(s)+M-1)!} \tag{3.3}$$

to the string x where

$$c(s) = \sum_{i \in A} c(i,s) \tag{3.4}$$

label, we have adopted it and will use it instead of the original, more ambiguous term "context".

is the number of times context s occurs in x, i is an index that runs through all M symbols, and s is an index that runs through all k_j contexts. The number of contexts, k_j, for each level j depends on the sorting function. As an example, using the identity permutation and an alphabet size of M implies that the jth level has M^j contexts. Note that (3.3) is valid only if we use a Laplacian pmf estimator (see Section C.2.3). The origin of the factorials in the expression will become clearer in the example that follows.

The stochastic complexity of a string with respect to the model class is then given by

$$l(x) = \min_j \{\log P(x|j) + \log^* j + c_K\} \ , \tag{3.5}$$

with $\log^*$ and c_K as previously defined.

So, to choose the optimal model order based on the stochastic complexity, the context tree should be constructed and, for each level j of the tree, the negative logarithm of the probability assigned to the string as in (3.3) should be calculated. Then, using (3.5), the level which has the minimum codelength is chosen as the optimal model order to code the input.

The predictive approach is similar. As each symbol x_t is received, the previously described algorithm is applied to the context tree $T(t-1)$ to find the optimal order to use in the coding of the symbol.

Finally, the same method can also be applied to individual contexts. The model order chosen is the one that best compressed the past symbols that occurred in the specific context, rather than the order that best compressed the entire string. This is because some regular features exist at higher orders, and choosing the overall best order ignores the fact that these higher order contexts are good coders. Motivated by this observation, Rissanen (1986a) chooses the coding node as the highest order context such that the parent codelength is smaller than the sum of the childrens' codelengths. A more detailed description of the coding node choice is given in Section B.1.

Notable in the predictive version of the algorithm is the fact that we do not penalize the codelength based on the model order. This is because in the predictive case both the coder and the encoder can determine the state from the past string.

As an example, let us look at how the string "00101000" is parsed (Rissanen

Table 3.1: Order 0 counts for the string "00101000"

x	$c(0)$	$c(1)$	$p(x)$	$\prod p(x)$
0	0	0	1/2	1/2
0	1	0	2/3	1/3
1	2	0	1/4	1/12
0	2	1	3/5	1/20
1	3	1	2/6	1/60
0	3	2	4/7	1/105
0	4	2	5/8	1/168
0	5	2	6/9	1/252

Table 3.2: Order 1 counts for the string "00101000"

x	$c(0\|0)$	$c(1\|0)$	$c(0\|1)$	$c(1\|1)$	$p(x)$	$\prod p(x)$
0	0	0	0	0	1/2	1/2
0	1	0	0	0	2/3	1/3
1	2	0	0	0	1/4	1/12
0	2	1	0	0	1/2	1/24
1	2	1	1	0	2/5	1/60
0	2	2	1	0	2/3	1/90
0	2	2	2	0	3/6	1/180
0	3	2	2	0	4/7	1/315

1986a). This example is instructive and also relevant, because, to the extent of this author's understanding, Table I in the aforementioned reference is in error.

To begin, Table 3.1 shows the progression of the counts maintained by an order 0 model. The first column shows the symbol, and the second and third columns show the frequency of the symbols "0" and "1", respectively. The fourth column shows one possible estimation of the probability from the counts, namely, the Laplacian estimator described in Section C.2.3. The fifth column shows the product of the probabilities up to and including the current symbol.[3] The codelength is the negative logarithm of this number. Likewise, Table 3.2 shows the progression of the counts maintained by an order 1 model. In Table 3.3, we change to the notation of Table I in (Rissanen 1986a). We assume that the optimal model choice is as shown in column 2. The inverse of probability that would be assigned by the order 0 and order 1 models

[3]The product of fractions with incrementally increasing denominators is the origin of the factorials in (3.3).

Table 3.3: Corrected Table I

x	Optimal order	$1/P(x\|0)$	$1/P(x\|1)$	$I'(x)$
0	0	2	2	log2
0	0	3	3	log3
1	0	12	12	log12
0	0	20	24	log20
1	1	60	$\boxed{60}$	$\boxed{\text{log}50}$
0	1	105	90	log75
0	0	168	180	log120
0	0	$\boxed{252}$	315	log180

are shown in columns 3 and 4, respectively. In column 5, the overall codelength (the predictive stochastic complexity) resulting from this optimal choice is shown under the heading $I'(x)$. The numbers that are in error in the original table are outlined. We see that varying the order of the model has an impact on the overall codelength, and if we choose the order correctly, we can reduce the codelength.

Furlan (1990) presented an modification to the UMC algorithm that works with non-binary alphabets, is locally adaptive, and is less computationally expensive than a straightforward non-binary equivalent to the algorithm in (Rissanen 1986a). This algorithm is described in detail in Appendix B.2.

3.1.5 JBIG

JBIG (Joint Bi-level Image Experts Group 1992) stands for Joint Bi-level Image Experts Group. It losslessly compresses binary image data using an extension of the Q-Coder. The intent of JBIG is to replace the current Group III and Group IV FAX algorithms. Hampel (1992) report that "On images containing text and/or line art, JBIG compression is generally 1.1 to 1.5 that of MMR coding, the most efficient of the G3/G4 techniques. On bi-level images rendering greyscale via halftoning, JBIG's compression ratio advantage typically increases to a factor of 2 to 30."

Like the system described in (Rissanen and Langdon 1981), JBIG determines a context based on some function the pixels in the neighborhood of the pixel being coded, and this context allows a better prediction for the current symbol probability to be made. But unlike the FSM in (Rissanen and Langdon 1981), JBIG also has adaptive templates so that the actual structure of the FSM changes.

JBIG can be successfully used on non-binary images by applying the algorithm one bit plane at a time. However, as with binary run-length coding and binary version of UMC, the pixel values should be Gray coded first. Again, (Hampel 1992) report that with this method, "...compression ratios of at least comparable to those of lossless JPEG coding are obtained. If the intensity resolution is coarse, say, less than 8 bits, JBIG can be significantly better."

Despite the superiority of the algorithm, again, and as with arithmetic coding, the fact that IBM, AT&T, and several other large companies own patents on components of the algorithm, it cannot be used legally without paying royalties. This has hindered the acceptance of JBIG.

3.1.6 Multimodal Data Compression (MMDC)

The multimodal data compression algorithm (MMDC) (Williams 1991) maintains from 1 to `MaxModels` asymptotically adaptive models (the *ordinary models*) and a single locally adaptive mode (the *local model*). Each model generates a prediction for each arriving symbol. The predictions are used to maintain a performance measure for each model.

In MMDC, the selection methodology is to choose the model whose *recent* performance is the best as measured by the entropy of the recent symbols (i.e., the codelength of the recent samples) relative to the models prediction. Given that $p_i(x_t)$ is the probability assigned to sample x_t by model i based on the thus far seen input x^t, and q is a locality parameter, a performance measure is

$$H_{i,t} = -\sum_{j=t-q}^{t} \log p_i(x_j) \ , \tag{3.6}$$

that is, the codelength is summed over the last q samples. If it is undesirable to store k previous entropy values, then the performance measure can be decayed by a factor ρ before updating it. In this case, (3.6) becomes

$$H_{i,t} = -\sum_{j=1}^{t} \rho^{t-j} \log p_i(x_j) \ . \tag{3.7}$$

The parameter ρ corresponds to the half-life of the performance information via the equivalent expressions

$$\rho = e^{\log\frac{1}{2}/h} \leftrightarrow h = \frac{\log\frac{1}{2}}{\log\rho} \tag{3.8}$$

where the half-life means that the effect of the performance of model i at time $t-h$, $\log p_i(x_{t-h})$, on the total performance measure at time t, $H_{i,t}$, is one half the effect of the performance of model i at time t.

At all times there is an *active model* and a *best model*. The active model is defined to be the ordinary model whose performance measure is the best. The best model is defined to be the model (ordinary or local) whose performance measure is lowest. At each step the prediction of the best model us used to code the next symbol.

Each arriving symbol is used to update only the active model and the local model. Every other model generates a prediction for the symbol and updates its local performance measure but it does not alter its parameters.

Whenever the local model performs significantly better than the active model (and hence better than all the ordinary models) a new model is created. If there are already `MaxModels` models, the least recently used model is destroyed so as to free a slot for the new model. The new model starts as a copy of the local model but its parameters are set so that it is asymptotically adaptive.

Whenever a new model is created, it is put on trial for a fixed period called the *trial period*. Only one model can be on trial at a time. New models cannot be created while a model is on trial. If a trial model does worse than any ordinary model, it is destroyed. If, at the end of the trial period, the local model is performing significantly better than the trial model, the trial model is taken to be the local model and a new trial period begins. Otherwise, the trial model is taken off trial and becomes an ordinary model.

MMDC was designed with the particular goal recognizing distinct modes in heterogeneous input streams. Each model really corresponds to a mode. In the sections that follow, the idea of running models in competition will be shown to be closely related to the MDL principle and will be used as a convenient method of implementing structural adaptation.

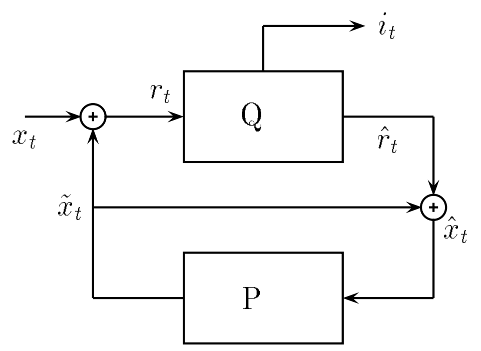

Figure 3.1: DPCM encoder

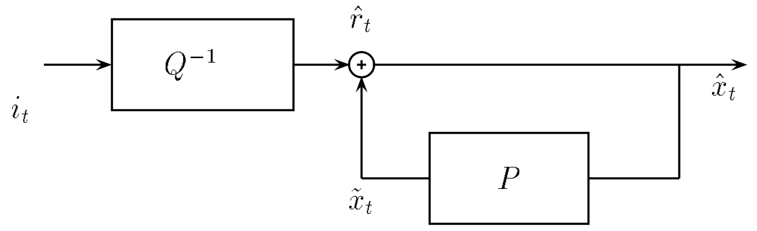

Figure 3.2: DPCM decoder

3.1.7 Differential Pulse Code Modulation (DPCM)

DPCM is a well known technique depicted in Figures 3.1.7 and 3.1.7. In DPCM, a predictor generates predictions based on the past input that are subtracted from the input. The residual sequence is optionally quantized and transmitted.

DPCM is extremely useful in the context of signal data compression because often signal data like images can be quite well represented by linear stochastic processes (Maragos, Schafer, and Mersereau 1984). We can use DPCM to remove much of the lower order process energy in the signal, and use the more flexible Markov algorithms on the higher order residual. In this way, the task of compression is somewhat simplified, in that the residual represents only that which we cannot explain about a signal.

For the class of sources that can be accurately described by an order N linear system driven by white noise, using a trained order N Markov model is equivalent to the order N linear model, but is inefficient with respect to memory consumption. However, for those sources that cannot be accurately described by such a linear model, a Markov model may work better.

It is known that simple uniform quantization followed by an entropy coder can yield a system which, for a given distortion, yields a rate within 0.255 bits of the optimal achievable performance for memoryless sources (Gersho and Gray 1992, page 301). Farvardin and Modestino (1984) have shown this be true even at low rates.

It is easy to determine the characteristics of the residual sequence if the input is an autoregressive source and we model it using a linear predictor. If the linear predictor is the same order as the noise source and its parameters are chosen to be identical to the generating parameter, the DPCM residual will be white noise, the order 0 statistics of which depend on the noise driving the generator. More realistically, when operating correctly, all we know about the DPCM residual sequence is that it is a lower power sequence than the input, and its order 0 statistics in practice are approximately Laplacian.[4]

For lossless compression of signals that are already quantized to r bits, the predictor is modified to generate r-bit predictions, and the quantizer is chosen to be an r-bit uniform quantizer – in effect, it does nothing. For reconstruction, the DPCM loop is run again.

To do lossy coding with DPCM on previously quantized signal data, the predictor can produce infinite resolution predictions, but the quantizer is chosen to have q bits. The distortion introduced results solely from the quantization noise.

3.1.8 Run Length Coding

For sequences in which single symbols are frequently repeated in long runs, run-length coding is a simple and effective compression method. One simply replaces runs of identical symbols with the repeated symbol and the number of times it is repeated. Of course, if symbols do not occur in long runs, its efficiency decreases and can actually expand, rather than compress, an input. Run-length coding is included here because in the realm of image coding, it compares favorably (see Section 3.2) with other far more complicated techniques.

[4]In Section 7.1 we explore the characteristics of the residual for an order 3 linear predictor in the context of joint DPCM/Markov coding.

Table 3.4: Performance on the test set

image	LZ77	LR81	UMC-R	UMC-F	JBIG	DPCM	RLE
`barb`	7.18	5.59	5.35	6.51	6.07	5.57	5.86
`lena`	6.97	5.30	4.84	6.54	5.83	4.92	5.72
`mandrill`	7.25	6.47	6.29	6.57	7.19	6.23	6.42
`pentagon`	6.60	5.45	5.25	5.66	5.98	5.29	5.59
`photog`	6.42	5.25	4.90	6.04	5.85	5.30	5.22
`texture`	6.73	6.10	5.85	6.44	6.67	5.67	6.39

3.2 Performance of the Methods

Some performance numbers for each of the methods described above.

3.2.1 Ziv-Lempel

According to the manual page for the `gzip` utility, `gzip` uses LZ77 as its data compression engine. The data in Table 3.4 result from running `gzip -9` on the images in the test set.

3.2.2 Fixed Order Markov Modelling

An order 1 model based on (Langdon and Rissanen 1981) was coded The results for the images in the test set using the 10 pixel template are shown in Table 3.4 under the heading "LR81".

3.2.3 UMC

UMC Rissanen

In (Rissanen 1986a), an early version of stochastic complexity was used along with the algorithm context to compress text. No explicit numerical results were given, only that "...strings defined by English text can be compressed very well with a reasonable workspace size."

Based on the algorithm in (Rissanen 1986a) a version of UMC was coded. Each image in the test set was Gray coded and separated into bit planes and each bit plane was presented to the model. The results are shown in Table 3.4.

UMC Furlan

Furlan (1991) reports that his version of UMC gives the following results when applied to compression of grey level images:

> When applied to the common grey scale test image "Lena" sampled into 256×256 pixels, one gets a reduction of the 0-order entropy 7.60 bits/pixel to 5.1 bits/pixel. This compares favorably with the result in Ho and Gersho (Ho and Gersho 1989) who obtained 5 bits/pixel by applying a DCT on the same image but sampled at the resolution 512×512 which permits a greater compression than the more coarsely sampled image.

Based on the algorithm in (Furlan 1991) a version of Furlan's UMC was coded and the test set was presented to it. The results are shown in Table 3.4. The `RECMIN` and `RECMAX` parameters were set to -7.5 and 7.5, respectively, and the non-linear pmf estimator was used. These parameters may or may not be optimal for the test set.

The author was unable to reproduce the cited 5.1 bits/pixel on the `lena` image. Possible reasons are 1) the `lena` image used by Furlan is different than the author's 2) the algorithm was improperly coded by the author, 3) the algorithm was improperly specified in (Furlan 1991) or 4) the result in (Furlan 1991) is in error. Option 2 is most likely, although great care was put into developing the code.

Note that the UMC Furlan algorithm could also be applied to the Gray coded bit planes.

3.2.4 JBIG

For the data shown in Table 3.2.4, each image in the test set first Gray coded, then split into bit planes. Each of these was then coded using JBIG and the parameters in Table 3.5. Note that the parameter choice was only one of many possible choices, and may have been suboptimal for the images in the test set. Another factor affecting performance is that on some of the lower significance bit planes, JBIG expanded instead of compressed the input. It would be quite easy for the algorithm to test for this and transmit the original instead if there is no bound on the amount of delay the coder can introduce. However, to make a fair comparison with the other methods,

Table 3.5: JBIG parameters

parameter description	setting
deterministic prediction	enabled
lowest resolution layer typical prediction	enabled
differential layer typical prediction	enabled
number of differential layers	5
lines per strip at layer zero	4
max horizontal offset for adaptive template pixels	8
data ordering	0
lowest resolution layer two line template	0
variable length	0

the bit planes that JBIG expanded were included in the codelength calculation along with the planes it compressed.

3.2.5 MMDC

Since MMDC is a model manager, its performance depends heavily on the models it manages. Further, in its original incarnation, MMDC was designed to deal with data streams that are heterogeneous on a large scale, for example the concatenation of a PostScript file, a UNIX `a.out` file, and a GIF image. Since we are interested in the performance of the models on single images, MMDC is not appropriate and we will therefore not present any performance data for MMDC.

3.2.6 DPCM

The DPCM performance summarized in Table 3.4 is based on order 3 predictor coefficients determined by a pre-scan of the data. The DPCM loop contains a predictor that makes integer predictions, the quantizer is disabled, and the codelength of the residual as determined by an asymptotically adaptive order 0 model is given as the rate. These figures may be different if instead we used and adaptive predictor in the DPCM loop and/or an adaptive order 1 entropy coder for the residual.

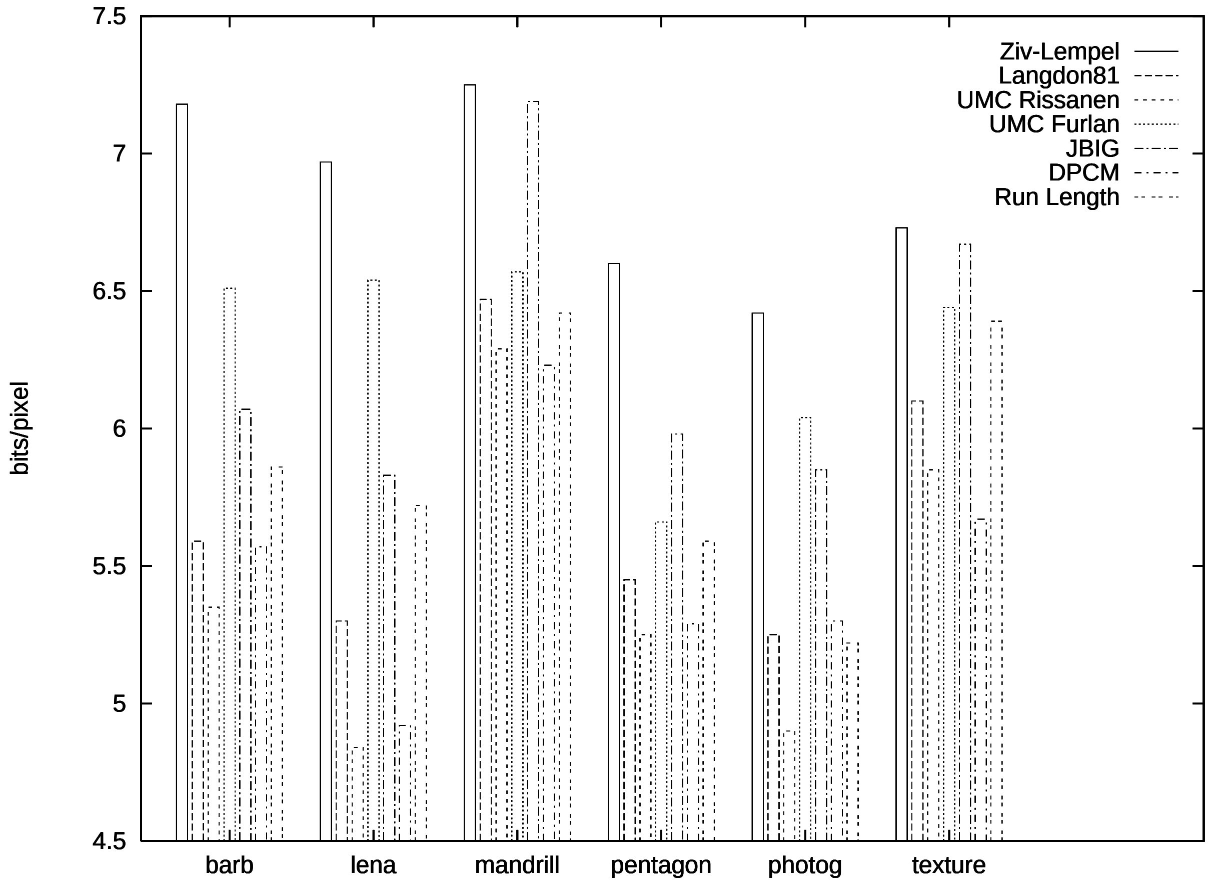

Figure 3.3: Comparison of some traditional methods on the test set

3.2.7 Run Length Coding

For the data in Table 3.4, the data was first Gray coded, then split into bit planes. Each of these was then run length encoded. The run length encoding was performed by writing the run identification (1 bit) followed by the number of samples in the run as a prefix coded integer (see Appendix D).

3.3 Summary

We have seen several approaches to data compression, each with varying success in losslessly coding signal data. A summary of the data is shown in Figure 3.3. In the following chapters we will gain insight on why techniques like coding the DPCM residual and high-order Markov modelling of Gray coded bit planes work so well in comparison to the other techniques by investigating other techniques that attempt to solve the learning problem in Markov modelling.

Chapter 4

Order Zero Adaptation

4.1 Introduction

As stated in Chapter 1, this thesis concentrates primarily on structural adaptation. However, in a variable order model, we still need an order 0 model, that is, a histogram. If the data in a particular order n context is not stationary with respect to its order 0 statistics, then it may be in our best interests to apply some level of context adaptation to the samples recorded in that context. [1]

As described in Section 2.1.2, one method is a simple decay of the histogram counts. However, this is not always to best thing to do. When the data is in fact stationary in the particular context, no decay at all is needed. Decaying the counts would slow down the learning process and increase the overall codelength. This implies that there is room for improvement over fixed decay rate context adaptation. The following sections describe a possible method.

[1]As we will see in later chapters, structural adaptation decisions are based on the performance of the order 0 models in each context. Hence, the kind of order 0 adaptation described in this chapter can affect the operation of structurally adaptive algorithms. For example, it may be the case that using context adaptation results in making structural adaptation decisions that are incorrect because the a context may perform better than it would have if no context adaptation were in effect. However, it is often the case that the data generating mechanism is more complicated than any feasible structural adaptation will allow, resulting in non-stationarities in the individual contexts. It is in these situations where such order 0 adaptation is beneficial.

4.2 Per-Symbol Codelength Sequence

Given a source x that generates a sample x_t for each time t and a model that generates a probability estimate $p(x_t)$ for x_t, the per-symbol codelength sequence is

$$l_t = -\log p(x_t) \tag{4.1}$$

If the model is doing well and if the source is stationary, the per-symbol codelength sequence should be either constant or decreasing, implying that the model is becoming better at predicting the input, and hence can code it more efficiently. If we look at the derivative of the per-symbol codelength, we should therefore see a function that is either negative or zero.

Most real sources are not stationary, however, and so the per-symbol codelength is not as well behaved. Consequently, the derivative takes on both positive and negative values, and these values tell us something about the state of the model. Specifically, if the derivative is above zero, we see that the per-symbol codelength is increasing; the model is doing worse and worse. If the derivative is negative, the model is doing better and better.

We can base our choice of the decay factor on this information. If the model is doing well, we should not decay at all. If it is doing poorly, we should decay in proportion to how poorly it is doing.

Let us consider the case of a non-stationary but still fictitious source. This source consists of the concatenation of two Gaussian noise sources, each consisting of 2^{15} samples and with different means, μ_1 and μ_2. We present this source to an asymptotically adaptive order 0 model. Figure 4.1 shows that at the very beginning of the sample, the model, starting from a zero information case, is learning the first mode of the source. With each sample the model becomes better at coding the source, that is, the per-symbol codelength sequence is decreasing and the derivative is negative as shown in Figure 4.2.[2] This is the case until enough samples have been seen so that the model is no longer changing. The smoothed derivative of the per-symbol codelength stays near zero. Notice however that when the model encounters the second mode, the per-symbol codelength goes up sharply. During this initial period, the new samples

[2]For this figure, the data was strongly filtered for presentation. In the system described in Section 4.4, the filtering includes more information from the derivative of the per-symbol codelength signal, but makes the data unintelligible when presented graphically.

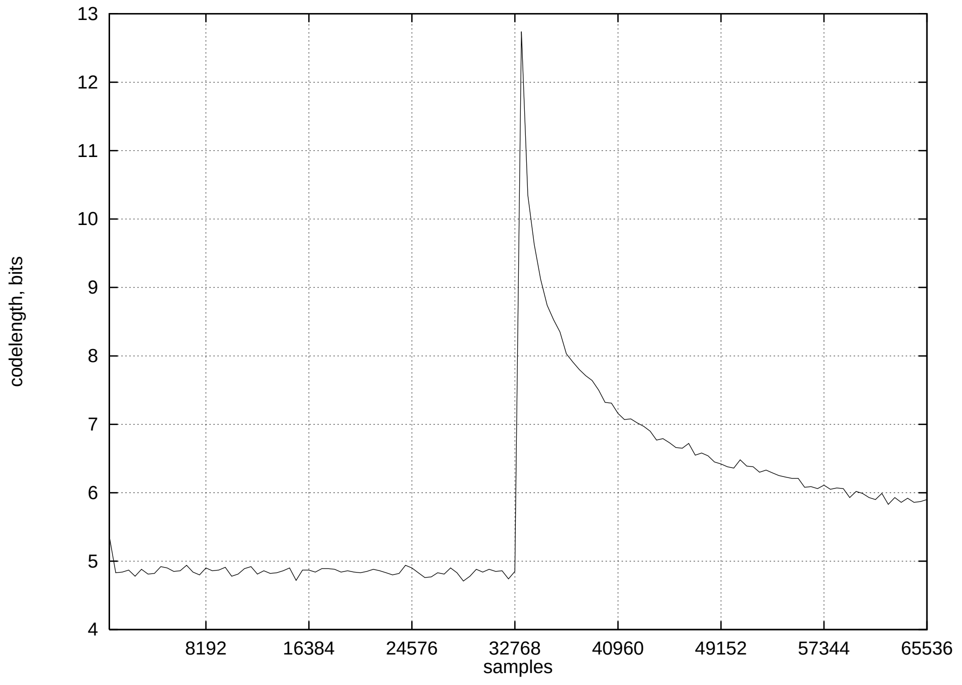

Figure 4.1: Per-symbol codelength of a bimodal source

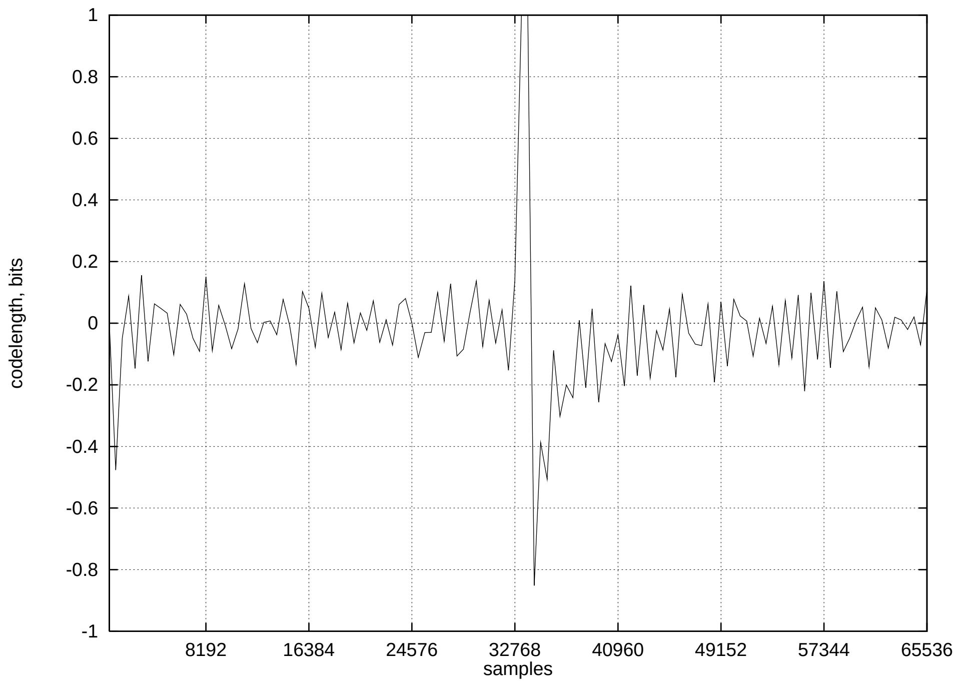

Figure 4.2: Smoothed derivative of the per-symbol codelength of a bimodal source

are serving two purposes 1) to overwhelm the effect of the previously learned sample and 2) to build up a good model of the new sample. The per-symbol codelength increases until such a time that the state of the model begins to represent the source. Then the per-symbol codelength sequence begins to decrease again.

Observing the codelength sequence for this fictitious source is instructive. We could argue that while we are in the first mode, we should not decay because the derivative of the per-symbol codelength is zero or negative, but when we pass into the second mode, we should decay strongly at first, and then gradually less once we have sufficiently counteracted the effect of the first mode. The following sections will explore this conjecture.

4.3 Variable Decay Rate Model

To be more concrete, in a practical system the decay rate at time t should be a function of the smoothed derivative of the per-symbol codelength sequence

$$\delta_t = f(g(l_t)) \ , \tag{4.2}$$

where f is the decay function and g is the smoother and differentiator. The following sections investigate the form these two functions.

4.3.1 Form of the Decay Function

Qualitatively, the requirements for the decay function f are as follows:

- f should map $[-\infty, 0]$ to 1; if the derivative of the codelength is less than zero the model is doing well and no decay is necessary nor is any augmentation of the existing counts desirable

- f should map $[0, \infty]$ to $[\delta_{\text{min}}, 1]$ for some $0 \leq \delta_{\text{min}} < 1$; δ_{min} represents the strongest factor by which the histogram counts are decayed

- f should be monotonically decreasing from 1 to δ_{min} in the range $[0, \infty)$

Requirement 1 uniquely defines the behavior of f in $[-\infty, 0]$ but several functions could satisfy requirements 2 and 3 for the range $[0, \infty)$. One possible choice is

$$f(x) = \delta_{\text{min}} + (1 - \delta_{\text{min}})e^{-px^2}, \text{ any } p > 0 \ . \tag{4.3}$$

In the preceding equation, higher values of p result in stronger decays for the same values of the smoothed derivative.

4.3.2 Form of the Differentiator

We are interested in a function that computes the derivative of the sequence l_t. Following the discussion in (Oppenheim and Schafer 1989, page 96), the time differentiation property of the Fourier transform, $\frac{d^n x(t)}{dt^n} \leftrightarrow (j\Omega)^n \mathcal{F}(x(t))$ leads us to conclude that the ideal differentiator would have a transfer function $H(j\Omega) = j\Omega$. If we assume the input is bandlimited to $|\Omega| < \frac{\pi}{T}$ then the corresponding discrete time differentiator has a frequency response

$$H(e^{j\omega}) = j\omega T \ , \ |\omega| < \pi \ , \tag{4.4}$$

which has an impulse response

$$h(n) = \frac{\pi n \cos \pi n - \sin \pi n}{\pi n^2 T} \ . \tag{4.5}$$

Multiplying the impulse response by a symmetric window of length $M + 1$ centered about $n = 0$ gives us a practical differentiator. Choosing $M = 2$, scaling the magnitude, and shifting to get a causal filter yields the system

$$y(n) = x(n-2) - x(n) \ . \tag{4.6}$$

Fourier analysis of the per-symbol codelength sequence for the images in the test set as analyzed by an order 0 model reveals that it is relatively low-pass, but to guarantee that the sequence is bandlimited, we precede the system with a lowpass IIR filter

$$y(n) = ay(n-1) + x(n) \ , \tag{4.7}$$

so that the overall system has a transfer function

$$H(z) = \frac{1 - z^{-2}}{1 - az^{-1}} \ . \tag{4.8}$$

Returning to our previous notation by setting $y(n) = l'_t$ and $x(n) = l_t$, an approximation of the derivative of the per-symbol codelength is thus given by

$$l'_t = al'_{t-1} + l_t - l_{t-2} \ . \tag{4.9}$$

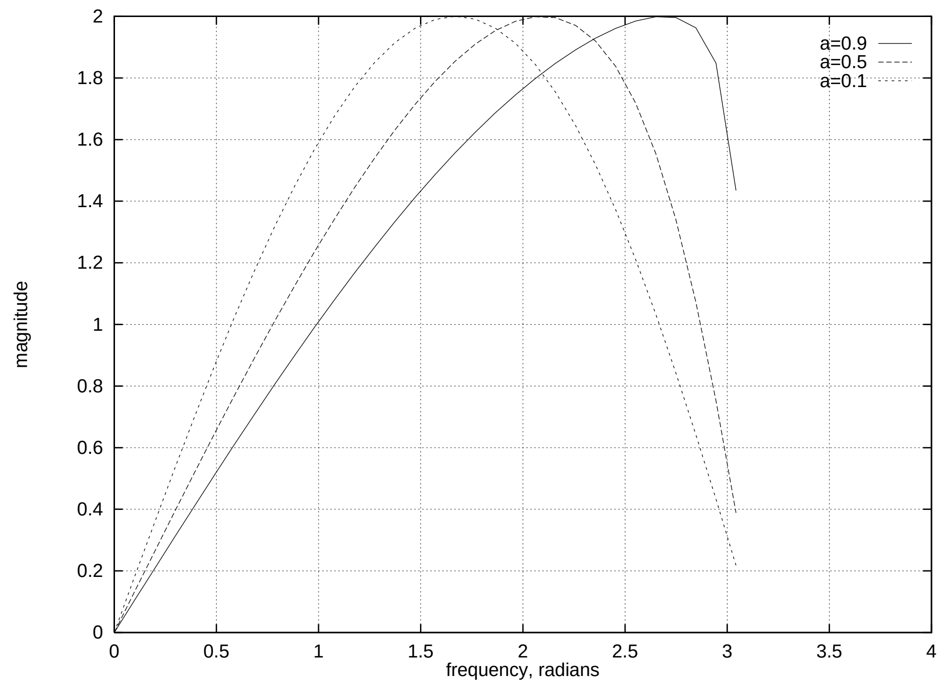

Figure 4.3: Frequency response of the differentiator

The filter should be a good differentiator over the effective frequency of the signal. A plot of the frequency response of this filter for various choices of the coefficient a is shown in Figure 4.3. Such a simple filter is desirable because 1) we may consider employing this order 0 model in a larger system in which it may be replicated tens of thousands of times and a system with little memory and correspondingly low computation will not significantly impact the performance of the algorithms and 2) a filter with a small memory reacts more quickly to changes in the data and in our application a significant delay is undesirable.

4.4 Experiments

For the purposes of the following experiments, we will use (4.3) for the decay function with $\delta_{\min} = 0.90$ and $p = 0.05$, and $a = 0.99$ in (4.9). These parameters were chosen empirically. If the spectrum of the per-symbol codelength sequence is significantly low pass, the choice of a is practically irrelevant as it controls the amount of high frequency information passed to the decay function. Empirically, the performance is affected less by the $\delta_{\min}$ parameter choice than by the choice of p. A full exploration

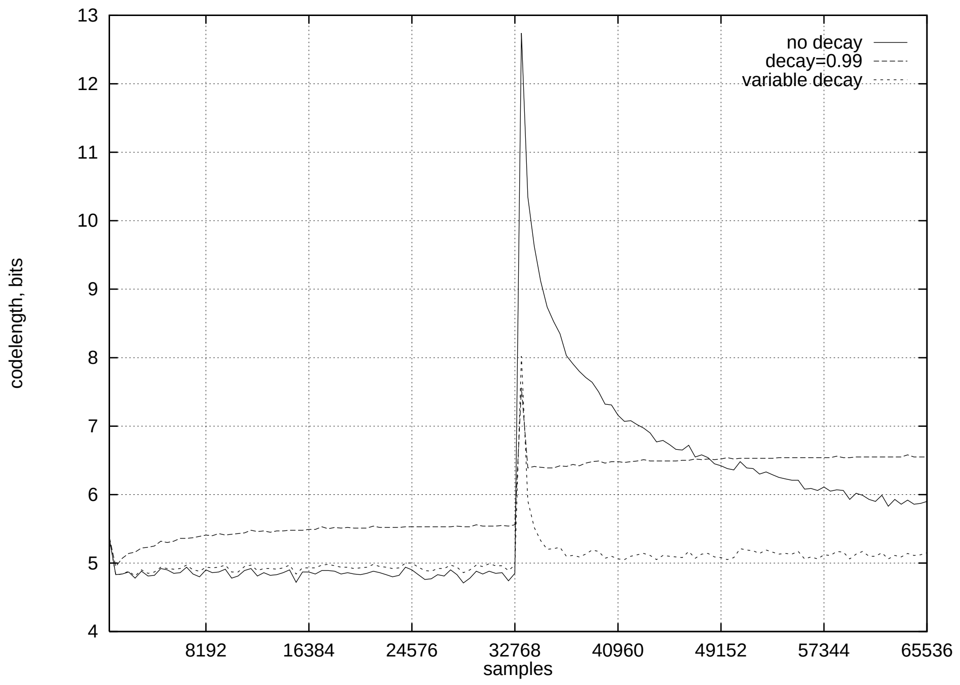

Figure 4.4: Variable decay comparison

of the form of the decay function and its parameters is a subject for future research.

We will begin by examining the performance of the system in comparison to fixed decay methods on the bimodal source presented earlier. In Figure 4.4, we see that the fixed decay model has the desired effect of making the model converge on the second mode statistics quickly, but at the expense of adding to the codelength when no decay is necessary (in the quiescent regions on either side of the boundary between the two modes). Conversely, the fixed decay model learns the source well in the first mode, but takes a long time to recover from the mode switch in the middle of the sample. The variable decay model, on the other hand, does not decay when the model is doing well, and so in the first region, performs exactly like the no decay model. However, at the interface between the first and second modes it decays strongly as it should, but then goes back to normal for the second region. The final codelengths for the no decay, fixed decay, and variable decay models were 5.86, 5.98, and 5.06 bits/pixel, respectively. The variable decay model does clearly better.

We would expect many image sources to be order 0 multimodal by virtue of the fact that different objects in the scene have different intensities and/or colors (otherwise it would be more difficult to distinguish them as objects). Results for the test sources

Table 4.1: Order 0 model with fixed and variable decay

	$\delta = 1.00$	$\delta = 0.99$	variable
`barb`	7.64	7.73	7.45
`lena`	7.47	7.54	7.29
`mandrill`	7.36	7.69	7.32
`pentagon`	6.80	7.27	6.78
`photog`	7.23	7.66	6.88
`texture`	6.81	7.56	6.93

are shown in Table 4.1. Except for the `texture` image, the variable decay model consistently outperformed the fixed decay model with $\delta = 0.99$, and outperformed (although sometimes not by a great margin) the fixed decay model with $\delta = 1.00$ on all sources except the `texture` image, which was the most uniform with respect to its order 0 statistics.

The results for the real images are not as dramatic as for the synthetic source. The modes are smaller due to the fact that the algorithm gets data in a raster scan order while the mode boundaries are two dimensional instead of one dimensional. Imagine that the synthetic source in Section 4.2 was arranged first so that it was a 256×256 image with the first mode occupying the first 128 lines and the second mode the last 128 lines. Since we have not fundamentally changed anything, the results would have been the same. If, however, this same image were rotated 90 degrees, the results would have been less dramatic, since there would be a mode change on every line of the image. Even though there are really only two distinct modes with a single boundary, the boundary is two dimensional and a raster scan sees it as several boundaries.

Of course, it is unlikely that anyone would choose to use an order 0 model on image data anyway. This experiment was chosen simply to demonstrate another method of dealing with non-stationarity in the data. Realistically, this order 0 model would be used in a higher order system, in which non-stationarity becomes harder to visualize than in a simple order 0 system.

4.5 Comparison to UMC and MMDC

There is a relationship between UMC Furlan, MMDC, and the variable decay rate model of Section 4.3. Each uses what is essentially a low pass filtered version of a codelength difference function to determine the system performance.

In UMC Furlan, the value of the relative efficiency counter for models M and N at time t, $R_{MN,t}$, is given by

$$\begin{aligned} r &= R_{MN,t-1} + (l_{M,t} - l_{N,t}) && (4.10) \\ R_{MN,t} &= \max(\min(r, R_{\max}), R_{\min}) \; , && (4.11) \end{aligned}$$

where the max and min serve to constrain R_t within $[R_{\max}, R_{\min}]$. If one removes this constraint and replaces the function with

$$R_{MN,t} = aR_{MN,t-1} + l_{M,t} - l_{N,t} \tag{4.12}$$

for some $a < 1$, making it a low pass filtered version of the differential codelength sequence, approximately the same behavior results.

In MMDC, the performance metric P for model M is

$$P_{M,t} = aP_{M,t-1} + l_{M,t} \; , \tag{4.13}$$

effectively a low-pass version of the per symbol codelength function for model M. The decision that model M is better than model N is based on the relative magnitude of their metrics. That is, if $P_{M,t} < P_{N,t}$ or equivalently if $P_{M,t} - P_{N,t} < 0$, then we choose model M. However, $l_{M,t}$ and $l_{N,t}$ are both applied to the same linear system and then differenced, which is equivalent to applying the difference $l_{M,t} - l_{N,t}$ to that system.

Finally, in the variable decay model, the comparison is applied to the model itself, filtering $l_t - l_{t-1}$ and requiring that the model should forget more when it itself is doing worse and worse.

Hence we see the recurring use of a filtered version of the difference of two per-symbol codelengths as a method for determining coding system performance.

4.6 Summary

We have seen that we can use the derivative of the per-symbol codelength function to make an intelligent choice for the decay rate of an order 0 histogram model. The method performed well both on synthetic and natural sources and better than fixed decay methods. We have also seen that using the derivative of the codelength appears to be a common theme in several successful data compression strategies which, on first inspection, appear to be quite different.

It is expected that the MMDC algorithm running several order 0 models in competition would perform similarly, and may perform better if an order 0 mode encountered and learned early in the source reappeared later. This is the particular strength of MMDC. However, as mentioned previously, we may wish to use a context adaptive algorithm in a larger higher order system in which there would be several thousand order 0 models running. The variable decay model uses strictly less than half the memory of the MMDC algorithm (and even less if there are several distinct modes), while offering reasonable adaptation to non-stationarity.

Chapter 5

Variable Order/Variable Resolution Modelling

5.1 Introduction

As described in Section 2.1.2, signal data presents unique problems for Markov models. The existing variable order techniques that were designed for text compression counteract to some extent the learning problem for Markov models, but still do not give us sufficient control over the number of states in the model. We will see in this chapter that we need not only to vary the order of the conditioning information as we compress signal data, but also its resolution. This chapter begins by determining the optimal resolution/order for some synthetic and natural sources by exhaustive search, and then presents some novel methods for choosing the optimal resolution/order combination without an exhaustive search or even a pre-scan of the data.

Before proceeding we need to introduce the term *state weight*, which is simply the number of order 0 histograms (or the number of contexts) in a Markov model. Define a permutation as an ordered set of n integers as in (Langdon and Rissanen 1981)

$$p = \{p_1, p_2, \cdots, p_n\} \ . \tag{5.1}$$

The source to be coded is given by (as in Section 3.1.4)

$$x^t = x_1, x_2, \cdots, x_t \ . \tag{5.2}$$

We define the order n context of x at t using permutation p as the sequence

$$x_{t-p_1}, x_{t-p_2}, \cdots, x_{t-p_n} \; . \tag{5.3}$$

For the context to be *causal*, each element of p must be greater than zero.

Now let us define a *resolution modification*, m, as an ordered set of n integers in $[0, r]$, where r is the source resolution in bits

$$m = \{m_1, m_2, \cdots, m_n\} \; . \tag{5.4}$$

This modification is applied to the previously defined context resulting in the resolution reduced context

$$(x_{t-p_0} 2^{m_1-r}, x_{t-p_1} 2^{m_2-r}, \cdots, x_{t-p_n} 2^{m_n-r}) \; . \tag{5.5}$$

Note that in performing the resolution reduction, we truncate the fractional part of the individual products. To avoid ambiguity between elements of contexts in which not all the resolution modification values m_i are the same, we will use the notation x/y for the xth element in a y-bit range for $0 \leq x < 2^y$. For example, the context specification $(10/7, 4/5)$ defines an order 2 context, the first element of which is symbol 10 from a 7-bit alphabet, and the second element of which is symbol 4 from a 5-bit range. The element 10/7 in the context specification could, for example, represent symbols 20 or 21 from an 8-bit alphabet. Similarly, the element 4/5 could represent symbols 32 through 39 from an 8-bit alphabet.

Choosing a specific value of n and a specific sequence m results in $N = 2^{\sum_{i=1}^{n}(r-m_i)}$ possible contexts. If we choose to run a model with these contexts, we say that the model has a *state weight* of N. An order n model will usually be denoted by

$$(r_1 = r - m_1, r_2 = r - m_2, \cdots, r_n = r - m_n) \; , \tag{5.6}$$

that is, by the resolution in bits of the conditioning information.

The following four terms will be used to describe the models in this section. The set of *fixed order/fixed resolution* (FOFR) techniques includes those in which a choice of model order n and resolution modification m are made before coding begins and remain fixed for the duration of the sample. In *fixed order/variable resolution* (FOVR) techniques, the order is fixed but the resolution modification m may vary. In a sense, a choice of zero for any element of m effectively reduces the order, but we will still describe the order as "fixed" because it is constrained to some maximum. Techniques like UMC are *variable order/fixed resolution* (VOFR) and, since we have described them in the background and survey chapters, we will just note them here for symmetry. In VOFR models, the resolution of the conditioning information is fixed (usually at the source resolution) but the order can vary and is essentially unconstrained. Finally in *variable order/variable resolution* (VOVR) modelling, neither the resolution modification m nor the order n are specified or constrained.

5.2 Fixed Order/Fixed Resolution Modelling

As the discussion in Section 2.1.2 pointed out, we are faced with a learning problem with all adaptive modelling of data. That is, the model never has the opportunity to analyze all the data until after it has sent all its predictions to the coder and the data is on the channel, at which point it is too late. Hence, decisions have to be made on partial information that may be either helpful or misleading.

Based on the success of the other simplifying methods like splitting the input into bit planes, running variable order models, or treating the input as an autoregressive source, we expect that varying the resolution of the conditioning information in a Markov model will improve the performance.

To test this conjecture, 2^{16} samples were generated using an AR(2) model driven by white Gaussian noise. The samples were quantized to $r = 8$ bits of resolution and translated to fall in the range $[0, 255]$. The parameters for the AR(2) model, $\phi = \{0.01, 0.89\}$, were chosen to demonstrate some properties of the models described later. The correlation sequence for this source is shown in Figure 5.1. An exhaustive set of order 2 models spanning all resolutions for the order 1 and order 2 conditioning information were run on the AR(2) source. That is, we ran the models that estimate the probability distributions

$$p(x_t | x_{t-1} 2^{r_1 - r}, x_{t-2} 2^{r_2 - r}) \tag{5.7}$$

for all $0 \leq r_1, r_2 \leq r$, where r is the source resolution.[1] The zero resolution case ($r_1 = 0$

[1]In this experiment and in all subsequent ones, the pmf estimator used for the individual order 0 histograms constituting the higher order models was the non-linear pmf estimator described in Section C.3.1 as this seemed to consistently produce the lowest codelengths.

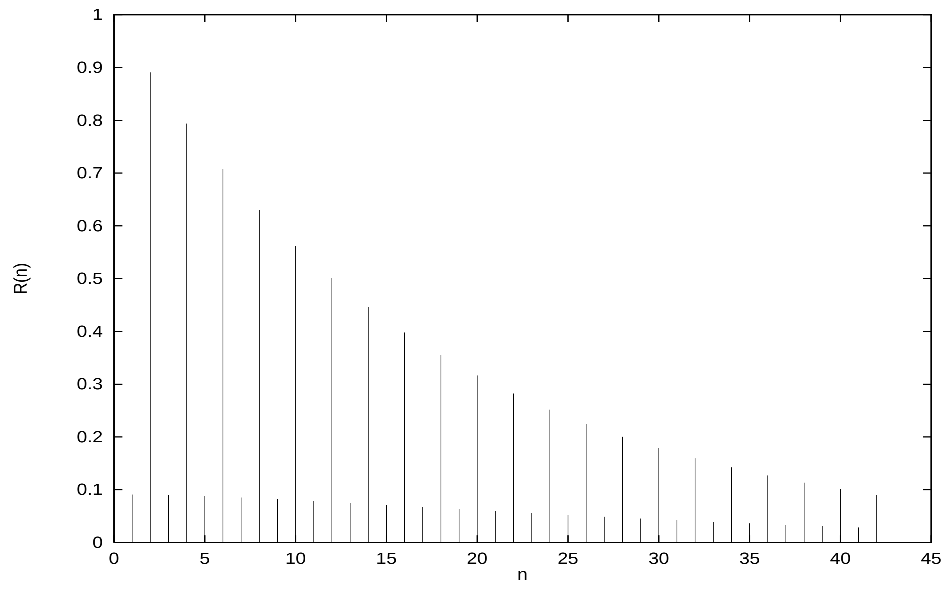

Figure 5.1: Correlation sequence for the $\phi = \{0.01, 0.89\}$ AR(2) source

or $r_2 = 0$) means that all symbols are mapped to the same context (zero), and hence, it corresponds to ignoring that element of conditioning information. Specifically, the $(0, 0)$ model implements the standard order 0 model.

The procedure used will now be described. First, for each sample x_t, the prediction made by each model was saved. Then after the entire sample x had been coded, we went back and analyzed the performance of all models, as judged by the sum of the negative logarithm of the predictions. The model with the lowest overall codelength was called the *optimal static* choice. If more than one model performed equally well, the one with the lower state weight was chosen. The performance of the models is shown in Figure 5.2. The best performance attained by a single model on this source was with the $(0, 5)$ model at 5.19 bits/pixel. That is, the best order 2 model for this sample ignored the order 1 conditioning information altogether, and used 5 bits for the order 2 conditioning information. Referring to the correlation sequence of this test source, this choice makes sense, in that the sample immediately prior to the current sample is not strongly correlated to the current sample, where as the sample two samples prior is. We would expect that, had we run all possible order 4 models, the order 3 conditioning information would likewise be skipped and the order 4 information would be used.

This optimal static choice represents a savings of 21.6% over the $(0, 0)$ model,

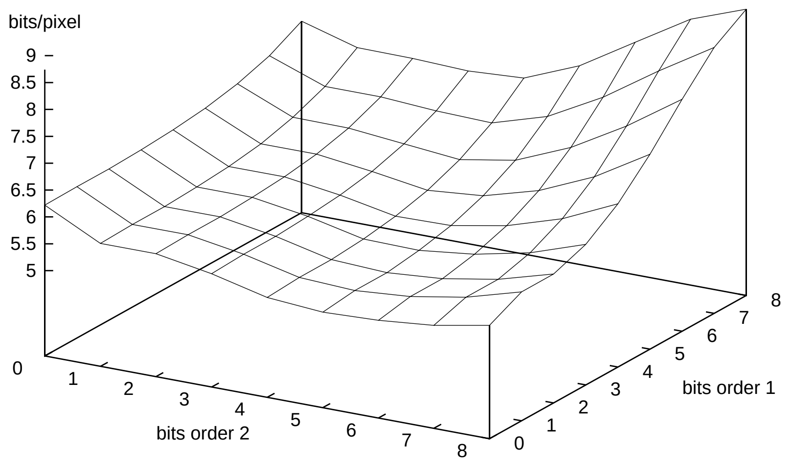

Figure 5.2: Performance of all FOFR models on the AR(2) source

6.4% over the $(0, 8)$ model, 25.6% over the $(8, 0)$ model, and 40.6% over the $(8, 8)$ model. These models were chosen as comparison because they represent the order zero model, a good choice of a full resolution order 1 model, a poor choice of a full resolution order 1 model, and the full resolution order 2 model, respectively. These are models that one might choose to use to model the source if one did not know about the effect of using reduced-resolution conditioning information.

The optimal static order 2 resolution modification was determined for each image in the test set and the performance of the FOFR model is shown in Figure 5.3 in comparison to the order 0 model, the two full resolution order 1 models, and the full resolution order 2 model, as in the preceding discussion. The average minimum improvement over the test set when compared to the $(0, 0)$, $(0, 8)$, and $(8, 0)$ models was 6.2% and the average maximum improvement was 24%. Hence, using reduced resolution contexts is useful for real world sources as well.

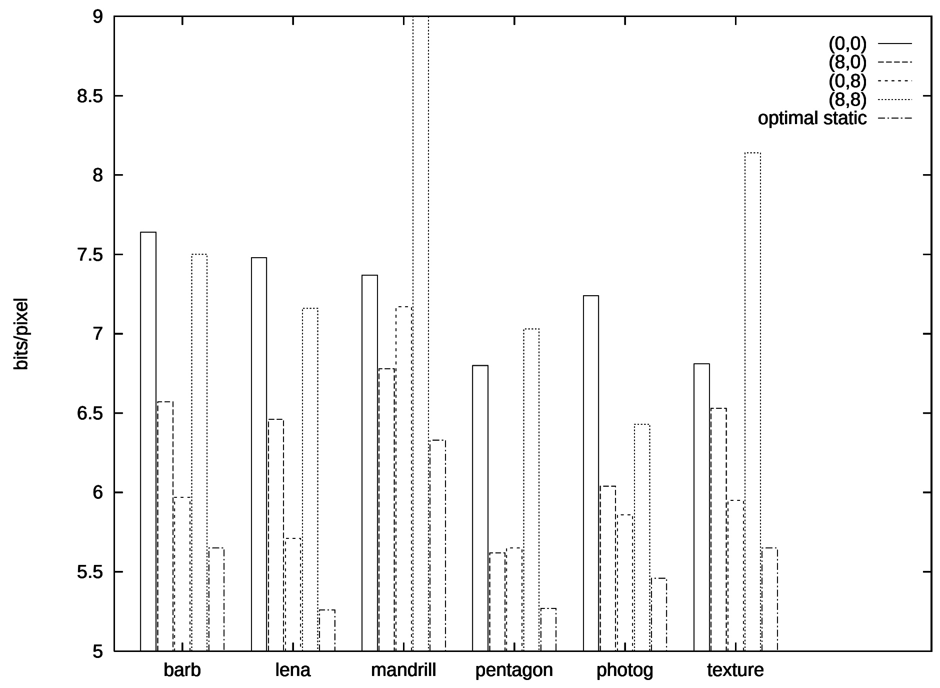

Figure 5.3: Performance of optimal FOFR model on the test set

5.3 Fixed Order/Variable Resolution Modelling I

Based on these results in Section 5.2, a system could determine the optimal resolution via a pre-scan through the data, and transmit as side information the model order and the resolution modification sequence at a very small overhead. But is this fixed resolution model the best we can do? Additionally, is it possible to avoid the pre-scan?

Figure 5.4 shows the performance of some of the models on the AR(2) source presented earlier, where the ordinate indicates the average codelength the model assigned to the last 512 samples. We see that lower resolution models do better in the very early portion of the sample. As more data is seen, the higher resolution models begin to outperform the lower resolution models. This fact, coupled with the success of variable order/fixed resolution techniques, leads us to the conjecture that the state weight of the optimal model choice increases monotonically with the number of samples processed when adaptively modelling a stationary signal source. Further, it tells us that we can in fact do better than making a fixed choice of the resolution modification.

Hence, we would like to initially use low state weight models earlier and then move

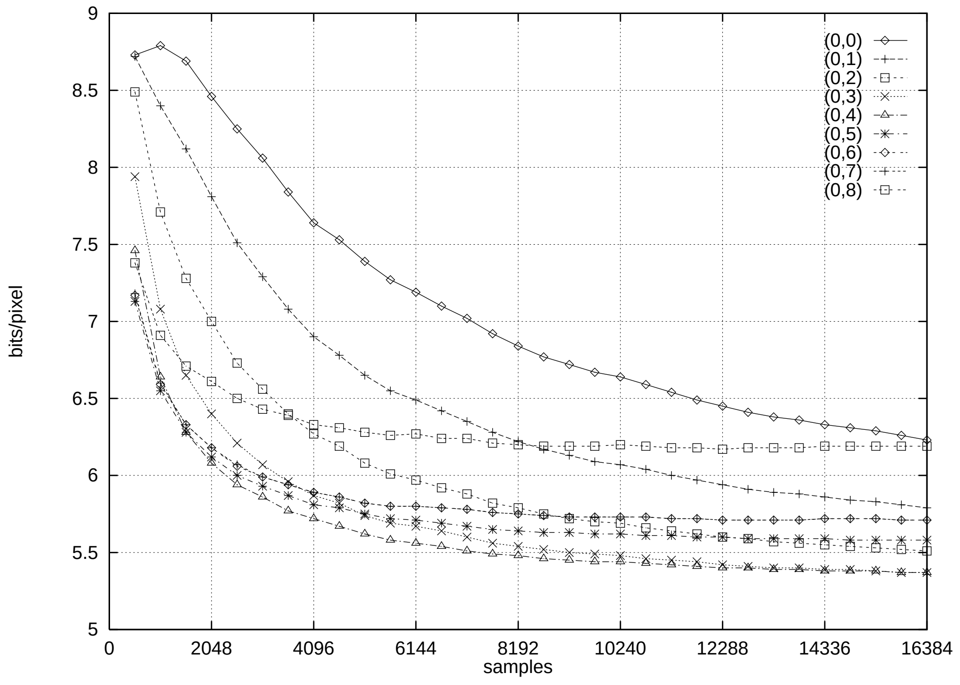

Figure 5.4: Early performance of FOFR models on AR(2) source

to higher state weight models as more data is seen, choosing the resolution in such a way as to optimally exploit the correlation in the sample. One way of doing this is to run several models in competition and to choose between them based on their recent codelength as in the MMDC algorithm of Section 3.1.6.

For the FOVR-I algorithm we choose an arbitrary maximum order n. At the initialization, all $(r+1)^n$ models are created, one model for each possible combination of $r+1$ possible resolutions taken n at a time. As in the MMDC algorithm, each model maintains a recent performance measure (see (3.7)) and, at each step, the model with the lowest recent codelength is chosen as the coding model. However, unlike the MMDC algorithm, no new models are created, all models are asymptotically adaptive, and all models are updated on every sample. As a final modification, in the case where several models are performing equally well, the one with the lowest state weight is chosen as the coding model. For the data in Figure 5.5, the AR(2) source was presented to the FOVR-I algorithm. The `PerformanceDecay` parameter ρ was set corresponding to a half-life of 128 samples and the `MaxOrder` was set to 2. The numbers in parenthesis indicate the resolutions chosen. For example, the notation (14) indicates that the order 1 conditioning information resolution choice was 1 bit

Table 5.1: Parameters for the FOVR-I algorithm

Parameter	Description
MaxOrder	the order n of the models in the system
PerformanceDecay	the half-life h for the performance calculation

```
(00).(10)(11)(00)..(60)(70)(80)...............(70).....(68)(66).(63).....(64)(6
5)....(63).....(65)...(63)..(65)........(67)........(66)......(65)(66)(78).....
........(88)...(00)............................................................
...............................................................................
......(03).....................................................................
...............................................................................
(01)...........................................................................
....................................(03)........................(01).........(0
3)(04)(01).....................................................................
......................................................................(02).....
..............................(03).(02)........................................
................(03)(04).(02).(04)(02)...........(01)..........................
.................(04)................................(01)......(04).........(01
)(04)........(01)..(02).(04)...(02)............(04)............................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
......................................................................(05).....
......................................(04).....................................
...............................................................................
...............................................................................
.........(05)..(04)............................................................
...............................................................................
...............................................................................
.................(05).................(04)...(05)(04)..(05)..(04)..(05).......
................................................(04)...........................
.....................................................(05)......................
..........................................................(04)...(05)(04)......
.........................(05)..(04)(05).(04)...................................
...............................................................................
....................................(05)...(04).(05)......(04).........(05).(0
4).............................................................................
...............................................................................
..........................................................(05)(04).(05)........
...............................................................................
...............................................................................
...........................................................(04)(05)(04).(05)...
............(04).................................(03)...(04)...(05)...........
.................................................(04)..........................
...............(05).(04).(05)..............(04)....................(05)(04)(05)
...............................................................................
...............................................................................
...............................................................................
...............................................................................
..............................................(04).(05)......................(0
4)(05).........................................................................
...............................................................................
...............................................................................
.......................................................................(04)....
..............................(05).....(04)...............(05)(04).............
.......................(05)....................................................
```

```
.............(04)..............................................................
...............................................................................
...............................................(05)..(04)(05)..(04).(05).(04).(
05).....................(04)..(05).(04)......(05)(04).(05)..(04)..(05).........
(04)(05)..............................................................(04).....
...............................................................................
...............................................................................
.........................(05).(04).(05)..(04)(05)........(04)(05)..............
...............................................................................
.............................(04)(05).......(04)...............................
...............................................................................
..............................................(05)....(04).....................
...............................................................................
............(05)......(04).....................................................
.(05)...(04).......................................(05).(04)...................
..................(05).............(04).(05)(04)..(05)(04).....................
.................................................(05)(04).(05)..(04)...........
.....................(05)........(04)...........(05)........(04).(05)..........
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
..............(04)(05)(04)....(05)..........(06)........(04)........(06).......
....................................................................(05).......
(04)..........(05)..................................(06).......................
.................(05)..........................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
................................................(24)...........................
.........................(05).........(24).(05)................................
...............................................(04)............................
.......(05)........................(04)............(05)....(04)..(05)..........
...............................................................................
..................................................(04)...(05)..................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
..............................................(06).............................
(05)...................................(06).(05).....(06)......................
........................(05).....(06)(05)......................................
......(06)(05).................................................................
..........(04)...........(05)..................................................
..(04).................................................................(05)....
..........................(04)(05).................(04)........................
....................................................................(05).......
...........................(04)..(05).......(04)(05).............(04)(05)......
.......................................................(06)(05)...(06).........
.................................(04)(05)...........(04).......................
..(05).........................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...................................................(06)(05)....................
```

Figure 5.5: Early FOVR-I model choice for AR(2) source

and the order 2 conditioning information resolution choice was 4 bits. The dots in the figure indicate strings of identical resolution pair choices, each dot corresponding to one sample and the number of samples seen increases left to right and top to bottom. Observing the resolution choice, we see that after a few tens of samples, during which the resolution choices were chaotic, the model began chose the lower resolution models earlier and moved to the higher resolution models later. Note also that it very rapidly converged (within around 3000 samples) on the order/resolution choice determined for this sample in the previous section, resulting in an overall codelength of 5.18 bits/pixel. The difference in codelength compared to the best choice (5.19 bits/pixel) is not significant. Since the algorithm converged so quickly on the choice determined by a pre-scan, less than 5% of the samples were coded under a model other than that choice. Hence, when it is desirable to avoid a pre-scan of the data or send side information, this system would meet the requirement.

One problem with this approach is that some subset of these models may never be chosen at all, and yet (a lot of) resources will be consumed updating them. For the AR(2) above source, only 24 out of the 81 models were ever used, and only 12 were used after around 1000 samples. While this has no adverse effect on the coding performance, it does limit it applicability. This problem is addressed in the next section.

5.4 Fixed Order/Variable Resolution Modelling II

Suppose we again choose an arbitrary maximum order n but this time we begin modelling our signal source using just the order 0 model defined by

$$(r_1, r_2, \cdots, r_n) = (0, 0, \cdots, 0) \ . \tag{5.8}$$

Since it is our conjecture that the state weight of the optimal model choice increases monotonically with the number of samples seen, a good guess for the next optimal model is the one that has an incrementally higher state weight than the current model. However, having chosen the arbitrary maximum order n, there are in fact n such models, namely $(1, 0, \cdots, 0), (0, 1, \cdots, 0), \cdots, (0, 0, \cdots, 1)$, each of which has a state weight of 2. To be thorough, the algorithm should create these n higher state weight models and run the order 0 model in competition with them to determine when to change to a model with a higher state weight. Assuming that there exists some redundancy in the signal to be exploited, one of the n models, will be chosen, most likely the one that uses the conditioning information at the offset with the highest correlation with the current symbol being coded. Once the algorithm selects this model, it should again create and run in competition those models that have an incrementally higher state weight compared to the selected model.

From that starting point let us define the FOVR-II algorithm as follows. As with the FOVR-I algorithm, each model in the system needs to maintain a local performance metric. During the initialization, the order zero model is created and is designated the *best* model. Thereafter, the following steps are repeated:

1. read next sample and code it with the *best* model
2. update the performance metric for all existing models
3. generate list of models performing the best by comparing the performance metrics of all existing models
4. choose the one model from list with lowest state weight and designate it the *best* model
5. for all models on the list, grow "child" models with incrementally higher state weight
6. if there are more samples go to step 1 else quit

Table 5.2 summarizes the parameters for the FOVR-II algorithm.

A model is allowed to grow if it is found to be the model that performs the best at the current instant t. If there are several that are performing identically, all are allowed to grow.

Model growth is defined as creation of the n models with incrementally higher state weight than the parent model. Thus the n children of modeler $(3, 2, 5)$ would

Table 5.2: Parameters for the FOVR-II algorithm

Parameter	Description
MaxOrder	the order n of the models in the system
MaxModels	maximum number of models in the system before growth stops
PerformanceDecay	the half-life h for the performance calculation
MemoryUsage	total algorithm memory usage limit

be $(4, 2, 5)$, $(3, 3, 5)$, and $(3, 2, 6)$. Of course, the maximum resolution of the models is limited to the source resolution. According to this definition, a node could have been created from several different parents. All of these newly created models have the same state weight, but could be anything from order 1 to order n (in the conventional understanding of the term) because the components with resolution zero are skipped.

Since the models are not all created at the same time, the algorithm should maintain a history of all the past samples to train each newly created model.

A model is destroyed when there are `MaxModels` models currently being run and a growth is occurring, or when the algorithm has exceeded the `MemoryUsage` limit. In either case, the least frequently used model is destroyed to make space for the new model. Note: This has implications on the coding of non-stationary data; a discussion follows in the next subsection.

Because any given model may be considered the child of several different parents, and because models may be destroyed if they do not perform well, we must make sure somehow that we do not recreate models that have been destroyed.

The evolution of the model for the first 4096 samples of the AR(2) source described above is shown in Figure 5.6 in the same manner as in Figure 5.4. For this experiment, the `PerformanceDecay` parameter ρ was again set corresponding to a half-life of 128 samples. The maximum number of models, `MaxModels` was set to 128, the `MemoryUsage` to 16 megabytes, and the order of the models, `MaxOrder`, was set to 2. We see that, like the FOVR-I algorithm, the model converges rapidly on the optimal resolution choice. However, when potentially good models need to be created and models that are doing poorly are in the way, the poorly performing models are destroyed (for example, the (20) model was destroyed to make room for the (07) model).

The overall codelength using the FOVR-II model on the AR(2) source was 5.19

```
(00)
children of (00):
        (10)
        (01)
................................................................................
children of (10):
        (20)
        (11)
(10).................(00)...................
children of (01):
        oops! already exists...
        (02)
(01)...............................................(00)(01).....................
...............(00).......(01)..................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
...........................................................................
children of (02):
        (12)
        (03)
(02)....................................................(01)..............(02)..
...........(01).........................................(02)....................
................................................................................
................................................................................
................................................................................
................................................................................
...............................................................................(
01).......(02).........(01).....................................................
............................................(02)...............(01).............
...............
children of (03):
        (13)
        (04)
(03)............................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
............................................................................
children of (04):
        (14)
        (05)
(04).....................(03)....(04)...........................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
```

```
................................................................................
................................................................................
................................................................................
...............................................................
children of (05):
        (15)
        (06)
(05)..................................................(04)......................
................................................................................
................................................................................
................................................................................
................................................................................
.................(05)...........................................................
................................................................................
................................................................................
................................................................................
......................(04)......................................................
................................................................................
..............................................(05)..(04)......(05).(04)......(05
)(04)..(05)(04)..(05)...........................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
..........(04).(05)(04)...........(05)..........................................
................................................................................
................................................................................
................................................................................
................................................................................
.........
children of (06):
        (16)

killing (20)...
        (07)
(06).(05).......................................................................
................................................................................
................................................................................
................................................................................
................................................................................
........
```

Figure 5.6: Early FOVR-II model choice and growth for AR(2) source

bits/pixel which again is the same as the optimal static $(0, 5)$ model.

This algorithm has the advantage that, given that our assumptions about stationary sources are true, only the models that need to be created are created and updated. Fixing an upper limit on the number of models gives us some control on the resource usage of the algorithm, but potentially at the cost of coding performance.

5.4.1 FOVR Modelling and Non-stationarity

The FOVR-II model is expected to perform poorly on non-stationary sources due to the fact that the model growth is always in the direction of increasing state weight, and that, depending on the values of `MaxModels` and `MemoryUsage`, the lower state weight models may have been destroyed.

Let us return briefly to the initial experiments of Section 5.2. Instead of the stationary AR(2) source, let us instead concatenate two AR(1) sources with different statistics. The first is an AR(1) source with $\phi_1 = 0.9$ and the second is an AR(1) source with $\phi_1 = -0.9$, but after generating the source, a non-zero offset was added. Hence the conditional probability distributions for orders greater than zero will be identical but the order 0 statistics will be different as will the odd order conditional probability distributions because the sign of the correlation of the two sequences at odd orders are inverses.

In Figure 5.7, the model choice is shown starting with a few hundred samples before the interface between the modes. Right before the boundary, the algorithm had determined that the resolution choice at that point was to use 4 or 5 bits, but right after passing through a non-stationarity, the algorithm began choosing the lower state-weight models, which have a tendency to learn (and un-learn) more quickly. Thereafter, when the higher state-weight models had seen enough samples again to be reliable, they began to be chosen.

A possible modification for to the FOVR-II algorithm would be to note the derivative of the per-symbol codelength as in Chapter 4. If the derivative is negative, the algorithm is doing well. However if the derivative of the codelength goes positive, it indicates that a non-stationarity has been encountered. Using a context-adaptive order 0 model is not enough in this case because the n currently existing models, even if they were to forget what they had learned from the previous input, may not be of

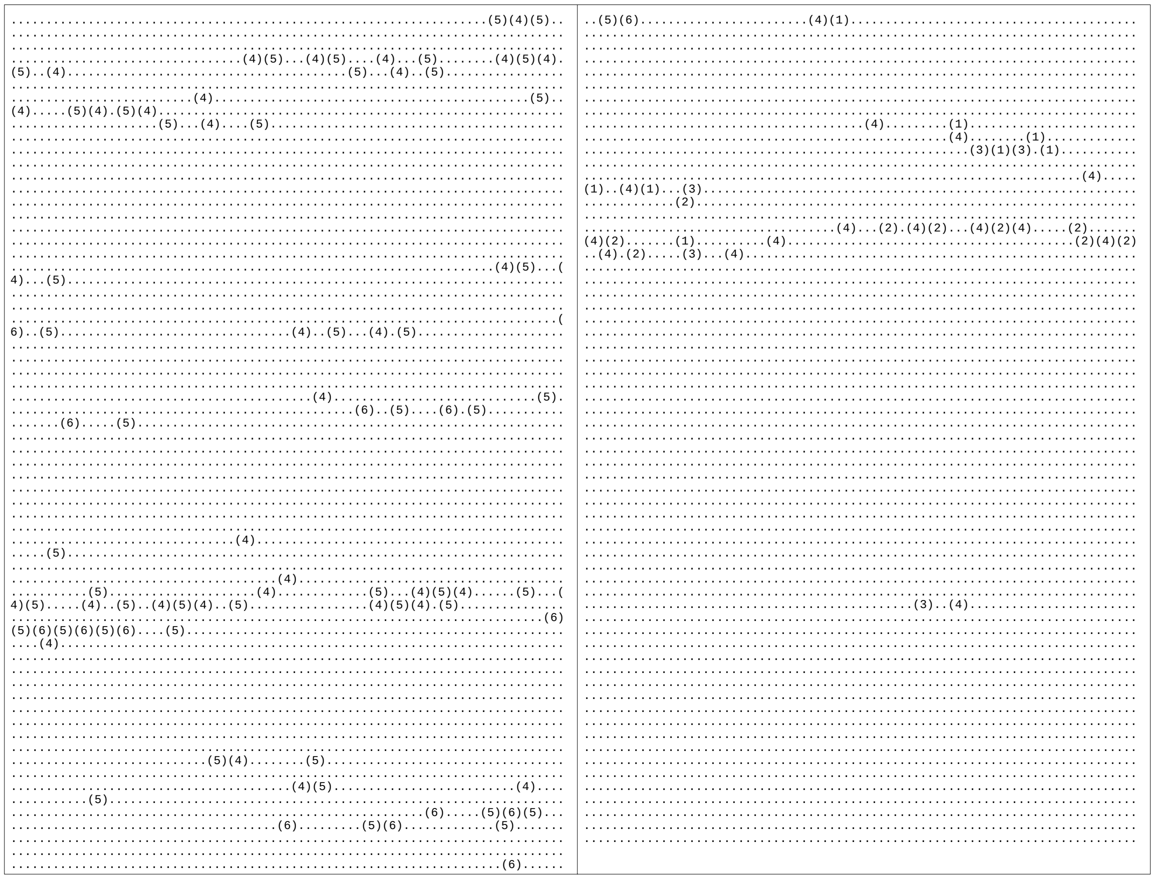

Figure 5.7: Early FOVR-I model choice for a bimodal AR(1) source

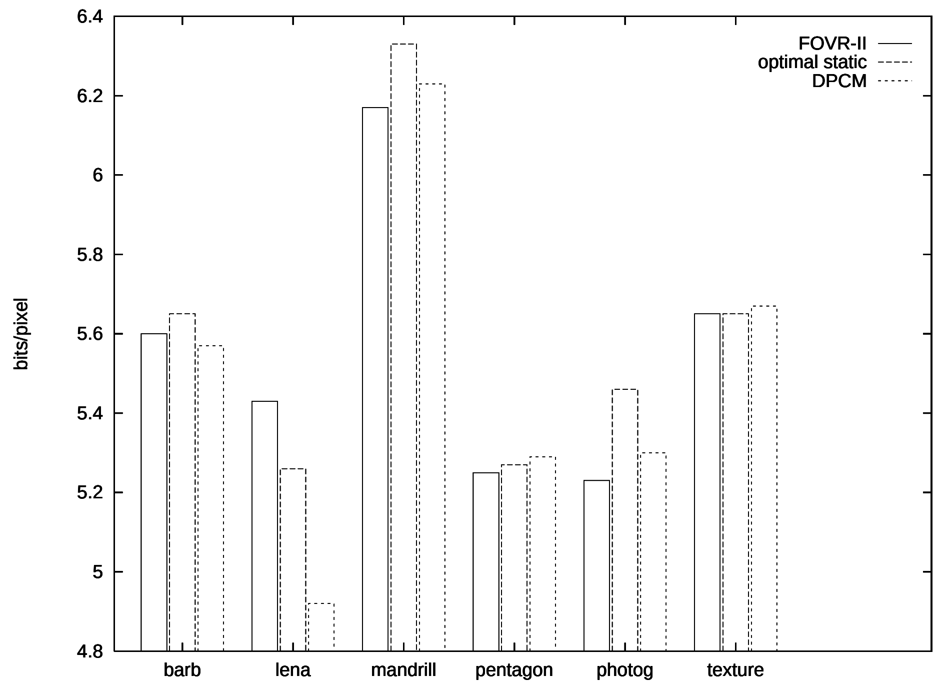

Figure 5.8: FOVR-II performance on the test set

the correct resolution and order. In fact, lower state-weight models would need to be created. This is a topic for further research.

5.4.2 Experiments

The performance of the FOVR-II model on the test set is shown in Figure 5.8. The parameters were the same as in the previous experiment: `PerformanceDecay` was set corresponding to a half-life of 128 samples, `MaxModels` was set to 128, `MemoryUsage` was set to 16 megabytes, and `MaxOrder` was set to 2. For comparison, the DPCM coder performance from Section 3.2 is included. Despite our initial hesitation about the performance of the algorithm on non-stationary sources, the FOVR-II algorithm performed well overall on the test set, resulting in codelengths below the statically chosen optimal value for all sources except `lena`, for which the codelength was approximately 3.2% higher. Interestingly, this is also the only source for which the DPCM model outperformed the FOVR-II model.

In summary, the fixed order/variable resolution model chooses low state weight models first, grows to higher state weight models, and reaches some limiting value of the state weight (which is the same as determined by exhaustive search) while systematically avoiding conditioning information that is not useful. In this way the overall performance is optimized to some extent.

5.5 Variable Order/Variable Resolution Modelling

The previously described methodology for variable resolution/variable order coding could be described as "coarse-grained" in that the entities that are run in competition are the full models, and so the performances being compared include the performance of the model in all contexts. Based on the success of UMC, which instead compares order zero models corresponding to individual contexts, we speculate that perhaps even better performance could be gained by modelling at this finer level. Hence, the task is to define an algorithm that takes into account that models with fewer states tend to give reliable estimates faster than models with more states and that operates on the context level. The VOVR algorithm that is described in the following section is inspired by Furlan's (1990) version of UMC and can be thought of as UMC with variable resolution extensions.

Let us begin defining the VOVR algorithm. Like UMC, the algorithm manages a tree structure. Each node in the tree corresponds to a context and can be uniquely identified by a k-tuple of pairs

$$(c_1/r_1, c_2/r_2, c_3/r_3, \cdots, c_k/r_k) \tag{5.9}$$

that defines a path from the root to the node. Recall that here the notation (x/y) does not indicate division but rather the xth element from a y-bit range. The root is the order 0 model or the null context $(\emptyset)$. For example, if $r = 8$ bits and the past few symbols seen were

$$\cdots, 15, 45, 100, 122, 112 \tag{5.10}$$

then all possible order 2 contexts (assuming the identity permutation) consistent with the history are

$$(112 \cdot 2^{r_1 - r}/r_1, 122 \cdot 2^{r_2 - r}/r_2)\ , \tag{5.11}$$

for any $0 \geq r_1, r_2, \geq r$. When r_1 or r_2 are equal to zero, then all symbols get mapped to the same conditioning information, and the context is skipped.

We call two contexts C_1 and C_2 *common* if, via a resolution modification m, either context C_1 can be changed to context C_2 or vice versa. Continuing with the previous example, one of the possible order 2 contexts consistent with the history is the context $(14/5, 122/8)$. Via successive resolution modifications to the second element in the context, we can generate the contexts $(14/5, 61/7)$, $(14/5, 30/6)$, $(14/5, 15/5)$, $(14/5, 7/4)$, $(14/5, 3/3)$, $(14/5, 1/2)$, $(14/5, 0/1)$, and $(14/5, \emptyset) = (14/5)$, which is effectively an order 1 context. Similarly, applying successive resolution modifications to the first element in the context, we can generate the contexts $(7/4, 122/8)$, $(3/3, 122/8)$, $(1/2, 122/8)$, $(0/1, 122/8)$, and $(\emptyset, 122/8)$, which is effectively an order 1 context. The other possible resolution reduced contexts are determined similarly. All these contexts are common to $(14/5, 122/8)$.

Conversely, any context C_2 that could be subjected to a resolution modification to produce C_1 is common to C_1, including specifically all higher order contexts. In our example, the contexts $(28/6, 122/8)$, $(29/6, 122/8)$, $(56/7, 122/8)$ through $(59/7, 122/8)$, $(112/8, 122/8)$ through $(119/8, 122/8)$, and all contexts with order higher than 2 whose first two elements are $(14/5, 122/8)$ are all common to the order 2 context $(14/5, 122/8)$.

The reason for defining common contexts in this way is that common contexts all have the possibility of coding the same source context and thus must be compared.

Each node has an order 0 histogram (which may or may not be adaptive) that maintains frequency counts for the number of times each symbol in the alphabet A was seen in the context corresponding to the node. Each node also has a *performance list*, which is a list of relative efficiency counters in the style of Furlan (1990) that compare the performance of the node to all nodes that have a common context. The relative efficiency counter (REC) for two nodes is updated so that it represents cumulative difference in codelength between the two nodes for all symbols in the thus far processed sample that both of them could have coded.[2]

[2]As a matter of implementation, since each node is a pointer to a structure, the "direction" of the REC, that is, whether we add node1's codelength and subtract node2's, or vice versa, is determined by the magnitude of the node pointers. If $p_1 > p_2$, then we add node1's codelength to the REC and subtract node2's; if $p_1 < p_2$, then we add node2's codelength to the REC and subtract node1's. Hence, if $p_1 < p_2$ and the REC is greater than zero, we know that node1 is doing better. Similarly, if $p_1 > p_2$ and the REC is greater than zero, we know that node2 is doing better. Obviously, if the node pointers are equal, we are addressing the same node and we will not compare the node to itself. We have to be careful that we update the REC only once per sample. This can be handled easily

Table 5.3: VOVR node selection logic

fact	meaning	implication	choices remaining
initially	no information	no implication	012345
$R_{13} = 5$	1 is better than 3	3 can't be the best	012 45
$R_{25} = -1$	5 is better than 2	2 can't be the best	01 45
$R_{01} = -4$	1 is better than 0	0 can't be the best	1 45
$R_{12} = -8$	2 is better than 1	1 can't be the best	45
$R_{42} = -2$	4 is better than 2	2 can't be the best	45
$R_{54} = 1$	4 is better than 5	5 can't be the best	4

Every node can have up to $\sum_{i=0}^{r} 2^r = 2^{r+1} - 1$ children. For example, the node (14/5) has two 1-bit children $(14/5, x/1), 0 \leq x < 2$, four 2-bit children $(14/5, x/2), 0 \leq x < 4$ through to 2^r r-bit children $(14/5, x/2), 0 \leq x < 2^r$. All such children are related by their common parent and so are called siblings, that is, contexts of the same order but not the same resolution.

To code the input, we form the set of all nodes that match the current context, that is, all nodes that have a context that is common to the full resolution source context. The performance lists, taken together, form a set of "facts". We want to find the member of the set that is the best to code the next input. The initial hypothesis is that none is best (or, equivalently, that they are all best). Then we go through the set of facts one by one and eliminate the hypotheses that don't match the facts. Hopefully we end up with one true hypothesis. For example, in Table 5.3, we are given nodes 0 through 5 with the associated performance information. The "facts" are the value of the REC. So model 4 is the best choice. Of course, there may be the case that two nodes have exactly the same performance. In this case, we use the rule that has served us best so far, namely, we choose the one with the lowest state weight.

Tree growth can grow in two ways: in resolution and in order. Two conditions must be met: 1) the node must be better than all the nodes on its performance list and 2) the count for the current symbol in the node's context must be at least 1. When a node does grow, it grows simultaneously in order and in resolution, in accordance with

by, given the matching set $(p_1, p_2, \cdots, p_k)$, going through the list sequentially starting with p_1 and updating REC_{jk} for all $j > k$, if nodes p_j and p_k need to be compared, that is, if node p_k is on the performance list of p_j.

our conjecture about increasing state weight. Growing in order means that it creates the two 1-bit children that correspond to adding (0/1) and (1/1) to the node's context specification. For example, if the node with context $(14/5, 122/8)$ grows in order, the two children $(14/5, 122/8, 0/1)$ and $(14/5, 122/8, 1/1)$ are created. A node grows in resolution by asking its parent for two siblings which are higher resolution equivalents of itself. Growing the context $(14/5, 122/8, 1/1)$ in resolution means creating the two nodes $(14/5, 122/8, 2/2)$ and $(14/5, 122/8, 3/2)$. Because the parent of a node can also grow in resolution and order, only the last element in the context grows in resolution, unlike the FOVR-II algorithm. From the moment of a node's creation, the members on its performance list are determined and its REC with each on the performance list is subsequently updated. Additionally, on creation of the node, the histogram is updated using an amount of the history (which could be the entire history) specified on initialization of the algorithm. Note that neither Rissanen (1986a) nor Furlan (1990) address this initialization problem.

The tree is initialized with the order 0 model, the null context. Its initial performance list empty so it is better than all members of its performance list and once any symbol has occurred twice, it immediately grows the two 1-bit children (0/1) and (1/1).

Now that we understand how the VOVR model is supposed to work, let us examine its performance on the AR(2) source described in Section 5.2. Figure 5.9 shows the early performance of the VOVR model on the AR(2) source. The context choice is indicated for each new symbol, and as in the previous figures, the number of samples seen increases left to right and top to bottom. When new children are created, it is indicated by a line beginning with "new children", and lists the newly created children. We see that the model begins with the order 0 model, the null context, which is used for four samples. Then the two 1-bit children of the root node are created as shown in Figure 5.10. Both of these children are compared to the root node, but they are not compared to each other because there does not exist an order 1 context that they both could possibly code. During the next 10 samples, all three existing models are used and updated. The $(0, 1)$ model performs well and so it grows in order and resolution, and the $(0/1, 0/1)$, $(0/1, 1/1)$, and $(0/2), (1/2)$ models are created as shown in Figure 5.11. After a few more samples, the $(0/1, 1/1)$

```
() () () ()

new,children:   (0/1) (1/1)

() () () (1/1) (1/1) (0/1) (1/1) (0/1) (1/1) (0/1)

new,children:   (0/1,0/1) (0/1,1/1) (0/2) (1/2)

(1/1) (0/1) (1/1) (1/2)

new,children:   (0/1,1/1,0/1) (0/1,1/1,1/1) (0/1,2/2) (0/1,3/2)

(1/1) (0/1,1/1)

new,children:   (0/1,2/2,0/1) (0/1,2/2,1/1) (0/1,4/3) (0/1,5/3)

(1/1) (0/1,1/1)

new,children:   (0/1,4/3,0/1) (0/1,4/3,1/1) (0/1,8/4) (0/1,9/4)

(1/1) (0/1,1/1) (1/1)

new,children:   (1/1,0/1) (1/1,1/1) (2/2) (3/2)

(0/1,1/1,0/1) (1/1) (0/1,1/1) (1/1,0/1) (0/1,1/1)

new,children:   (0/1,8/4,0/1) (0/1,8/4,1/1) (0/1,16/5) (0/1,17/5)

(1/1,0/1) (0/1,1/1) (1/1,0/1)

new,children:   (1/1,0/1,0/1) (1/1,0/1,1/1) (1/1,0/2) (1/1,1/2)

(0/1,1/1)

new,children:   (0/1,16/5,0/1) (0/1,16/5,1/1) (0/1,32/6) (0/1,33/6)

(1/1,0/1) (0/1,16/5) (1/1,1/2) (0/1,1/1) (1/1,1/2) (0/1,1/1)
(1/1,1/2)

new,children:   (1/1,0/1,1/1,0/1) (1/1,0/1,1/1,1/1) (1/1,0/1,2/2) (1/1,0/1,3/2)

(2/2) () (2/2) (2/2) (2/2) (1/1,1/1) (2/2) (2/2) (2/2)
(2/2) (2/2) (0/1,17/5) (1/1,0/1,1/1) (0/1,17/5) (1/1,0/1)

new,children:   (1/1,0/1,1/1,0/1,0/1) (1/1,0/1,1/1,0/1,1/1)
                (1/1,0/1,1/1,0/2) (1/1,0/1,1/1,1/2)

(0/1,1/1) (1/1,0/1,1/1,0/1) (0/1,1/1) (1/1,0/1,1/1,1/2) (1/1,1/1)
(1/1,1/1) (0/1,1/1,1/1) (1/1,1/2) (1/1) (1/1,1/1) (0/1,1/1,1/1)

new,children:   (0/1,17/5,0/1) (0/1,17/5,1/1) (0/1,34/6) (0/1,35/6)

(1/1,1/2) (0/1,17/5) (1/2) (1/2) (1/1) (2/2) (2/2) (2/2)
(1/2) (2/2) () (1/1,0/1,1/1,1/2)

new,children:   (1/1,0/1,1/1,0/1,2/2) (1/1,0/1,1/1,0/1,3/2)

(2/2) (2/2) (0/1,16/5) (2/2) (2/2) (0/1,16/5,1/1) () (1/2)

new,children:   (0/1,34/6,0/1) (0/1,34/6,1/1) (0/1,68/7) (0/1,69/7)

(1/1,0/1,1/1,0/1,1/1) (0/1,34/6) (0/1,0/1) () () () ()
(0/1,16/5,0/1) () () () () (1/1,0/1,0/1) () () () (2/2) ()
(1/1,0/1,1/1,0/1,0/1) () (0/1,33/6) () () () () () () () () ()
() () () () () (2/2) () (1/1,0/1,1/1,0/1,1/1)
```

```
new,children:   (1/1,0/1,1/1,0/1,4/3) (1/1,0/1,1/1,0/1,5/3)

() () () () (1/2) () () (1/1,0/1,1/1,0/1,1/1) ()
(1/1,0/1,1/1,0/1,1/1) () () () () () () () () () () () ()
() () () () () () () () (1/2) () () () () ()

new,children:   (1/1,1/1,0/1) (1/1,1/1,1/1) (1/1,2/2) (1/1,3/2)

(1/1,1/1) (1/2) () () () () () () () () () () () () () ()
() (1/1,1/1) () () () () () () () () () () (1/1,1/1) () ()

new,children:   (1/1,1/2,0/1) (1/1,1/2,1/1) (1/1,2/3) (1/1,3/3)

() (1/1,1/2) () (1/1,1/2) () (1/1,1/2) () (1/1,1/2) ()
(1/1,1/2)

new,children:   (1/1,3/3,0/1) (1/1,3/3,1/1) (1/1,6/4) (1/1,7/4)

() (1/1,3/3)

new,children:   (1/1,6/4,0/1) (1/1,6/4,1/1) (1/1,12/5) (1/1,13/5)

() (1/1,6/4) () (1/1,1/2) () (1/1,0/1,1/1,1/2) ()
(1/1,0/1,1/1,1/2)

new,children:   (1/1,7/4,0/1) (1/1,7/4,1/1) (1/1,14/5) (1/1,15/5)

() () (1/1,7/4) () (1/1,7/4) () (1/1,3/3) () (1/1,14/5)

new,children:   (1/1,14/5,0/1) (1/1,14/5,1/1) (1/1,28/6) (1/1,29/6)

() (1/1,1/2) () (1/1,14/5) () (1/1,3/3) (0/1) () () () () ()
() (1/1,1/2) () (1/1,2/3) (0/1,1/1) (1/1,3/3) (0/1,1/1)
(1/1,0/1,1/1,1/2) (0/1,1/1) (1/1,1/2) (0/1,1/1) (1/1,0/1,1/1,1/2)

new,children:   (1/1,2/3,0/1) (1/1,2/3,1/1) (1/1,4/4) (1/1,5/4)

(0/1,1/1) (1/1,12/5) (0/1,1/1) (1/1,2/3) (0/1,1/1) (1/1,2/3,1/1)

new,children:   (1/1,5/4,0/1) (1/1,5/4,1/1) (1/1,10/5) (1/1,11/5)

(0/1,1/1) (1/1,2/3) (0/1,1/1) (1/1,7/4) (0/1,1/1) (1/1,7/4)
(0/1,1/1) () () () () () (1/1,7/4) (0/1,1/1) (1/1,7/4)
(0/1,1/1) (1/1,7/4) (0/1,1/1) (1/2) (1/2) (1/1,7/4) (0/1,1/1)
(1/1,7/4) (1/1) (1/1,1/1,1/1) (1/1,1/1,1/1) (1/1,1/1,1/1)
(0/1,1/1) (1/1) (1/1) (1/1) (1/1,1/1) (1/1,1/1) (1/1,1/1)
(1/1,1/1) (1/1,1/1) (1/1,1/1) (1/1,1/1) (1/1,1/1) (0/1,1/1)
(1/1) (1/1,1/1) (1/1,1/1) (1/1) (1/1,1/1) (1/1) (1/1) (1/1)
(1/1) (1/1) (2/2) (0/1,1/1) (1/1) (0/1,1/1) (1/1) (0/1,1/1)
(1/1) (0/1,1/1) (1/1,0/1) (0/1,1/1) (2/2) (1/1) (1/1) (1/1)
(0/1,8/4) (1/1) (1/1) (0/1,1/1) (1/1) (0/1,1/1) (1/1)
(0/1,1/1) (1/1,1/2) (0/1,1/1) () () () () () () () ()
(1/1,12/5) () () (1/1,1/2) () (1/1,1/2) () (1/1,0/1,1/1,0/1) ()
(1/1,1/2) () (1/1,1/2) (1/1) (1/1,1/1) (0/1,1/1) (1/1,1/2)
(1/1) (1/1) (1/1) (1/1) (1/1) (1/1) (1/1) (1/1) (1/1)
(1/1) (1/1) (1/1) () (1/1,1/2) () (1/1,1/2) () (1/1,1/2) ()
(1/1,1/2) () (1/1,1/2) () (1/1,1/2) () (1/1,1/2) () () ()
(1/1,12/5) () () () () () () () () () () () () () () () ()
(1/1,2/3) () (1/1,2/3) () (1/1,12/5) () (1/1,1/2) () () ()
(1/1,1/2) () (1/1,1/2) (1/1) (1/1) (1/1) (1/1) (1/1) (1/1)
() () (1/1,1/2) (1/1) (1/1) () () () () () () () ()
```

Figure 5.9: Early VOVR model choice and growth for the AR(2) source

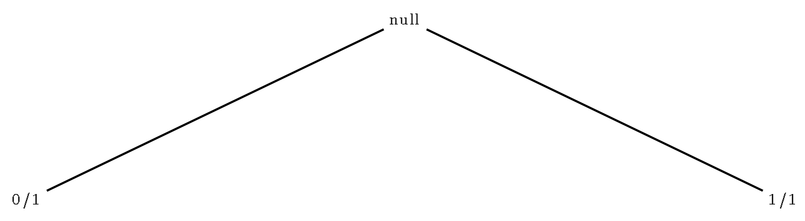

Figure 5.10: VOVR tree after 4 samples

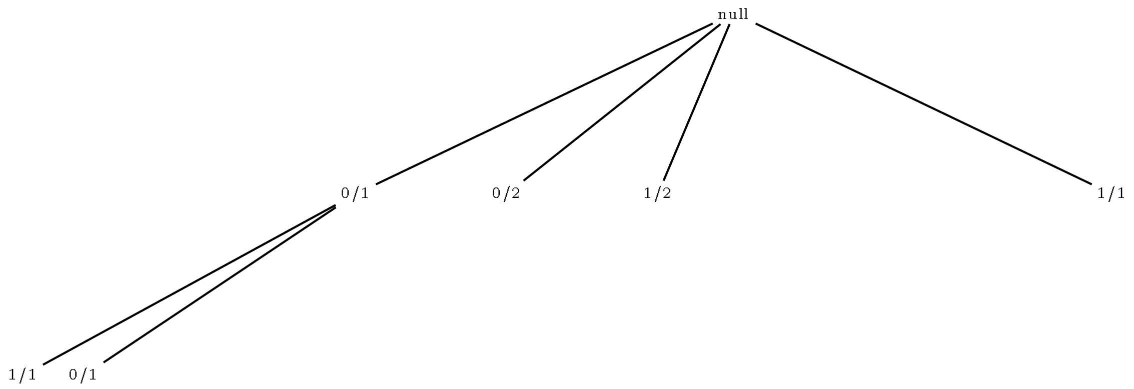

Figure 5.11: VOVR tree after 14 samples

model outperforms all the other models on its performance list and grows the children $(0/1, 1/1, 0/1)$, $(0/1, 1/1, 1/1)$, and $(0/1, 2/2)(0/1, 3/2)$ as shown in Figure 5.12. Within a few samples the $(0/1, 2/2)$ model grows, creating children $(0/1, 2/2, 0/1)$, $(0/1, 2/2, 1/1)$, $(0/1, 4/3)$, and $(0/1, 5/3)$. So we see that, like the FOVR-II algorithm, the VOVR algorithm rapidly converges on the appropriate order and resolution. After a few more samples, the $(1/1)$ side of the tree begins to grow also, as samples in that region of the state space appear. The tree after 33 samples is shown in Figure 5.13.

The final codelength for the AR(2) source of 5.28 bits/pixel, which compares to 5.19 bits/pixel for the optimal static choice and 5.18 bits/pixel for the FOVR-II model. The difference is attributable to the fact that the VOVR model cannot completely skip conditioning information as can the FOVR-II model, but is instead forced to allocate at least one bit to it.

5.5.1 Experiments

Figure 5.14 shows the performance of the VOVR model on the test set. As with the FOVR-II algorithm, the performance of the VOVR algorithm was as good or better than the optimal static resolution choice as determined by exhaustive search, except in the case of the `lena` image. Although the VOVR algorithm performed better on `lena` than the FOVR-II algorithm. The performance on the `photog` image is an anomaly

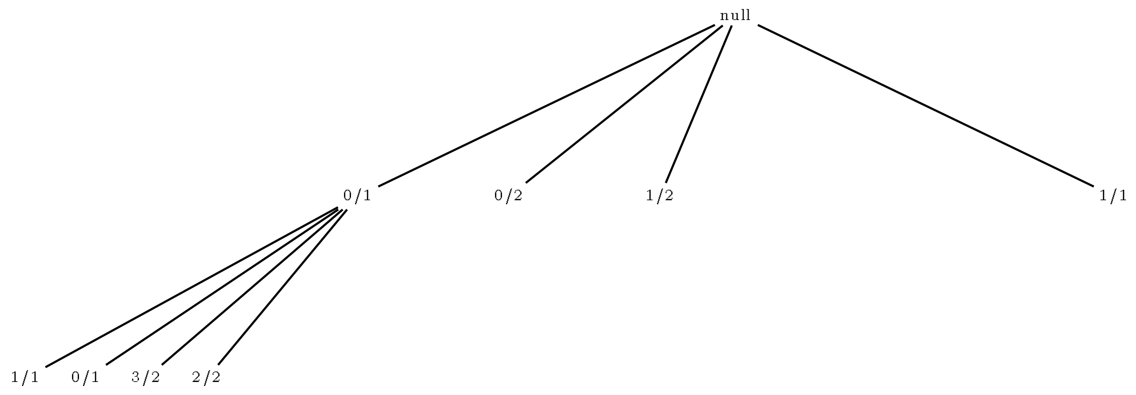

Figure 5.12: VOVR tree after 16 samples

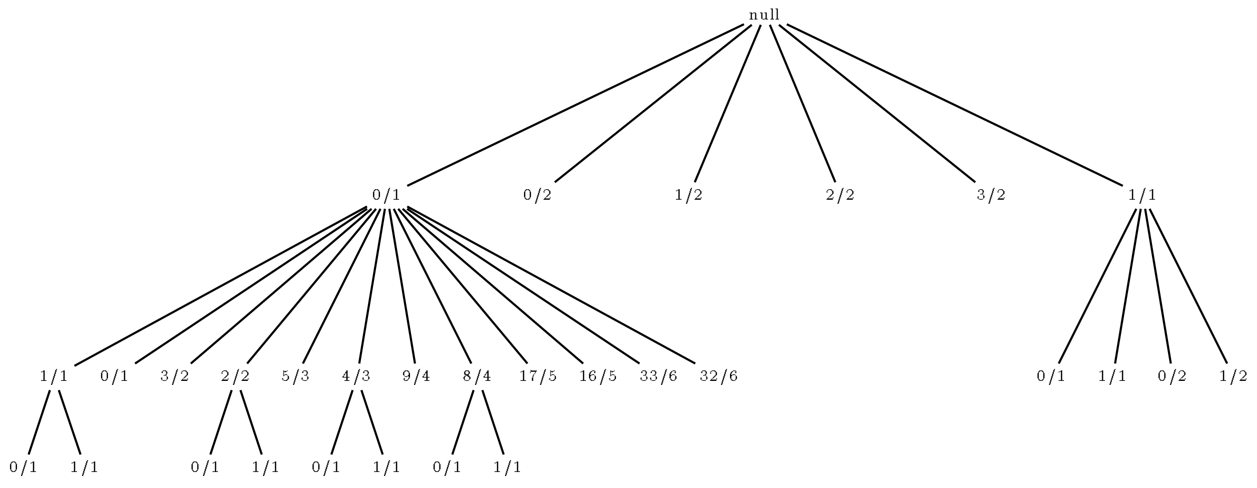

Figure 5.13: VOVR tree after 33 samples

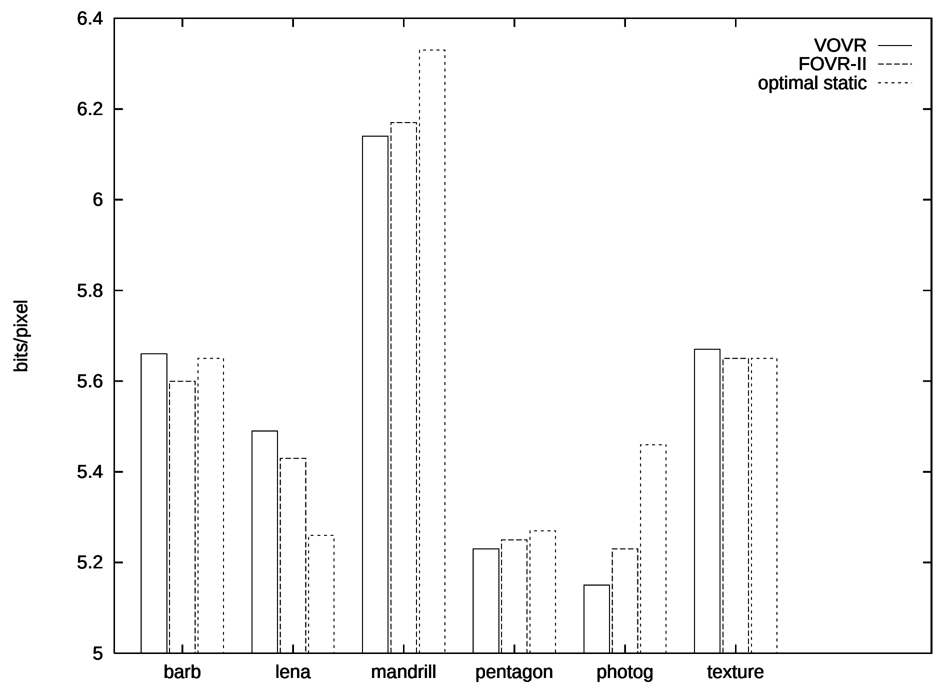

Figure 5.14: VOVR performance on the test set

also, although in this case, the VOVR algorithm produced much better results than either the optimal static or the FOVR-II algorithm. Both these performance increases may be attributable to the fact that the VOVR algorithm retains all lower state weight models instead of destroying them after they are not used for some time. Because they are retained, when the algorithm encounters a non-stationarity, the lower state weight models, which may have been already discarded in FOVR-II, still exist. As mentioned previously, these lower state weight models learn and un-learn more quickly and are more likely to be used when the algorithm encounters a non-stationarity.

5.6 Comparison to PPMI

These algorithms differ in a significant way from PPMI. Instead of using fixed order context, this algorithm explores both variable order and variable resolution contexts using a minimum description length criteria. It grows only the contexts that are performing well and only to the extent in resolution and order that is necessary. Finally, instead of using an arbitrary criterion for how many samples is enough to constitute a reliable set for a given order/resolution the algorithm uses just the recent codelength.

Unfortunately, none of the results presented by Howard and Vitter (1991) were on the same images as those in the test set for this thesis, and since we discovered their work as the work in this thesis was being completed, no method for numerical comparison was developed.

5.7 Relationship to Permutation Selection

Having control over both the context resolution and order is related to permutation selection in the following way. A particular permutation chosen based on *a priori* knowledge of the source (say that it is an image with rows of N pixels each yielding the set of pixels ordered by Euclidean distance from the pixel to be coded) may not be perfectly suited to the statistics of the source being coded. In that case, we may actually waste time and memory collecting statistics for contexts $x_c, \forall c \in A$ such that $P(x|x_c)$ is virtually the same as $P(x)$; that is, where the autocorrelation function $R(c)$ is near zero. For these situations we would ideally like to eliminate this poorly chosen element of our permutation vector. Varying the resolution approximates this by assigning as few bits as possible to that permutation element.

The model growth rules in both FOVR-II and VOVR are in fact implementing a form of permutation function definition, as well as choosing the optimal resolution for the conditioning information. In the case where we have incorrectly selected the permutation function, those members of the function that do not help as much as others will have fewer bits of resolution allocated to them by the algorithm. To be more concrete, say we chose a permutation function appropriate for image data and applied it instead to scalar data. In this case, the permutation function components close in terms of Euclidean distance in an image would actually be quite far apart in Euclidean distance for the scalar, and the correlation would most likely be smaller. As shown by their performance on the AR(2) source with parameter vector $\phi = \{0.01, 0.89\}$ and the corresponding correlation sequence in Figure 5.1 these algorithms skip (by allocating zero bits to the context in FOVR-II or very few bits to the context in VOVR) the conditioning information not strongly correlated to the value being coded. The algorithms would do the same had we poorly chosen the permutation, skipping the entries corresponding to the values that would be on the line above the current pixel were the scalar source actually an image source.

Interestingly, even if the permutation choice is nearly correct, there still exists a fundamental limitation of tree structured algorithms like UMC. There may be correlation that could be exploited at a different offset from the current value being coded than was specified by the permutation function, but the algorithm insists on increasing the order only if the current order under trial begins to do better. Hence, the algorithm tends to get "stuck" on low correlation conditioning information. The FOVR-II and VOVR algorithms solve this problem by using low resolution conditioning information that will tend to show a trend faster than full resolution conditioning information.

5.8 Computational Complexity

As noted at the end of Section 5.3, the expense of maintaining nearly 100 models in the FOVR-I algorithm was great enough to merit investigation into a more efficient algorithm. It was seen that the FOVR-II algorithm, running at most only 16 models simultaneously, performed equivalently, and consumed correspondingly fewer resources. Judging from the description alone, the VOVR algorithm appears more complicated due to the list handling overhead. However it is difficult to compare the algorithms objectively because they are structurally adaptive and the amount of resources they consume depends on the nature of the data being coded. However, as a purely subjective comment, it appeared that the FOVR-II algorithm processed a 512×512 image comfortably (within a few minutes) on a normally loaded Sun SparcStation 10, whereas the FOVR-I and VOVR took significantly longer (tens of minutes). As a final disclaimer, the code (especially the VOVR code) was written with modularity and extensibility in mind, rather than execution speed.

5.9 Summary

In this chapter, we have investigated several methods for solving the Markov model learning problem. Instead of adjusting the number of states in the model using only the model order, as in VOFR techniques, to code signal data we have the ability to vary the resolution of the conditioning information as well. In fact, to obtain the performance obtainable from methods like DPCM, we *must* vary the resolution this way. Due to their flexibility, we see that on signal sources that cannot be well represented by a linear model, variable order/variable resolution models can outperform DPCM. However, it appears that many natural image sources can be well represented by linear models. The two classes of techniques, FOVR and VOVR, differ in computational complexity, adaptation to the stationary non-stationary characteristics of the source, but their performance overall is approximately the same. Both implement a useful form of permutation selection, and effectively ignore data that does not help code the source.

Chapter 6

Bit Group Modelling of Signal Data

6.1 Introduction

We noted in Chapter 3 that the binary variable order algorithms like UMC and JBIG could be made to operate on non-binary data by splitting the non-binary data into planes, each of 1 bit resolution, and passing each plane to a separate instance of the algorithm. The UMC algorithm performed quite well in this regard as shown in Section 3.2.3.

In this chapter we attempt to develop an understanding of why binary modelling of non-binary signal data is so effective. We investigate the common technique of Gray coding the data first before splitting it into single-bit planes and passing to the model and coder, and compare it to a simple weighted binary coding. We pose the question, is Gray coding the data and splitting into bit planes the best approach to modelling signal data or can we do better? We propose a non-binary pseudo-Gray code as a method of generating planes of resolution greater than or equal to 1 bit, and compare it with the other conventional methods. Finally, we relate the work in this chapter to the ideas from Chapter 5 on variable order/variable resolution modelling.

6.2 Bit Group Modelling

Consistent with our discussions on the learning problem with Markov models, splitting an r-input into r separate planes results in r separate streams that are each easier to code in the sense that the histogram counts reach their actual values quicker; however the sum of the codelength may not be less. Counters of ones and zero reach stable counts faster than higher resolution counters, but because we may be disregarding some relationships between the planes, the resulting probability distribution (the convolution of the lower resolution distributions) may not be the same as the higher resolution distribution, which, when properly trained, approximates the real distribution more closely.

Two methods of splitting an input into single-bit planes are now described.

Weighted Binary An r-bit input can be split into r separate inputs according to bit placement. We call this method *weighted binary coding.*

Binary Reflected Gray Code An r-bit source can first be coded with an r-bit binary reflected Gray code and then the resulting coded input can be weighted binary coded. We will see that the effect of the Gray code is to preserve more of the correlation that exists between two adjacent pixels than the simple weighted binary code would.

To construct an r-bit binary reflected Gray code

1. start with the all zero code word corresponding to the source zero
2. form the next codeword by changing the least significant bit that results in an unused code word

A four-bit binary Gray code is shown in Table 6.1. Included in the table are the corresponding weighted binary code, and, in the first column, one possible domain in R^1 that could be mapped to the binary index. Also included in the table are the number of bit changes in each plane of the weighted binary code and the Gray code. We notice that a Gray code tends to minimize the number of changes per plane and for the code overall. Also, it tends to try to concentrate the changes in the lower

Table 6.1: Four-bit binary Gray code

real number	decimal index	binary index	gray code
-7.5	0	0 0 0 0	0 0 0 0
-6.5	1	0 0 0 1	0 0 0 1
-5.5	2	0 0 1 0	0 0 1 1
-4.5	3	0 0 1 1	0 0 1 0
-3.5	4	0 1 0 0	0 1 1 0
-2.5	5	0 1 0 1	0 1 1 1
-1.5	6	0 1 1 0	0 1 0 1
-0.5	7	0 1 1 1	0 1 0 0
0.5	8	1 0 0 0	1 1 0 0
1.5	9	1 0 0 1	1 1 0 1
2.5	10	1 0 1 0	1 1 1 1
3.5	11	1 0 1 1	1 1 1 0
4.5	12	1 1 0 0	1 0 1 0
5.5	13	1 1 0 1	1 0 1 1
6.5	14	1 1 1 0	1 0 0 1
7.5	15	1 1 1 1	1 0 0 0
changes		1 3 7 8	1 2 4 8

bits. This has the effect that within any continuous range of values $[x_{\text{min}}, x_{\text{max}}]$, more often than not, the values will all get mapped into either a to 0 or a to 1, preserving boundaries between regions containing similar values as well as possible. The likelihood decreases, of course, with the size of the range.

However, some symbols of similar, but not exactly the same value will get mapped in the opposite sense at the region boundaries, and there are more boundaries in the lower significance bits. For example, in bit plane 3 (the most significant bit plane) of Table 6.1, there is only one case where two values separated by the minimum Euclidean distance get mapped into different symbols (going from index 7 to 8) whereas in bit plane 1 there are four instances; contrast this with the weighted binary case for which, in bit plane 1 there are seven instances. Similarly, two symbols widely separated in Euclidean distance, may be get mapped to the same code in one or more of the bit planes. For example, in bit plane 2 of the binary Gray code, indices 0 and 11 get mapped to the same value 1.

Overall then, the Gray code attempts to arrange the values so that when processed

Table 6.2: Entropy and conditional entropy for bit planes of Gray coded and original `lena` image

	weighted binary		binary Gray	
bit plane	$H(x_t)$	$H(x_t\|x_{t-1})$	$H(x_t)$	$H(x_t\|x_{t-1})$
0 (LSB)	1.000	1.000	1.000	1.000
1	1.000	1.000	1.000	0.998
2	1.000	1.000	0.999	0.972
3	1.000	0.973	1.000	0.880
4	1.000	0.906	1.000	0.787
5	1.000	0.745	1.000	0.521
6	0.984	0.599	0.815	0.401
7 (MSB)	1.000	0.380	1.000	0.380
total	7.984	6.603	7.814	5.939

by an algorithm or by visual inspection, the features that were distinct to order 1 in the original data are reproduced as distinct in binary form. The binary Gray code is useful for Markov modelling because it maintains correlation between values fairly well, but reduces the resolution so that the learning problem is reduced.

Comparison Let us continue the discussion of weighted binary versus Gray coding by looking at the entropy of the bit planes generated using the two methods. For the `lena` image, Table 6.2 shows the entropy of weighted binary coded and Gray coded bit planes. Per the previous discussion, the increased in compression performance is attributable to the fact that the binary reflected gray code preserves more of the sample to sample correlation than does the weighted binary code. Contrast, however, these numbers with the entropy and conditional entropy of the original `lena`, 7.446 bits/pixel and 5.5052 bits/pixel, respectively. The difference is due to the fact that splitting the r-bit input into r separate planes ignores any inter-plane relationships. Hence, this result leads us to speculate that using the original greyscale pixels is superior to using the bit planes and is the motivation for the work in Chapter 5. But from this work, we know that the two frequency counters for a one bit alphabet will approximate their distribution faster than say, the four counters for a two bit alphabet, i.e., the learning problem. So, is there a middle ground? Can we the input into groups of bits of different sizes than just one bit per group in attempt to keep the inter-plane relationships that may be useful, but also keep the resolution low enough so that the learning problem is not so great?

6.3 Non-Binary Pseudo-Gray Coding

This section defines a methodology for constructing a non-binary pseudo-Gray code. The standard reflected binary Gray code is a special case.

The logic of the algorithm is essentially the same as that for the construction of a binary gray code, except everywhere we thought Hamming distance in the binary case, we think Euclidean distance in the non-binary case.

We are given as a specification a set of boundaries dividing the r-bit word in to $n = 2$ to r groups of 1 to r pixels so that the sum of the sizes of the groups is r. If $n = r$ then the group size for all n groups is 1, and the algorithm reduces to the binary Gray code. If $n = 1$ then the size of the group is r, and the code performs the identity operation. To generate the non-binary pseudo-Gray code

1. start with the all zero code word corresponding to the source zero
2. form the next codeword by changing the least significant *group* by the least amount that results in an unused code word; the change can be either an increment (add 1) or decrement (subtract 1) of the value in the group; if an increment or decrement does not generate a new code word, move to the next higher significance group

A $(3, 2)$ non-binary pseudo-Gray code is shown in Table 6.3 along with a 5 bit binary Gray code and the 5 bit weighted binary representation for comparison. Immediately noticeable is the fact that the most significant group of the non-binary pseudo-Gray code is always the same as the corresponding bits in the weighted binary representation.

Interpreting the non-binary pseudo-Gray code in the same way as in the discussion of the binary Gray code, we see that the code tends to assign values which are close in Euclidean distance codewords that are close in Euclidean distance. At the same time,

Table 6.3: The non-binary pseudo-Gray code (3,2)

decimal index	weighted binary	non-binary pseudo-Gray	binary Gray
0	000 00	000 00	000 00
1	000 01	000 01	000 01
2	000 10	000 10	000 11
3	000 11	000 11	000 10
4	001 00	001 11	001 10
5	001 01	001 10	001 11
6	001 10	001 01	001 01
7	001 11	001 00	001 00
8	010 00	010 00	011 00
9	010 01	010 01	011 01
10	010 10	010 10	011 11
11	010 11	010 11	011 10
12	011 00	011 11	010 10
13	011 01	011 10	010 11
14	011 10	011 01	010 01
15	011 11	011 00	010 00
16	100 00	100 00	110 00
17	100 01	100 01	110 01
18	100 10	100 10	110 11
19	100 11	100 11	110 10
20	101 00	101 11	111 10
21	101 01	101 10	111 11
22	101 10	101 01	111 01
23	101 11	101 00	111 00
24	110 00	110 00	101 00
25	110 01	110 01	101 01
26	110 10	110 10	101 11
27	110 11	110 11	101 10
28	111 00	111 11	100 10
29	111 01	111 10	100 11
30	111 10	111 01	100 01
31	111 11	111 00	100 00

it allows groups of sizes larger than one bit in an attempt to preserve some inter-plane information.

In the following sections we will see whether using the non-binary pseudo-Gray code gives us an advantage over using single-bit planes in adaptive Markov modelling.

6.4 Experiments

In this set of experiments, we separate an 8-bit source into all possible groups of adjacent bits. Table 6.4 summarizes. For the discussion that follows, the identification in column 5 of the table will be used. The notation $l - b$ means a bit group of length l starting with bit b. Hence, the group identification "5-6" means the five-bit group starting with bit 6, i.e., bits 6,5,4,3, and 2.

The conventional methodology is to split an r-bit input into r separate planes and pass each plane to a Markov model. We would like to explore the continuum between this method and using the full r bits of information all at once. To be fair, we will choose the maximum allowable order of the Markov model such that the number of states is held (nearly) constant. The model maximum allowable order as a function of the group size is also shown in Table 6.4.

The coder we will use is a simple variable order, fixed resolution algorithm. This coder uses the highest-order matching context that has at least one symbol occurrence, up to the maximum allowable order.

In Table 6.5, the results of using several combinations the groups identified in Table 6.4. In column 1, $r = 8$ separate planes of 1 bit each were used, corresponding to the standard method. In columns 2 and 3, the results of coding one q-bit plane and $8 - q$ 1-bit planes are given. In columns 4 and 5, the results of coding one q-bit plane and one $8 - q$-bit plane are shown. The result of coding four 2-bit planes is shown in column 6. There were $r = 8$ choices for q for each image, but the one that resulted in the lowest codelength is shown. A subset of these results are shown in Figure 6.1 along with the rate of the optimal static order 1 and order 2 resolution-reduced coders (as might be determined by the FOVR-II algorithm, for example) for comparison. There are several things to notice. First, except for the `lena` and `barb` images, the optimal static order 1 codelength was close to the performance attained by Gray coding the bit planes first. Second, there was always a slight performance increase over simple

Table 6.4: All possible groups of adjacent bits, r=8

group size	order	weight	members	group id
1	8	8	7	1-7
			6	1-6
			5	1-5
			4	1-4
			3	1-3
			2	1-2
			1	1-1
			0	1-0
2	4	8	76	2-7
			65	2-6
			54	2-5
			43	2-4
			32	2-3
			21	2-2
			10	2-1
3	3	9	765	3-7
			654	3-6
			543	3-5
			432	3-4
			321	3-3
			210	3-2
4	2	8	7654	4-7
			6543	4-6
			5432	4-5
			4321	4-4
			3210	4-3
5	2	10	76543	5-7
			65432	5-6
			54321	5-5
			43210	5-4
6	1	6	765432	6-7
			654321	6-6
			543210	6-5
7	1	7	7654321	7-7
			6543210	7-6
8	1	8	76543210	8-7

Table 6.5: Performance of several bit groupings

image	Gray code	q	rate	q	rate	4 groups of 2
barb	5.49	2	5.40	4	5.58	5.40
lena	5.31	2	5.18	4	5.44	5.14
mandrill	6.52	4	6.26	4	6.32	6.36
pentagon	5.43	5	5.24	5	5.32	5.28
photog	5.49	4	5.30	4	5.51	5.27
texture	6.12	5	5.71	5	5.80	5.81

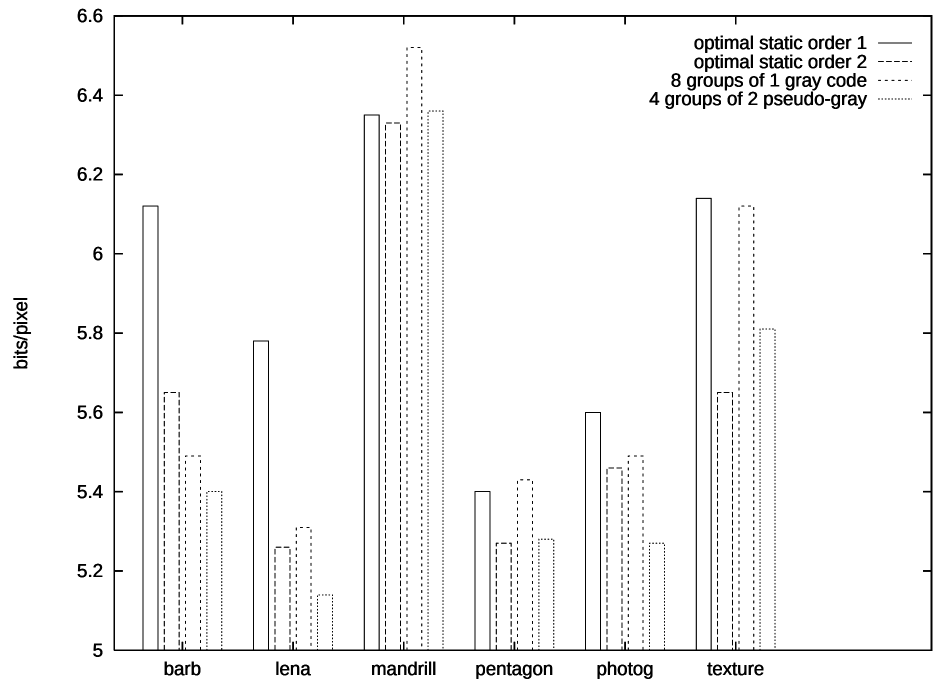

Figure 6.1: Comparison of bit group techniques

Gray coding, but generally less than 5%, in coding some larger group of bits than coding just the bit planes, and the optimal bit grouping varied from image to image. Those images whose performance was not increased significantly by coding larger bit groups were the same images for which the performance of the FOVR-II algorithm was poorer than Gray coding the bit planes, namely `barb` and `lena`. This suggests that in those images there is a more complex mechanism operating than simply the learning problem. Indeed, examining the codelength profile of those images, both appear to be non-stationary with respect to an order 1 model. The most striking improvement was obtained on the `texture` image, significantly the most homogeneous in appearance of all the images. Coding four planes of two bits consistently outperformed simple Gray coding also, and was nearly the same as coding q bits together and the remaining $8-q$ bits separately. Coding two groups, one of q bits and the other of $8-q$ bits were again anomalous for `barb` and `lena`, performing worse than the Gray code, but for other images, it performed slightly better, though consistently worse than the other groupings.

Unfortunately, not all combinations of bit groupings were explored, only the ones that were intuitively attractive. This is an area of future research.

6.5 Summary

We have investigated the method of splitting an r-bit input into planes and coding the planes separately as a technique for overcoming the learning problem in Markov models. The traditional method of doing this, namely Gray coding, was analyzed in order to get a better understanding of what this technique does. Motivated by its structure, a non-binary pseudo-Gray code was proposed, which has the desirable characteristic of allowing us to generate planes of resolution greater than 1 bit but less than the full source resolution. Thus, the non-binary pseudo-Gray code gives us the ability to preserve some inter-plane relationships. The non-binary pseudo-Gray code includes the standard binary reflected Gray code as a special case. The non-binary pseudo-Gray code was used to generate planes of varying resolutions for the members of the test set and the planes were coded using a variable order, fixed resolution Markov model. It was seen that some combinations of planes with resolution higher than 1 bit performed better than the binary Gray code method, though this performance increase was generally less than 5%. However, If it is desirable to try to get the extra performance, one could of course use the methodology upon which most of Chapter 5 was based, namely, run the models in competition and select between them based on their recent codelength.

Chapter 7

Applications

This chapter explores the application of Markov modelling to other areas besides directly compressing signal data. Based on Section 3.1.7 we know that signal data prediction is useful in the context of DPCM systems because the prediction is subtracted from the signal and the lower power residual signal is optionally quantized and transmitted or stored. We will see that a convenient side effect of Markov modelling is that we can also use the models as predictors, and that adaptive Markov predictors can function as well as or better than their linear model counterparts, but that most natural image data can be better represented by linear models. We continue the discussion of using Markov models coupled with DPCM, showing that the properties of the FOVR and VOVR models presented in Chapter 5 can yield very good compression. Finally, we look more carefully at the structure of these joint DPCM/Markov coders and present a new method of DPCM, distortion-constrained DPCM.

7.0.1 Using Non-Binary Markov Models as Predictors

A probability density estimate is available for each sample x, namely, the one that the coder would use to transform the distribution and symbol into a sequence of bits via an arithmetic coder. We know that the optimal predictor, linear or non-linear, is the (conditional) expected value, $E(X)$. Instead of using the distribution to code the input, we can easily calculate the expected value instead. The resulting estimate is unconditionally stable, although not necessarily good.

Table 7.1: Prediction of test set

	PSNR		
image	order 3 DPCM	order 3 ADPCM	FOVR-II
`barb`	24.54	24.59	24.30
`lena`	27.22	27.08	25.83
`mandrill`	22.45	22.36	22.04
`pentagon`	27.62	27.48	26.76
`photog`	22.58	22.53	20.06
`texture`	26.03	25.52	25.95

For a model that operates on r-bit streams, the expected value is calculated simply,

$$E(X) = \sum_{a \in A} aP(X = a) \ . \tag{7.1}$$

7.0.2 Using Binary Markov Models as Predictors

A binary Markov model gives us predictions for a single bit plane of an image with r bits/pixel. If we want to form an r-bit prediction for a particular pixel in the image, we need to form the new probability estimate

$$P(X = a) = \prod_{i=0}^{r-1} P(X_i = a_i) \ , \forall a \in A \ . \tag{7.2}$$

Given that probability distribution, we can apply 7.1.

7.0.3 Experiments

The inputs to the model were the members of the image test set. The images were predicted using 1) order 3 DPCM (after an initial pass to determine the linear predictor coefficients, 2) order 3 ADPCM, and 3) the FOVR-II algorithm of Chapter 5. For the FOVR-II algorithm, we set the maximum number of models, `MaxModels` to 128, the `MemoryUsage` to 16 megabytes, and the maximum order of the models, `MaxOrder`, to 2.[1]

The order 3 DPCM system worked the best overall, followed closely by the adaptive DPCM system. The FOVR-II algorithm yielded generally poorer results, ranging from

[1]Due to the `MemoryUsage` constraint, setting `MaxOrder` to 3 did not result in better performance overall, and in some cases the performance was worse. As postulated in Section 5.4.1, this may be due to the fact that these sources are non-stationary.

1.7% better on the `barb` image to 11% worse on `photog` image. Averaged over the test set, FOVR-II performed about 3.2% worse than ADPCM.

The equivalence of properly trained Markov models and DPCM on stationary linear sources is easy to explain. The differential entropy for a Gaussian source with mean μ and variance σ is $\frac{1}{2}(\ln 2 + \ln \pi + 1 + 2 \ln \sigma)$. The conditional probability distribution of the AR(N) source has mean $\sum_{i=1}^{N} \phi_i x(n-i)$ and, more importantly, variance σ_N. This has a differential entropy depending on the variance, as above. Every conditional probability distribution will have the same variance, σ_N. The overall entropy then is the expected value of the conditional differential entropy, which is a constant, so the overall entropy is the same value. For the DPCM case, if we have estimated the parameters correctly, we are just coding the error sequence, which by definition has power σ_N. So the two approaches are the same. The only difference arises during adaptive coding, but if the Markov model resolution and order are continually adjusted, it can learn almost as fast as DPCM.

However, we expect that the adaptive Markov model could do better than DPCM on some sources. If we think of DPCM as if it were a Markov model, all the conditional histograms will have the same shape; only their means will be different. On the other hand, a Markov model has no such restriction; the distribution can be completely different from one context to the next, resulting in greater modelling flexibility. Consider a source whose symbol probability distributions are identical in some set of order 1 contexts, but for the rest of the order 1 contexts the symbols are distributed exactly as if they had been generated by an order 1 linear model, that is, for each of those order 1 contexts the mean of the distribution is the value of the context. The increased flexibility of the Markov model should result in better performance than DPCM on this source.

We generated just such a source. The first 2^{17} samples were generated by an order 1 linear model driven by white Gaussian noise, to which was added an offset that moved the mean far enough away from zero so that the probability of a symbol less than zero occurring is negligible. The second 2^{17} samples were white Gaussian noise with the same variance as above, but from which was subtracted the same offset added to the order 1 source. In this way, over the whole sample, a symbol less than zero is almost surely to be followed by another symbol less than zero but the two symbols will be independent. At the same time, a symbol greater than zero is almost surely to be followed by another symbol greater than zero but the two symbols will be correlated. Hence, the conditional histograms that correspond to the order 1 model are effectively (though not ideally) localized in one half of the two-dimensional support of the sample. In the other half of the support, the conditional histograms are identical in mean and variance. This source should present a problem to the DPCM model because it assumes that the autocorrelation is constant over the sample. It would, in fact, appear bimodal to an order 1 DPCM system, but it would appear stationary to an order 1 Markov model, since the two characteristics of the source are in distinct regions of the support.

We ran the FOVR-II model, and adaptive and non-adaptive order 3 DPCM models on the source. As expected, the Markov model performed better. The PSNR for the three methods were 29.62, 28.30, and 28.26 dB, respectively.

7.0.4 Summary

A non-linear adaptive Markov predictor was seen to perform slightly worse than linear predictors on the test set. However, certain non-stationarities that would cause a linear predictor to perform poorly, even with a pre-scan, do not present a problem for a Markov predictor but, based on the test set, such non-stationarities do not appear to be common in natural images.

7.1 DPCM/Markov Coding

As mentioned in the introduction and in Section 3.1.7, we can use DPCM in conjunction with Markov coding because, while running a Markov model as a predictor is theoretically better than using a linear model, for a large class of inputs that can be effectively modelled as a linear stochastic system, a Markov model (with appropriately chosen resolution and order) appears to do no better, but is more computationally expensive. This leads us to speculate that Markov models may be better suited to coding the residual of a DPCM system. Several authors have already described systems, for example Tischer, Worley, Maeder, and Goodwin (1993) and Todd, Langdon, and Rissanen (1985). However, let us explore in more detail the characteristics of the residual that would be presented to a Markov model.

7.1.1 Characteristics of the DPCM Residual

If the DPCM parameters are exactly matched to the linear model that generates the source appearing at its input, the residual sequence is white noise with the same power as the noise driving the input generator. We can still use an order 0 model on this sequence and compress it down to its order 0 entropy quite easily. If, however, the generator has higher order than the model, the model parameters are wrong, or the generator is simply not linear, the resulting sequence will not be white. This implies that we can gain from coding the residual sequence with a Markov model. Of course, in the first two cases, we could also simply increase the order or accuracy of our DPCM system.

For an AR(N) source modelled by a order M linear model, with $M < N$, it is easy to determine the characteristics of the residual. However, for other sources it is more complicated and perhaps overall the exercise is not instructive. We do expect, however, that whatever the characteristics of the residual, a Markov algorithm like FOVR-II would determine the proper resolution and order to best code it.

It is significant to note that in practice, we can represent the residual from an r-bit input using only r bits instead of $r + 1$ bits as one might expect. This is done relying on the fact that the predictor generates integer predictions and using modulo-r addition and subtraction. However, the r-bit representation of the residual is quite different in character from the $r + 1$-bit representation. The entropies are the same, but it appears from preliminary investigations that the "adaptive entropies" are different, i.e., when presented to equivalent adaptive models, the resulting codelengths are different. Exploring the differences between the two representations and how Markov models react to it is a topic for further research.

While lossless coding has its place, there are many application areas in which some amount of distortion can be introduced into the image in order to increase the compression of the data. In the next section, we will describe in detail yet another variant of DPCM.

7.2 Distortion-Constrained DPCM

Because signal data is often noisy, we are lead to speculate that perhaps changing a few values in the input will not cause a perceptible difference to the viewer. The problem is determining which pixels those are and what is the effect of changing the pixels on the overall rate. One traditional method of reducing the rate is by quantizing the residual. However, in this section we propose another.

Suppose that we are allowed to alter the input to a lossless (quantizer disabled) DPCM coder by adding a noise sequence n_t, and we choose n_t in such a way as to minimize the entropy of the residual sequence while at the same time keeping the power of the noise, $E(n_t^2)$, low. In other words, we wish to minimize the entropy of the residual $H(r_t)$ subject to a constraint on the power of the noise sequence $E(n_t^2)$, where $r_t = x_t - \hat{x}_t$, and $\hat{x}_t$ is the prediction for x_t.

It is known that the constrained minimization problem $\min x$ given $y \leq z$ can be replaced by the unconstrained minimization $\min(x + \lambda y)$. The parameter λ can be adjusted for the particular required value of z. Hence, to solve the problem at hand, we perform an unconstrained minimization of $\min(H(r_t) + \lambda E(n_t^2))$ for some choice of λ. If λ is zero, we end up with an unconstrained minimization of the entropy. No consideration will be given to the resulting error. If, on the other hand, the value is non-zero, we include in a penalty for the squared error incurred by replacing the original with another value. If λ is very large, we can overwhelm the effect of the entropy and just minimize the error, which means that we would essentially never change any value, $e_t = 0$

Unfortunately, to solve this problem correctly, we would need to determine the residual resulting from every possible noise sequence of length N and choose the minimum. Obviously this is impractical. A more practical solution is to assume that at each step, the distribution of the residual doesn't change by much, and so doing a local minimization at each step should result in a fairly good global minimization.

The algorithm can be summarized as follows.

1. calculate the prediction $\hat{x}_t$
2. for each integer n_t in $[-n_{\max}, n_{\max}]$, calculate the cost function
$$-\log(P(\hat{x}_t - x_t + n_t)) + \lambda n_t^2$$
3. choose the value of n_t that results in the minimum cost function

7.2.1 Experiments

The DC-DPCM system described above was applied to the images in the test set. The predictor was an order 3 linear predictor whose coefficients were determined by a pre-scan of the data and whose output was constrained to integer values. Based on the results in Section 7.0.3, we could have just as effectively used one of the adaptive Markov models described earlier in the thesis that, for the sources under consideration, perform almost as well as a linear predictor. The rate distortion performance is shown in Figure 7.1 in comparison to a standard DPCM system using the same predictor but whose integer-valued outputs were further quantized using a uniform quantizer.[2] The quantized outputs were passed to an order 0 adaptive entropy coder. Another view of this data is in Figure 7.3. We see that the DC-DPCM system outperformed the standard DPCM on all the test images. For example, on the `lena` image at 1.5 bits/pixel the DC-DPCM system provided an approximately 5 dB gain over the standard DPCM system. Granted, the performance is nowhere near the rate-distortion performance attainable using techniques like vector quantization, but the result is interesting all the same.

7.2.2 Summary

DC-DPCM was inspired by entropy-constrained vector quantization (ECVQ) (Chou 1989; Chou, Lookabaugh, and Gray 1989). In a standard vector quantizer, the decision regions are adjusted to minimize the difference between the input for a given codebook size, M. It is assumed that the codeword indices are put directly onto the channel and

[2]While this quantization method is sub-optimal, since both the DC-DPCM predictor and the DPCM predictor operated in the same way, this was considered a fair comparison.

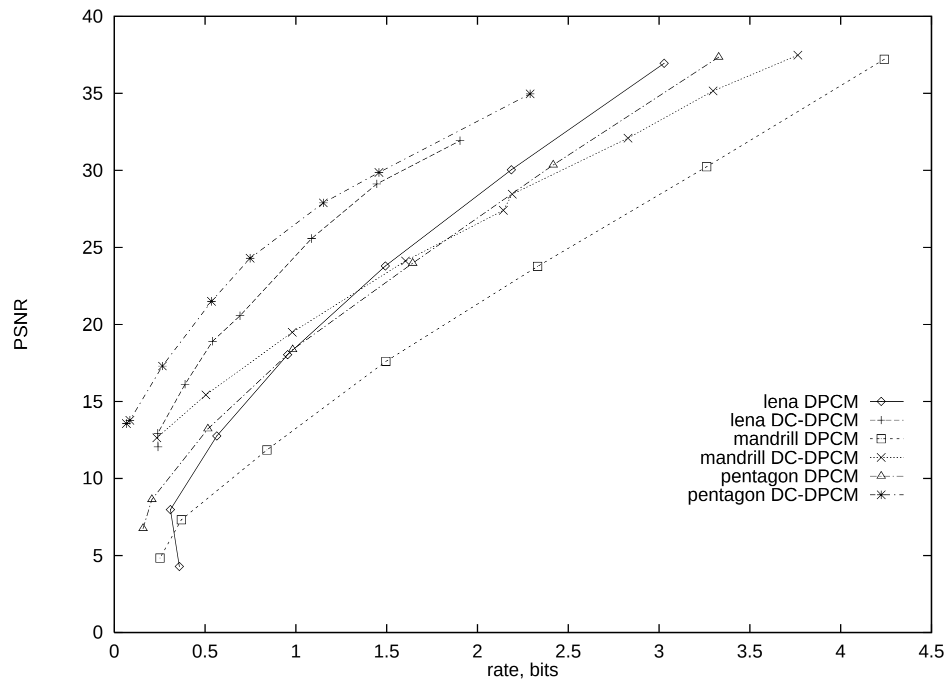

Figure 7.1: Rate-distortion for DC-DPCM *vs* standard DPCM on test set

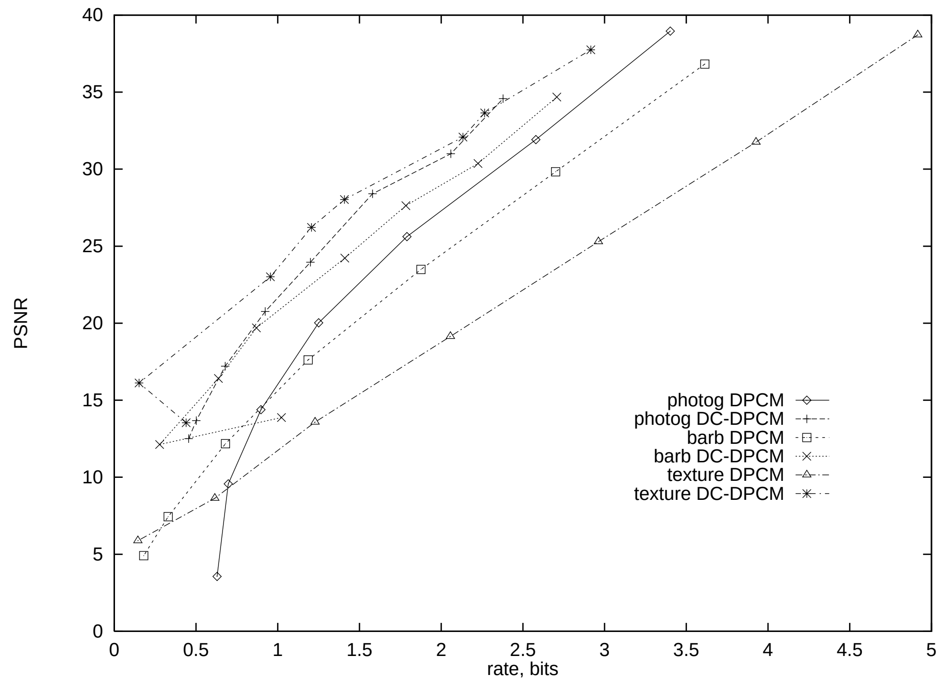

Figure 7.2: Rate-distortion for DC-DPCM *vs* standard DPCM on test set

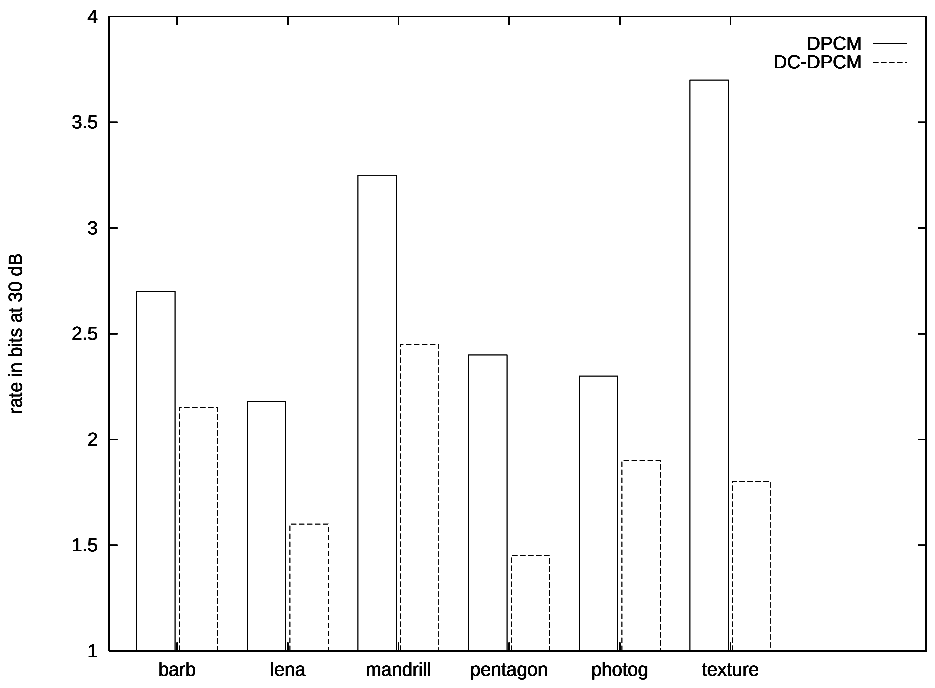

Figure 7.3: Rate at 30 dB for DC-DPCM *vs* standard DPCM

so the achievable rate is log M bits. To differentiate the techniques, Chou (1989) calls this first method *rate-constrained* vector quantization. In contrast, ECVQ explicitly assumes that the vector quantizer will be followed by an entropy coder which produces variable length channel symbols and hence, instead of minimizing the distortion given a fixed rate, the distortion is minimized subject to a constraint on the entropy. In this way, choosing the parameter λ allows one to choose relatively high distortion at very low entropy, very low distortion at very high entropy, or somewhere in between.

In DC-DPCM, we try to find the smallest rate given a restriction on the distortion by assuming that the channel symbols are not the actual values of the residual, but rather, as in ECVQ, the variable length strings produced by an entropy coder. In traditional DPCM, distortion is introduced via the quantizer, and, while we can estimate approximately what the quantizer noise power will be for some classes of quantizers (Gersho and Gray 1992, pages 151-166), we do not know what effect the quantizer has on the entropy of the output sequence. In DC-DPCM, on the other hand, the distortion is introduced explicitly and in a controlled way.

7.3 Summary

In addition to using Markov models for generating probability distributions that drive an arithmetic coder, we can also use the models as predictors. It was shown that adaptive Markov predictors can function as well as or better than their linear model counterparts, but that most natural image data can be well represented by linear models. Markov models can be as a back-end to a DPCM system, and the FOVR and VOVR models presented in Chapter 5 are especially well suited for finding the conditioning information remaining in the signal that results in the lowest rate. The new technique of DC-DPCM was seen to offer advantages over traditional DPCM and is an interesting starting point for further research.

Chapter 8

Conclusions

Whenever adaptive Markov models are used to code data, signal or text, careful attention must be paid to choosing the number of states in the model appropriately. For stationary sources, the optimal number of states appears to increase monotonically with the number of samples thus far seen. For text data compression, variable order Markov techniques have proven to be some of the most effective, their popularity lessened only by the fact that they are computationally more complex and require more memory than dictionary techniques. For signal data compression, in addition to varying the order of the model to control the number of states, we have at our disposal other techniques helpful in solving the learning problem. The well accepted technique of splitting an r-bit input into r separate bit planes is effective in this regard. This technique reduces the number of states in the model by effectively lowering the alphabet size, at the expense of losing relationships between the bit planes which potentially could help further compress the source. In this thesis, we have seen that we can also code the original r-bit input, simultaneously varying the resolution and order of the conditioning information to obtain similar performance. Additionally, we have also shown that splitting the r-bit input into planes containing more than one bit offers some performance gain over splitting into 1-bit planes.

Throughout this thesis we have exploited the use of running models in competition and choosing between them based on their recent codelength. This technique, inspired by (Williams 1991), was shown to be an application of the MDL principle. The principle can be applied to whole models (as in FOVR-II), or to individual contexts within a model (as in VOVR). The work in Chapter 4.3 on variable decay rate modelling showed that the MDL principle can be applied temporally as well, allowing a model to use information about its own recent performance as a method of altering its parameters.

This thesis has only scratched the surface of the number of issues involved in modelling of signal data. Some suggestions for further research are presented in the next section.

8.1 Suggestions for Further Research

VOVR with Zero-Resolution Branches The VOVR algorithm of Section 5.5 does not have the ability to completely skip conditioning information. In the original implementation, it was expected that assigning only one bit of resolution would be enough to skip conditioning information that does not significantly help compression, but in fact this choice makes VOVR suffer from the same problem as other tree-structured algorithms. A modification allowing VOVR to completely skip conditioning information by assigning it zero bits of resolution may solve this problem.

Non-stationary sources The proposed enhancement to the FOVR-II algorithm mentioned in Section 5.4.1 regarding the performance of the algorithm on non-stationary sources, should be pursued. Namely, the algorithm should be able to both increase and decrease the state weight of the models it is able to create in response to the input. This raises, also, several more issues about the non-stationarity of the input data with respect to the model which would be interesting to pursue.

Theory The discussion in Section 2.1.2 is intriguing in that it is at least possible to write down an expression for the expected value of the codelength as a function of the joint distribution of the source and the number of samples processed. With further investigation, there may found be some distribution-independent function which tells us how to best select a wide range of parameters ranging from the bin width of the order 0 histograms, through the resolution of the conditioning information, to the order of the conditioning information.

DC-DPCM with Human Visual System Constraints The distortion-constrained DPCM system described in Section 3.1.7 uses the mean-squared error as the distortion measure. We would speculate that using a distortion measure that takes into consideration the human visual system would produces subjectively better results than the DPCM system, and perhaps acceptable results at lower rates.

Appendix A

The Test Sources

This chapter gives some basic information about the sources in the test set.

Table A.1: Statistics of the test sources

source	size	H_0	H_1	min	max	$\bar{x}$	σ_x	r_{01}	r_{10}	r_{11}
`barb`	512×512	7.63	5.80	12	246	117.4	54.7	0.955	0.892	0.877
`mandrill`	512×512	7.36	6.15	0	230	129.1	42.4	0.751	0.863	0.716
`texture`	512×512	6.80	5.95	0	233	158.6	31.9	0.859	0.790	0.711
`pentagon`	512×512	6.80	5.23	46	238	134.3	30.9	0.892	0.891	0.815
`lena`	256×256	7.45	5.05	11	252	122.7	47.8	0.960	0.927	0.877
`photog`	256×256	7.21	4.59	4	255	141.1	87.2	0.960	0.827	0.900

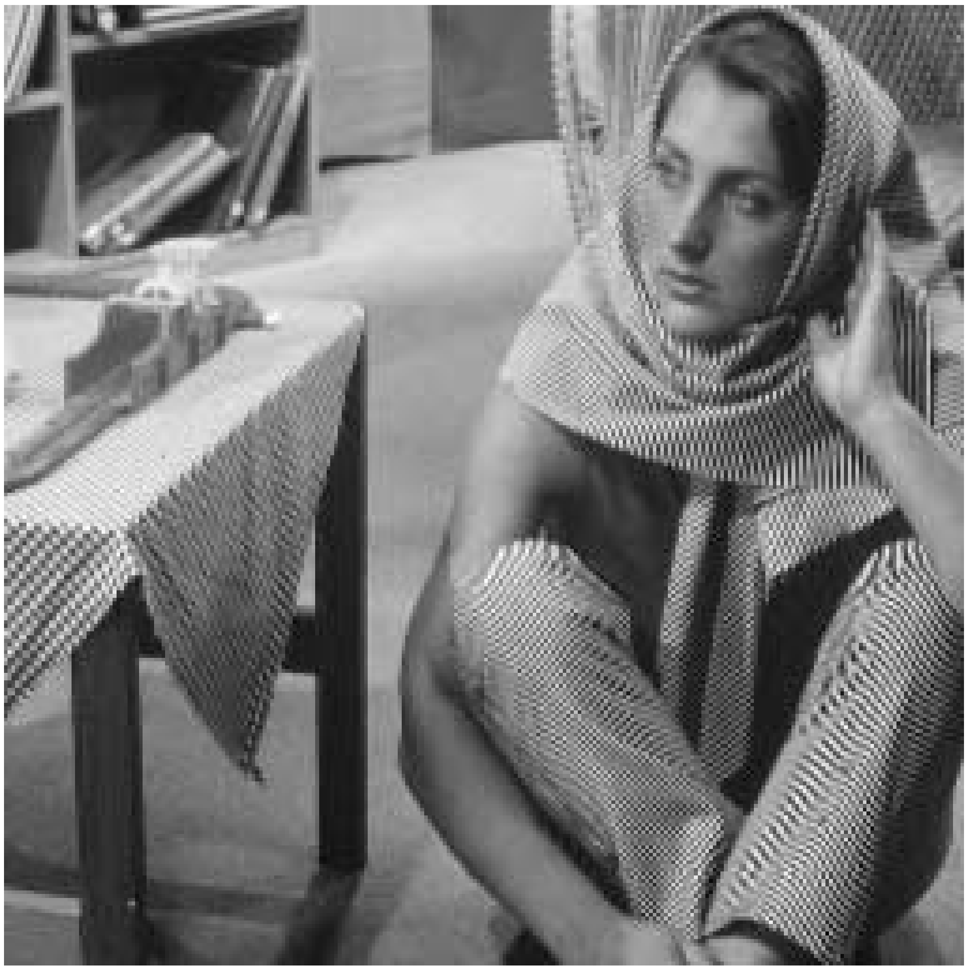

Figure A.1: barb

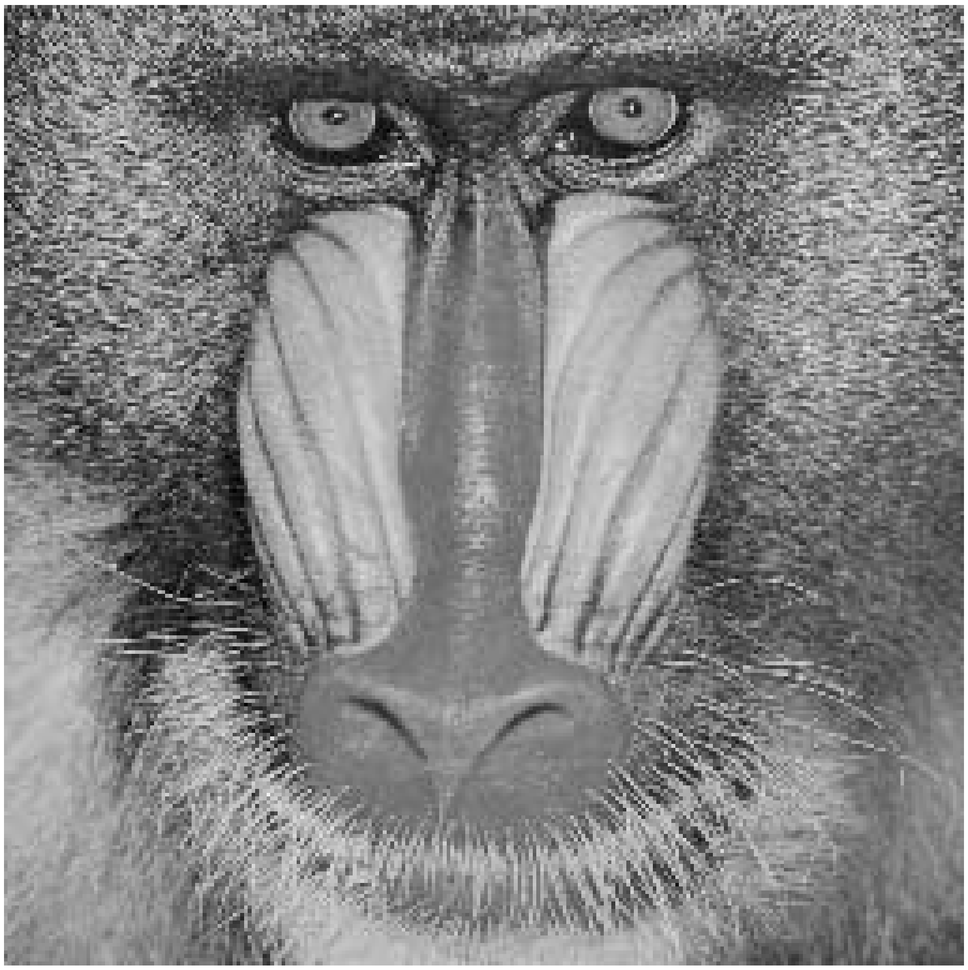

Figure A.2: `mandrill`

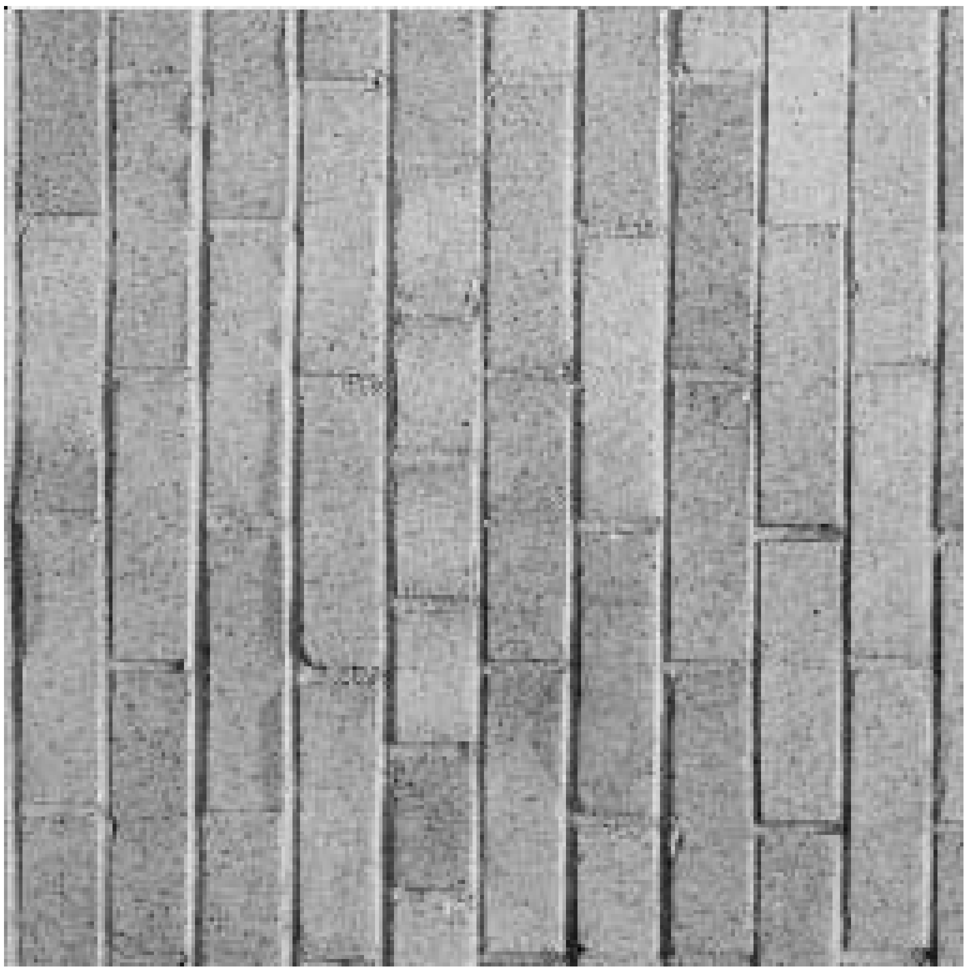

Figure A.3: `texture`

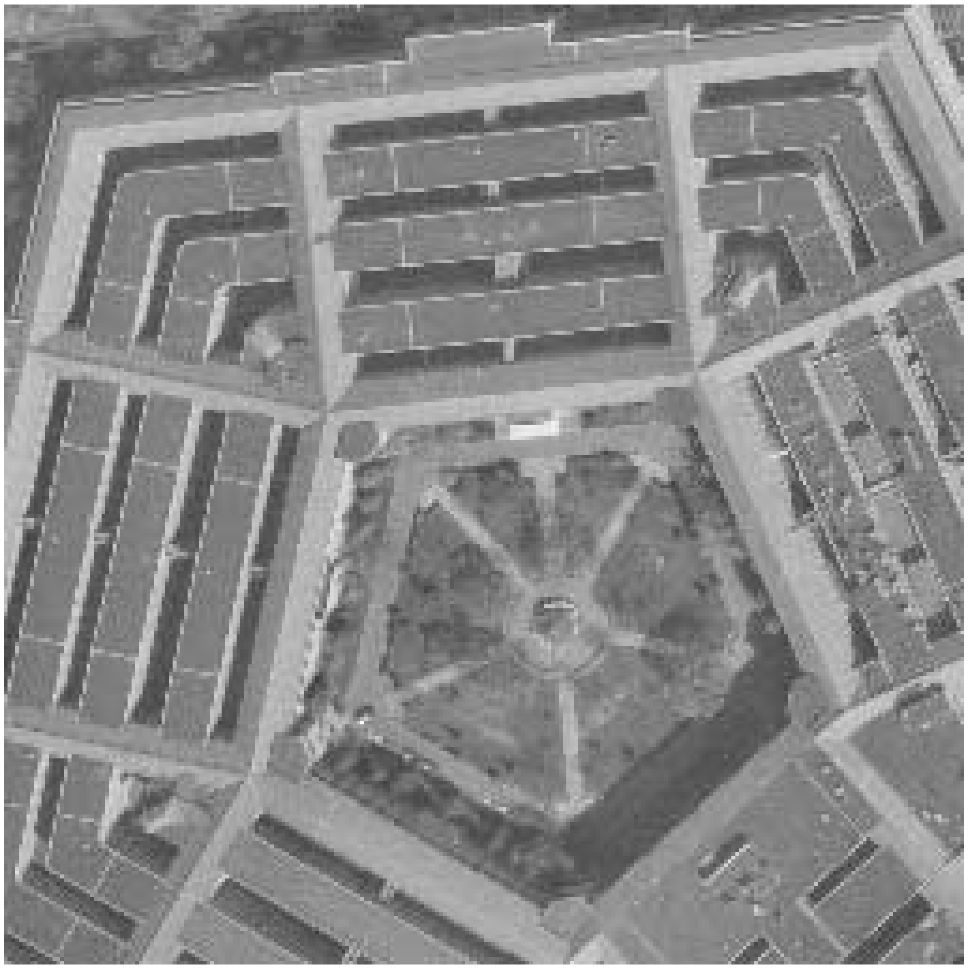

Figure A.4: `pentagon`

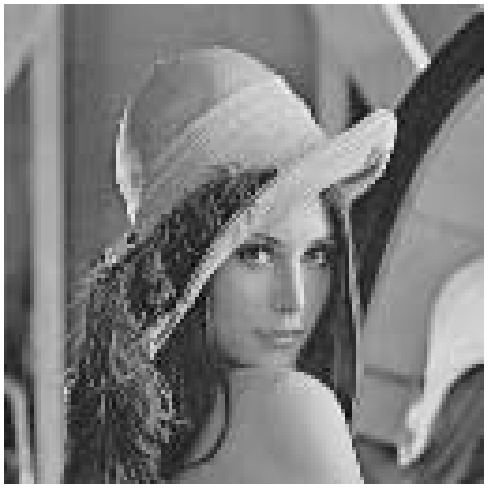

Figure A.5: `lena`

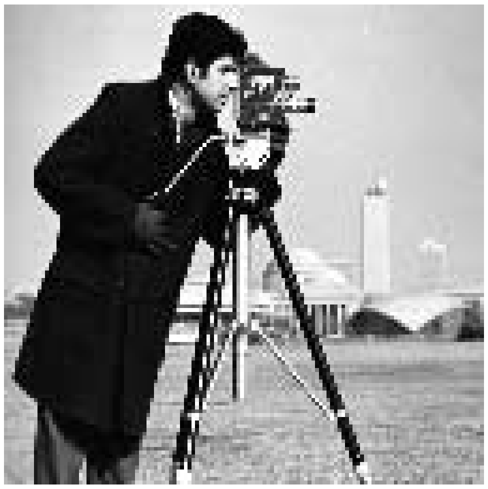

Figure A.6: `photog`

Appendix B

Algorithms

This appendix describes the details of the UMC algorithms of Rissanen and Furlan, and the MMDC algorithm of Williams.

B.1 UMC Rissanen

The context tree is constructed in the following way for a binary alphabet, with the extension to non-binary alphabets being straightforward. Given a sample $x = x_1x_2x_3\cdots$,

1. Declare the context tree of the first symbol x_1 to be the 1 leaf tree $T(0)$, where the only node, the root, is marked with the pair of counts $(c(0,\emptyset), c(1,\emptyset)) = (1,1)$ These counts are initialized to zero.

2. Let $T(t-1)$ be the most recently constructed tree. After the next symbol x_t is observed, generate the next tree $T(t)$ as follows. Climb the tree $T(t-1)$, starting at the root with $i = 1$ and taking the branch left for $z_i = 0$ and right for $z_i = 1$, indicated by each of the successive symbols in the past sequence $z = \sigma(x^t)$. For each node s visited, increment the component count $c(x_t, s)$ by one. Continue incrementing i until a node w is reached whose count $c(x_t, w) = 1$ before the update.

3. If w is an internal node, with $w0$ as the left and $w1$ as the right successor, increment the component counts, $c(x_t, w0)$ and $c(x_t, w1)$ by one and define the resulting tree to be $T(t)$. Goto Step 2.

Table B.1: Parsing the string 100011010

t	x^t	x_t	z
1	1	1	$\emptyset$
2	10	0	1
3	100	0	01
4	1000	0	001
5	10001	1	0001
6	100011	1	10001
7	1000110	0	110001
8	10001101	1	0110001
9	100011010	0	10110001

4. If w is not an internal node (it is a leaf), extend the tree by creating two new leaves $w0$ and $w1$. Let $u = x_t$ and let u' be the opposite symbol of u. Assign the same counts to both leaves: $c(u, w0) = c(u, w1) = 1$ and $c(u', w0) = c(u', w1) = 0$. Call the resulting tree $T(t)$. Goto Step 2.

After processing a source with this algorithm, each level the resulting tree corresponds to a Markov model of a different order. The root node corresponds to an order 0 model, the nodes at a depth of one in the tree correspond to an order 1 model, and so on.

The count for a symbol x at node s, $c(x, s)$ in the context tree is really one more than the number of times the symbol x was seen in context s. Hence to use the Laplacian probability estimator all the counts that are greater than zero must first be decremented.

As an example, let us parse input $x = 100011010$ using the identity permutation. The tree starts out with one node, initialized with the count $(1, 1)$. When the first symbol, $x_1 = 1$, is read, immediately the first node satisfies the requirement $c(x_t, w) = 1$. After incrementing the root count to $(1, 2)$, two new nodes, each with the count $(0, 1)$, are created. The resulting tree is shown in Figure B.1. For the next symbol, $x_2 = 0$, again, immediately the first node satisfies the requirement $c(x_t, w) = 1$. This time, however, the node is not a leaf, so no new children are created. After the root is updated, the existing child counts for the symbol $x_t = 0$ are incremented. The resulting tree is shown in Figure B.2. For the next symbol, $x_3 = 0$, the root node does not satisfy the condition $c(x_t, w) = 1$. So the tree is traversed according to the

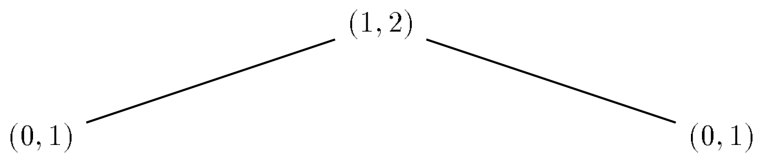

Figure B.1: Tree T(1)

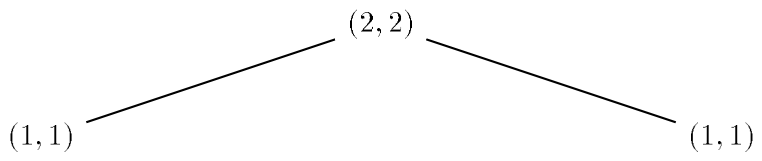

Figure B.2: Tree T(2)

history. Referring to Table B.1, at $t = 3$, $z_1 = 0$, so, after the root node is updated, the left branch is taken. At this node $c(x_t, w) = 1$, so, after updating the parent node count, two new nodes, each with the count $1, 0$ are created. The final tree is shown in Figure B.3. Using only the previous description, the tree can grow without bounds. Of necessity, however, the tree depth must be limited. In practice, an arbitrary upper limit on the tree depth is set and Step 4 is modified so that the tree grows only if the maximum tree depth would not be exceeded.

The tree resulting from running the previously described algorithm on the string x^t is $T(t)$. This tree can be used to choose optimal model order for coding the next symbol, x_{t+1}. To choose the coding node, climb the tree starting at the root with $i = 1$ according to the string $z = \sigma(x^t)$. Each z^i defines a node, which is a possible context for coding the symbol x_t. If we had chosen the node z^i to code the previous symbol occurrences at this state, we would have obtained the length

$$L_{\text{parent}}(z^i) = \log \frac{\prod_j c(j, z^i)!(M-1)!}{(c(z^i) + M - 1)!} \; . \tag{B.1}$$

from (3.3) and where j runs through all the symbols. On the other hand, if we had chosen the M children (where M is the alphabet size) $\{z^i0, z^i1, \cdots, z^iM-1\}$ as the

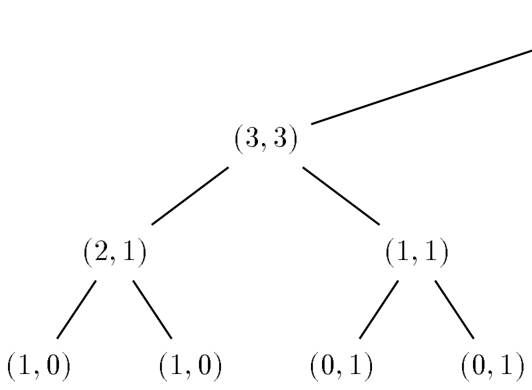

Figure B.3: Final tree for the string 100011010, UMC Rissanen

contexts, we would have spent

$$L_{\text{children}}(z^i) = \sum_{j=0}^{M-1} L(z^i j) \tag{B.2}$$

for encoding the same symbols. So, we choose the node z^i if $L_{\text{parent}}(z^i) < L_{\text{children}}(z^i)$, otherwise we increment i until a decision is made.

B.2 UMC Furlan

The enhancement to the UMC algorithm as proposed by Furlan (1991) is as follows:

1. The context tree for the first symbol x_1 is the 1 leaf tree $T(0)$, consisting solely of the root node, with all counts equal to zero; this is the zero information case.

2. When the next symbol $u = x_t$ is observed, climb the previous tree, starting at the root and taking the branch corresponding to each of the past symbols σx^t, where σ is the permutation as defined in the algorithm of (Rissanen 1986a).

3. For each node z visited, increment the count $c(u, z)$

4. Additionally, at each node visited, calculate the entropy of the symbol (the per-symbol codelength) u and add the difference in entropy between the child's

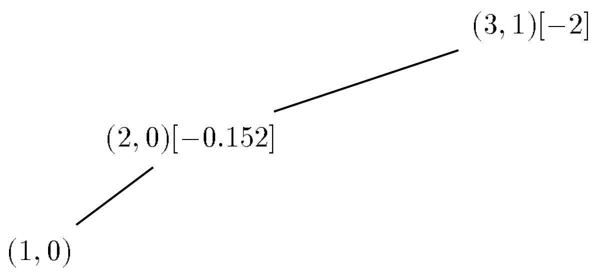

Figure B.4: Tree T(4)

symbol entropy and the parent's symbol entropy to the child's REC. Limit the resulting REC so that it falls within (RECMIN, RECMAX).

5. Continue until the terminal node; if, at that node, the count for the current symbol is greater than 1 (indicating it is a coding node) AND its REC is negative (indicating that it is a better coder than its parent) grow the tree.

As in Section B.1, let us process the string $x = 100011010$. We will add to our representation of the node the value of the REC in square brackets. For the purpose of this example, we choose RECMAX=2 and RECMIN=-2. The initial tree is the node $(0, 0)[-2]$ corresponding to the null context.

The first symbol,1, is coded using the null context, whose REC is never updated. Since, after the update, the 1 count is only 1, no tree growth occurs. The second symbol,0, is likewise coded with the null context, and the counts updated, but again, the count is not greater than 1 after the update. The third symbol is coded with the null context, and this time, after the update, the 0 count is 2 and the tree grows context [0/1]. Since the REC of the [0/1] context is non-negative, we again choose the null context for the fourth symbol, 0. The counts are updated and whereas the null context assigned $p = 3/5$ to the symbol, the [0/1] context would have assigned $p = 2/3$, the difference in codelength being 0.152 bits, in favor of the [0/1] context. Also, because the count for the current symbol is greater than 1 at the [0/1] context, and because its REC is negative, the [0/10/1] context grows, according to the history. Hence after the update, the tree is as shown in Figure B.4. The final tree is shown in Figure B.5.

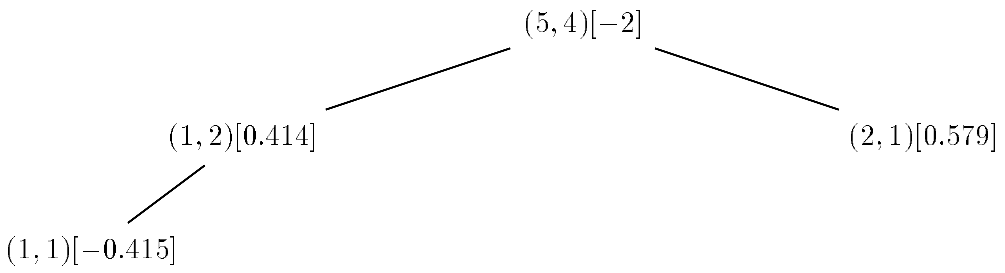

Figure B.5: Final tree for the string 100011010, UMC Furlan

Appendix C

Estimation Techniques

C.1 Introduction

A fundamental characteristic of all learning systems (including human learning) is the significant dependence of the nature of the inferences made by the system on the nature of the training. In this chapter, we will describe several common approaches to the seemingly simple task of estimating a distribution from a histogram of a sample. Usually, when there are many instances in the sample, the task is easier. However, when there are just a few, as is often the case when we are using high order Markov models, then the task becomes more difficult. For example, assuming a 256 symbol alphabet, having seen just two samples, say 152 and 199, what is our best guess for the probability distribution? What should be our estimate of the probability of the symbols that have not occurred?

Finding answers to the previous questions has practical significance for statistical data compression. Since we assume that the information conveyed by an event is the negative logarithm of its probability (as in (2.3)), assigning zero probability to an event leads us to a singularity. Instead, we need to generate "safe", that is, positive and non-zero, predictions. Roberts (1982) calls this the zero frequency problem.

The discussion that follows is based on (Williams 1991). We will assume that we have an alphabet A of size M. The histogram count for symbol a is denoted by $C(a)$, the total count of all symbols seen so far is denoted by C, and the number of symbols with zero frequency ($C(a) = 0$) is denoted by z.

C.2 Linear Estimation

Linear estimation allocates a small amount of probability to all symbols in the alphabet (whether they have non-zero counts or not) and then divides the remaining probability between the symbols with non-zero counts. According to Williams (1991), it can be shown that linear estimation with $\lambda = M$ is optimal if all possible probability distributions are equally likely, but that in practice, $\lambda = 1$ works the best.

C.2.1 General Linear Estimation

The general linear estimator is given by

$$\hat{p}(a) = \frac{C(a) + \lambda/M}{C + \lambda} \tag{C.1}$$

where $\lambda > 0$. The smaller the value of λ, the more we trust the histogram to be representative of the true distribution. The higher the value of λ, the more we expect a previously unseen symbol to appear.

C.2.2 Linear Moffat Estimation

Linear Moffat estimation allows the parameter λ to vary, effectively decreasing it if the histogram has a large number of zero frequency symbols, and increasing it if the histogram has a small number of zero frequency symbols. In this way, λ is modified according to the spikiness or smoothness of the distribution, dynamically adapting to the characteristics of the distribution.

The linear Moffat estimator is given by

$$\hat{p}(a) = \frac{C(a) + \lambda(M - z)/M}{C + \lambda(M - z)} \; . \tag{C.2}$$

C.2.3 Laplacian Estimation

Choosing $\lambda = 1$ in (C.1) results in the familiar Laplacian estimator. As previously mentioned, in practice, this is the best fixed choice for λ.

C.3 Non-Linear Estimation

Non-linear estimation divides the probability in two parts. The first part is divided equally between the symbols of zero frequency and the second part is divided between the symbols of non-zero frequency in proportion to their frequency. Non-linear estimation is useful in situations where there are many zero frequency symbols which, under linear estimation, would needlessly take away more of the available probability mass.

C.3.1 General Non-Linear Estimation

The general non-linear estimate is given by

$$\hat{p}(a) = \begin{cases} \frac{1}{Z}\frac{\lambda}{C+\lambda} & C(a) = 0 \\ \frac{C(a)}{C+\lambda} & C(a) > 0 \end{cases} . \tag{C.3}$$

C.3.2 Non-Linear Moffat Estimation

The non-linear Moffat estimator is given by

$$\hat{p}(a) = \begin{cases} \frac{1}{Z}\frac{\lambda(M-z)}{C+\lambda(M-z)} & C(a) = 0 \\ \frac{C(a)}{C+\lambda(M-z)} & C(a) > 0 \end{cases} . \tag{C.4}$$

Appendix D

Prefix Coding of Integers

Given an integer n we would like to form a prefix code. A method for doing this (Rissanen 1989) is described below.

Each integer n has a binary representation $b(n)$ consisting of $l(b(n))$ bits, the *length* of the binary representation $b(n)$. However, if the binary representation for an integer is followed by the representation for another integer, there is no way for a decoder to know when one integer stops and another begins. One might think of prepending the length of $b(n)$, $l(b(n))$, to $b(n)$, but that results in the same problem – except that the length is smaller than the original number. Continuing in the same way, we end up with a monotonically decreasing sequence of integers as in Table D.1. The problem is how to encode the smallest one. One solution is to let a stored integer j indicate that the next $j+1$ positions contain the next length indicator. The resulting prefix code $w(n)$ for some small integers is shown in Table D.2.

A simple encoder that outputs the prefix code for the integer p from left to right is given by

Table D.1: Attempt at forming a prefix code for the integer $n = 10256$

n	$b(n)$	$l(b(n))$
10256	10010000010000	14
14	1110	4
4	100	3
3	11	2
2	10	2

Table D.2: Prefix code for some small integers

n	$w(n)$
1	0
2	10 0
3	11 0
4	10 100 0
5	10 101 0
6	10 110 0
7	10 111 0
8	11 1000 0
9	11 1001 0
10	11 1010 0
11	11 1011 0
12	11 1100 0
13	11 1101 0
14	11 1110 0
15	11 1111 0
16	10 100 10000 0

```
1. output 0
2. if $p < 2$ quit
3. output $p$, set $p = w(p) - 1$, and goto step 2
```

and a simple, recursive prefix-coded integer decoder is given by

```
1. initialize $p = 1$
2. read 1 bit, $b$
3. if $b = 0$, report $p$ and quit
4. read $p$ more bits, interpret the resulting $p + 1$ bits as the new $p$
   and goto step 2
```

The length of a prefix-coded integer can be shown to be

$$L^*(n-1) = \log c_K + \log n + \log\log n + \cdots \ , \tag{D.1}$$

where the sum includes all positive terms and c_K is a constant so that the Kraft inequality is satisfied with equality, $c_K = 2.865$.

References

Bell, T., J. Cleary, and I. Witten (1990). *Text Compression.* Prentice Hall.

Chou, P. (1989). Application of entropy-constrained vector quantization to waveform coding of images. In *Proceedings SPIE, Visual Communications and Image Processing*, pp. 970–987.

Chou, P., T. Lookabaugh, and R. Gray (1989). Entropy-constrained vector quantization. *IEEE Transactions on Acoustics, Speech and Signal Processing 37*, 31–42.

Cleary, J. and I. Witten (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications 32*, 392–402.

Farvardin, N. and J. Modestino (1984). Optimum quantizer performance for a class of non-Gaussian memoryless sources. *IEEE Transactions on Information Theory 40*, 485–497.

Feller, W. (1957). *An Introduction to Porbability Theory and its Applications.* New York, New York: Wiley.

Furlan, G. (1990). *Contribution a l'Étude et au Développement d'Algorithmes de Traitement du Signal en Compression de Données et d'Images.* Ph. D. thesis, Université de Nice-Sophia Antipolis.

Furlan, G. (1991). An enhancement to universal modelling algorithm context for real-time applications to image compression. In *IEEE Proceedings of ICASSP*, pp. 2777–2780.

Gersho, A. and R. Gray (1992). *Vector Quantization and Signal Compression.* Kluwer Academic Publishers.

Hampel, H. (1992). Technical features of the JBIG standard for progressive bi-level image compression. *Signal Processing: Image Communication 4*, 103–111.

Ho, Y. and A. Gersho (1989). Classified transform coding of images using vector quantization. In *IEEE Proceedings of ICASSP*.

Howard, P. and J. Vitter (1991). New methods for lossless image compression using arithmetic coding. In J. Storer and J. Reif (Eds.), *Proceedings, Data Compression Conference, Snowbird, Utah*, pp. 257–266.

Huffman, D. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E 40*(9), 1098–110.

Joint Bi-level Image Experts Group (1992, April). *CCITT Draft Recommendation T.82/ISO Draft International Standard 11544*. Joint Bi-level Image Experts Group.

Kolmogorov, A. (1965). Three approaches to the quantitative definition of information. *Problems of Information Transmission 1*, 4–7.

Langdon, G. and J. Rissanen (1981). Compression of black-white images with arithmetic coding. *IEEE Transactions on Communications 29*, 858–867.

Li, M. and P. Vitanyi (1992). Inductive reasoning and Kolmogorov complexity. *Journal of Computer and System Sciences 44*, 343–384.

Maragos, P., R. Schafer, and R. Mersereau (1984). Two-dimensional linear prediction and application to adaptive predictive coding of images. *IEEE Transactions on Acoustics, Speech and Signal Processing 32*, 1213–1229.

Oppenheim, A. and R. Schafer (1989). *Discrete-Time Signal Processing*. Signal Processing. Englewood Cliffs, New Jersey: Prentice Hall.

Rissanen, J. (1983a). A universal data compression system. *IEEE Transactions on Information Theory 29*, 656–664.

Rissanen, J. (1983b). A universal prior for integers and estimation by minimum description length. *The Annals of Statistics 11*, 416–431.

Rissanen, J. (1986a). Complexity of strings in the class of Markov sources. *IEEE Transactions on Information Theory 32*, 526–532.

Rissanen, J. (1986b). Stochastic complexity and modelling. *The Annals of Statistics 14*, 1080–1100.

Rissanen, J. (1989). *Stochastic Complexity in Statistical Inquiry.* World Scientific.

Rissanen, J. and G. Langdon (1981). Universal modelling and coding. *IEEE Transactions on Information Theory 27*, 12–23.

Rissanen, J. and K. Mohiuddin (1989). A multiplication-free multialphabet arithmetic code. *IEEE Transactions on Communications 37*, 93–98.

Roberts, M. (1982). *Local Order Estimating Markovian Analysis for Noiseless Source Coding and Authorship Identification.* Ph. D. thesis, Stanford University.

Solmonoff, R. (1978). Complexity-based induction systems: Comparisons and convergence theorems. *IEEE Transactions on Information Theory 24*, 422–432.

Teuhola, J. and T. Raita (1991). Piecewise arithmetic coding. In J. Storer and J. Reif (Eds.), *Proceedings, Data Compression Conference, Snowbird, Utah*, pp. 33–42.

Tischer, P., R. Worley, A. Maeder, and M. Goodwin (1993). Context-based lossless image compression. *The Computer Journal 36*(1), 69–77.

Todd, S., G. Langdon, and J. Rissanen (1985). Parameter reduction and context selection for compression of greyscale images. *IBM Journal of Research and Development 29*(2), 188–193.

Williams, R. (1991). *Adaptive Data Compression.* Kluwer.

Witten, I., R. Neal, and J. Cleary (1987). Arithmetic coding for data compression. *Communications of the ACM 30*, 521–537.

Ziv, J. and A. Lempel (1978). Compression of individual sequences via variable rate encoding. *IEEE Transactions on Information Theory 24*, 530–536.