Context-based Compression Algorithms for Text and Image Data

Wong Ling

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Philosophy in Information Engineering at the Chinese University of Hong Kong.

May 1997

Contents

List of Figures

List of Tables

Abstract

A large volume of data is generated daily in both the public and the private sectors. Full and timely utilization of this large amount of files requires efficient archival and retrieval. Coding the files in less space with no loss of essential information, data compression have a great impact on the transmission and storage problem.

The basic problem in data compression is to determine the characteristics of the essential information in the original data set, and to design a compression scheme which preserves this essential information while maximizing the compression ratio. Employing the internal regularity of data, we can construct data compression models to predict data patterns in advance, or recognize repeated data from earlier data. This thesis shows how to find out the structure existing among the data and how to apply it to compression.

Due to the strong dependency between neighboring symbols of data, prior information — context is an important component for estimating or finding out the redundancy. In this thesis context is applied in the prediction rather than probabilities estimation. It considers processing data as learning, which provides more information for representing the data files. A robust theoretical framework, content prediction, has been developed in this thesis that relies heavily on above idea, and relates closely to learning. Besides, some new evolutionary branch is added to the content prediction and source coding, by the introduction of symbol remapping, bounded length block sorting and locally adaptive entropy coding. Applications and properties of the prediction and coding are presented, and illustrated by simulations.

The second main topic addressed in this research is context-based coding for image data. It is well known that image representation generally contains a lot of redundant pixels, which, if removed, do not affect the reconstruction fidelity of the image. Previously, all the redundant pixels were estimated or coded with low order conditional coding. In this work, a study on the types of context is performed, and some methods which using more pixels for estimation are proposed. It can be shown that the compression rate can be improved by these new thechniques.

Computer simulations show that for text and image data, the proposed coding schemes substantially improve the results obtained by previous algorithms, and performs better than the classical coders.

1. Introduction

A large volume of data is generated daily in both the public and the private sectors. These data include mammograms, scans, imagery, fingerprints, satellite imagery, commercial radar, etc. In addition, digital scanners and other devices are utilized to convert documents to digital arrays which can be represented on a computer.

Such data need to be transmitted over various channels of communication and stored on computers and other large databases utilizing compatible storage devices (CD's, external disks, etc.). Storage and transmission of such data require large capacity and bandwidth that can be cost prohibitive. Hence, full and timely utilization of this large database of files requires efficient archival and retrieval. Data compression where files can be stored in less spaces with no loss of essential information will have a great impact on these problems.

A small example will clearly illustrate the need for data compression of images and text data. The popularity of CD-ROM is mainly due to its high storage capacity, probably outperforming every other media in a cost/performance ratio. If a sequence of say, 640×480 pixel images with 24-bit per pixel resolution and a frame rate of 24 frames per second was to be stored on a CD-ROM. Then 30 seconds of this sequence would fill the entire 600M bytes disk up. To store a 90 minute file with frames of the specification mentioned above a reduction of data to approximate 180:1 is required.

So even with one of most dense storage capacity medial commercially available, the need for data compression of images and text data is evident. The quest is now to find proper data compression performing well with respect to the obtained compression ratio, the obtained image quality and the time usage of the encoding and decoding process. Compression for text and image data, it has very different approach.

Data compression techniques are concerned with reduction of the number of bits required to store or transmit data without any appreciable loss of information. The process of compressing data sets can be classified into "lossy" or "lossless" techniques. Techniques from both these classes can be used in a single compression scheme to achieve high compression ratios (low number of bits per pixel).

A perfect "lossless" compression scheme is one which retains all of the original data. That is, following the compression and decompression cycle the original data is reconstructed with no loss of information. Lossless compression techniques are usually used for storage and transmission purposes. One drawback of lossless compression techniques, however, is that the compression ratios that can be achieved are not very high. For some important data, such as commercial

records, lossless coding methods are necessary.

A lossy compression scheme, on the other hand, is one in which the compression or decompression cycle results in a loss of some of the original data. In addition to storage and transmission, lossy compression techniques can be utilized for denoising, removal of redundancy, and feature extraction. Thus, properly configured lossy compression schemes may actually increase the signal-to-noise ratio of the compressed data as compared to the original data, if only nonessential information in the data is lost. Another significant advantage of properly configured lossy compression schemes is that they allow much higher compression ratios than lossless compression techniques.

The basic problem in data compression is to determine the characteristics of the essential information in the original data set, and to design a compression scheme which preserves this essential information while maximizing the compression ratio. The best data compression model will predict data patterns in advance, or recognize data repeated from earlier data.

Although text compression and image coding have different approaches, the core of coding models is to find out the redundancy among the data based on its internal regularity. The strong dependency between adjacent symbols of data is usually expressed as a Markov model, with the probability of the occurrence of a particular symbol being expressed as a function of the preceding *n* symbols (an order-*n* Markov model, or a context of *n* symbols).

Prior information, *context*, is the basic component for estimating or finding out the redundancy. In most compression algorithms right now, context is combined with frequencies and probabilities.

The most frequent context modeling technique is Prediction by Partial Matching (see Chapter 2.3.1.1). For each symbol to be encoded, a typical PPM compressor will considers say the preceding 4 symbols (an order-4 context) and determines the probabilities of the symbols which it has already seen in that context. If the desired symbol has not seen in the order-4 context, the coder will "escape" down to lower orders until it determines a context from which the symbol can be emitted. An escape to a lower order appears as a symbol to the higher order, but one whose probability is not easily determined. Various heuristics are available to determine the escape probability and these largely differentiate the different PPM compressors and determine their relative qualities. However the handling of escapes or notification of the correct order remains one of the major problems in PPM compression. The data structures to represent the known contexts may be quite complex and subtle and some may require large amounts of data storage and computing time.

Actually, context can be applied in another field : *prediction*. It considers processing data as

learning, which provides more information for represent the data files. A robust theoretical framework has been developed that relies heavily on above idea, and relates closely to learning.

1.1 Motivation

Transformation or representation is a key component in many tasks in computer vision and image processing, and it has gained increasing attention on text in recent years. It consists generally of presenting a data in a form, different from the original one, in which desired characteristics are emphasized that can be more easily accessed.

For text data, many of today's popular lossless data compression algorithms are based on the sequential data compression published by Lempel and Ziv in 1977 and 1978. They are extended by combining with statistical coding, but the fundamental algorithms remain the same. Other methods include "Prediction by Partial Matching" (PPM) which were first developed in the 1980s. Recent improvements can be found in PPMD+, PPM* and PPMZ. In image coding, the quantized coefficient image is encoded due to the low order context model. In all the classical methods, context is combined with frequencies and probabilities. Actually, context can be applied in another field : *prediction*. In this thesis, I consider context-based prediction methods for both text and image data.

For text data compression, contexts are collected for content prediction. To process a character, the predictor first finds the best matching context, using this context index as an anchor into the context dictionary. Having found the anchor, it searches similar contexts for the matching to-be-processed symbol. If the to-be-processed symbol is found and occurs a distance *d* contexts from the anchor, the just inputting symbol is represented by *d*. The details information about prediction scheme can be found in the Chapter 3, 4 and 5.

In image coding, general compression system consists of three stages : representation, quantization and finally a lossless encoding. Typically the last coding procession can be divided into two components : modeling and entropy coding. The goal of modeling is to predict the distribution to be used to encode each pixel and context infromation is included into the system in this part.

1.2 Original Contributions

In the compression models, symbol context can use for prediction or estimation. Both approaches use the relationship between context and content. Symbol context is the prior information that has been encoded, symbol content is the strings that will be encoded. Statistical coding methods are using the context to estimate the probabilities of the inputting symbols. Lempel-Ziv algorithms find the repeated context in the symbol content. Prediction uses the symbol context as a content predictor for guessing the coming symbols.

In this thesis, context-based algorithms are proposed for both prediction and estimation (coding). The following are the main contributions of this thesis :

1. Concerning the framework of context-based prediction :

   (a) A generalized framework of *Content Prediction* is proposed.
   (b) A framework for Content Prediction operation in Block basis, *Bounded-Length Block Sorting*, is introduced.
   (c) A new context-based model is suggested for the embedded zerotree wavelet image coding.

2. Concerning the context-based coding :

   (a) Alternatively orderings on the symbol table, *Symbol Remapping*, are proposed for context collecting algorithms.
   (b) *Locally adaptive frequency ordering* are proposed in Bounded-Length Block Sorting.
   (c) Arithmetic Coding in 3-ary model are suggested for high skewed output distribution.

1.3 Thesis Structure

Chapter 2 surveys research related to the lossless compression, and partitions it into three areas : Information Theory, Early Compression and Modern Techniques for Compression. In the first part, the nature of information and entropy measure is examined. A brief history of data compression is given in the Early Compression, leading to the distinction between a model and a coder. Besides, some source codes are briefly overviewed here. A variety of modern compression algorithms are presented in last section.

Chapter 3 describes a new avenue for improving the compressibility of text. The main idea is that changing the ordering of the symbols for compression benefit. Different kinds of symbol ordering methods are testing as preprocessor on Block Sorting algorithm. Detailed introduction of the Block Sorting algorithm can be found here.

Chapter 4 introduces a general framework of the Content Prediction lossless algorithm. Content Prediction model uses prediction and ranking scheme. It is a extensions of Yokoo's context coding algorithm. Basic definitions about prediction, ranking and context sorting are explained before the proposed method. Then a baseline version of Content Prediction is introduced in detail. Various basic components in Content Prediction coder are extensive analysis here.

Chapter 5 acquaints a new algorithm based on the baseline Content Prediction and Block Sorting. The contexts in bounded length are used in the sorting algorithms. Thus, the system use less memory and processing time. Besides, some coding techniques are proposed to extend this algorithm.

All of the compression techniques discussed through chapter 5 are "lossless". Lossy methods can be used on image and video, and they are capable of achieving dramatically higher compression ratios. Chapter 6 shows how lossy compression can be used on digitized image data with techniques like wavelet transform and zerotree structure. And a context-based lossy coding based on the zerotree data structure is developed here. Some experiments is conducted to show how the context information can be included in zerotree data structure.

2. Background

Many early storage and transmission methods employed aspects of compression in a cost effective manner. For example, in the 1800s large codebooks were compiled listing common phrases and corresponding numerical codes. By transmitting the shorter codes, money was saved on the cost of telegrams; and as the codebooks had limited circulation some level of privacy was achieved.

In this chapter the techniques of lossless data compression are surveyed. Lossy coding will be introduced at Chapter 6. In section 2.1, the nature of information is examined and the entropy measurement invented by Claude Shannon is presented. Shannon's theory provides a quantitative way to measure "the amount of information". A brief history of data compression and some source codes are given in section 2.2, leading to the distinction between a model and a coder. A variety of modern compression algorithms are presented in section 2.3.

2.1 Information Theory

Data can be compressed whenever some patterns of data symbols are more likely to occur than others. Compression uses a means of encoding to eliminate that redundancy, thereby effectively reducing the size of the data traveling over a communication link or being stored in a repository. The process of compressing data sets can be classified into "lossy" or "lossless" techniques.

We are primarily concerned with lossless compression here; that is compression where the original message can be exactly reconstructed from the compressed version. Lossless compression is particularly appropriate for compressing natural languages, source code, program executables, and financial data, where they need to be reversible.

The amount of redundancy in a source is related to its entropy. In information theory, entropy is used to measure the amount of uncertainty in a system, or amount of information. Messages that occur most frequently are most redundant, and therefore, have the highest entropy. These are encoded using the shortest representations. It is impossible to give a precise entropy for any particular message since the information content depends on the observer. Nevertheless, entropy is the best guide available when it comes to specifying how many bits are required to specify a message.

More generally, suppose events $1,\cdots,n$ occur with probabilities $p_1,\cdots,p_n$ summing to one. Shannon established that the correct way to measure the entropy, is given by

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i \text{ bits.}$$

For example suppose all n events are equally likely, $p_i = 1/n$, then

$$H = -\sum_{i=1}^{n} \frac{1}{n} \log_2 \frac{1}{n} = \log_2 n.$$

From this formula one can determine that the maximum number of objects that can be distinguished in a game of twenty questions is $2^{20} = 1048576$. Further, to construct a block code for an alphabet of n symbols $\lceil \log_2 n \rceil$ bits are needed per block.

The formula for entropy can, at least in principle, be applied to any probability distribution (a slight generalization is needed to handle continuous distributions) and gives the expected number of bits required to encode an arbitrary event drawn from the given distribution. The entropy formula does not specify how the encoding is to be carried out. The invention of arithmetic coding (discussed in section 2.3.1) solves this problem by coding events arbitrarily close to their entropy.

2.2 Early Compression

For lossless compression, the original file can be faithfully restored from the compressed file. Therefore, the smaller compressed file contains all the information in the larger original file. The secret of compression lies in the better organization of files — *coding more frequent symbols with shorter codewords, and coding less frequent symbols with longer codewords.*•

Morse code (Figure 2.-1) is a good example of an early compression technique. Morse cleverly assigned the shortest codes to the most frequent symbols based on the frequencies of English letters. Intuitively, if symbols which occur frequently have the shortest codes, then on average messages will be shorter. The disadvantages of Morse code become apparent when something other than English text, say a page of numbers, needs to be transmitted. In this case the resulting message will be long because the digits have longer representations in Morse code. The more powerful techniques discussed in section 2.3 overcome this limitation by adapting dynamically

• fundamental principle of lossless compression

to a message. A second problem with Morse code is that symbols are considered in isolation; that is, without reference to the surrounding context. Better codes are produced by considering surrounding symbols. This is easily seen in English where the probability of a *u* occurring goes up dramatically just after coding a *q*.

Letters						*Numbers*	
a	.-	k	-.-	u	..-	0	-----
b	-...	l	.-..	v	...-	1	.----
c	-.-.	m	--	w	.--	2	..---
d	-..	n	-.	x	-..-	3	...--
e	.	o	---	y	-.--	4	....-
f	..-.	p	.--.	z	--..	5	.....
g	--.	q	--.-			6	-....
h	....	r	.-.			7	--...
i	..	s	...			8	---..
j	.---	t	-			9	----.

Figure 2.-1 A part of the Morse-code

In the past a variety of ad hoc techniques which took advantage of particular vagaries in the source were widely used. Although these techniques have been supplanted by adaptive compressors, a few are mentioned to illustrate the kinds of redundancy known to occur in practice.

- ***Run-Length Encoding*** In run-length encoding, consecutive identical symbols are replaced by a single instance of the symbol and a count indicating the number of times the symbol is to be repeated. The technique works best for small alphabets, particularly binary, and is widely used to compress bilevel images. Direct application of run-length encoding is ineffective for ASCII text.

- ***Differential Coding*** When compressing sorted records, it is often beneficial to store differences between consecutive records. The technique works well for static lexicons and consecutive lines of raster images.

- ***Move to Front Coding*** To simulate the book-stack with a dynamic list of words. Coding the symbol due to the ordering in the list. The coded symbol is moved to front of the list so that coding for same symbol occurs soon after with smaller order. Such technique is suitable for any localized region of the input that contains a large number of a few distinct characters [BSTW86].

The formula for entropy shows that the information content of a source is determined by the underlying probability distribution. This in turn indicates a logical distinction between the model (which specifies the probability distribution) and the actual coding of events drawn from the distribution.

2.2.1 Some Source Codes

Some source codes algorithms are introduced in followings. They apply the fundamental principle of data compression in different approaches.

2.2.1.1 Huffman code

In 1952 Huffman [HUF52] found a source code construction, of which it can be proved that it is optimal. It is a fixed-to-variable length code, of which the rate will not always reach the entropy, but falls with bounds "$H \leq L < H + 1$" and no other fixed-to-variable length source code will perform better.

First the algorithm computes the probability of every input sequence. It then combines the two least likely input sequences into one "new"-symbol. The new symbol gets a frequency count equal to the sum of the probabilities of the child symbols. The algorithm repeats the process until only one parent symbol remains. This symbol is the root of the tree. The tree is then a complete tree with a leaf corresponding to each input sequence. The codeword for an input sequence can now be found by tracing a path from the root of the tree to the appropriate leaf, and use the labels of the branches on the path as the consecutive codeword symbols.

2.2.1.2 Tunstall code

The Tunstall code [TUN68] is an optimal variable-to-fixed length for memoryless sources. No other variable-to-fixed length code will achieve a lower average codeword length. Comparison with Huffman code, Tunstall code does not map all possible sequences of source symbols to a sequence of codewords. It puts all codewords in a tree. Again we start in the root, and in every node we take the branch corresponding to the next source symbol. If the tree is complete (every node has all its children), then the leaves of this tree are a set of all possible prefixes of the source sequence. Thus the leaves of a complete tree can be used to represent a set of strings of source symbols that fulfills this condition.

The Tuntstall code is based on the idea of placing the sequences of source symbols in a tree. We will consider the binary case here. The algorithm assumes that the probabilities $p_1, \cdots, p_n$ of every events $1, \cdots, n$ is known. It is an easy two-step algorithm.

1. Start with a tree with two leaves (corresponding to the source sequences 0 and 1).

2. Repeat until the needed amount of source sequence / codeword combinations is reached : expand the most likely leaf by changing it into a node and adding two leaves to it. The branches to the leaves will be labeled with 0 or 1. The probability in the new leaves is the probability in the node times the probability of the symbol along the branch.

Once it finishes, the algorithm has constructed a complete tree, with M leaves. To every leaf (and thus to every source sequence) the algorithm assigns a codeword of length $\lceil \log_2 M \rceil$ bit. These source sequence / codeword combinations results in the optimal variable-to-fixed length code.

In general, the best variable-to-fixed length codes cannot always achieve the same rate as the best fixed-to-variable length codes under the same conditions. But the reverse is also true. Variable-to-fixed length codes are in general better suited for memoryless low entropy sources, while the fixed-to-variable length codes are better suited for high entropy sources.

2.2.1.3 Arithmetic Code

Shannon's noiseless coding theorem [SHA48] only proves that it is impossible to encode information less than its entropy on average. In this section, arithmetic coding is introduced and it is possible to code information arbitrarily close to its entropy.

Arithmetic code is used in the coding part of many modern data compressors. It is superior to Huffman code in compression result. Arithmetic code will always represent information at least as compactly as Huffman code. Huffman coding can only output an integral number of bits per input symbol (and at least one bit is output for each input symbol) [HUF52]. Arithmetic coding dispenses with this limitation by allowing fractions of bits to be output per symbol [WNC87, MNW95].

A brief explanation of arithmetic coding is given here, followed by an illustration of its operation with a static model.

In arithmetic coding, each message is uniquely represented by a subinterval of the half closed interval [0,1). Although the messages can be drawn from alphabets of arbitrary symbols our examples will use everyday alphabets like the Roman alphabet, ASCII, or binary. Longer messages result in smaller intervals and it takes more bits to represent a small interval than a larger one. The interval used for a particular message is determined by the content of the message, and the probability of the message according to the model. The more likely messages result in larger subintervals and therefore shorter code lengths. The output of the arithmetic coder is a sequence of bits representing the interval which contains the message.

Except for short messages, it is computationally infeasible to make a direct transformation from a message to a subinterval, especially when the coder is used on-line. Most models encode the message one symbol at a time; that is, encode the message incrementally. Thus "transmit message m", or "encode m", etc. should be understood to mean "encode message m one symbol at a time".

At each stage of the encoding (or decoding) there is a current interval $[l, h)$ which consists of a lower bound l and upper bound h. To avoid wasting codespace the current interval is initialized to [0,1).

Each symbol encoded causes the current interval to be narrowed in accordance with the symbol's probability. More probable symbols cause less narrowing than unlikely symbols. Knowing with complete certainty that a symbol will occur means no narrowing occurs (because it is unnecessary to transmit that symbol), whereas if a symbol with zero probability occurs, the coding interval width is narrowed to zero and thus takes an infinite number of bits to specify.

Assume we have an alphabet of n symbols $\{s_0, s_1, \cdots, s_{n-1}\}$. Let θ define a total ordering of the alphabet, in particular we could have $\theta(s_i) = i$ for $0 \le i < n$. We ignore the distinction and write " symbol i " to mean the i-th symbol in the ordering θ. Let p_i denote the probability of symbol i, and let P_i denote the cumulative probability of all the symbols up to but not including symbol i (the sum of the probability of all those symbols for which $\theta(s_j) < \theta(s_i)$);

$$P_i = \sum_{j=0}^{i-1} p_j .$$

Let $\delta = h - l$ be the width of the current interval $[l, h)$. If symbol s_i is encoded the new interval $[l', h')$ will be given by

$$\delta' = p_{\theta(s_i)}\delta ,$$

$$l' = l + p_{\theta(s_i)}\delta ,$$

$$h' = h - \left(1 - p_{\theta(s_i)+1}\right)\delta .$$

It can easily verify that $\delta' = h' - l'$.

To illustrate the operation of arithmetic coding we now present an example using a static model. Table 2.-1 is a static-model based on an alphabet set, $\{A{:}15,\ B{:}7,\ C{:}6,\ D{:}6,\ E{:}5\}$.

symbol	$\theta(s_i)$	$p_{\theta(s_i)}$	$P_{\theta(s_i)}$	coding interval
A	4	0.384615	0.615385	[0.615385, 1.000000)
B	3	0.179487	0.435897	[0.435897, 0.615385)
C	2	0.153846	0.282051	[0.282051, 0.435897)
D	1	0.153846	0.128205	[0.128205, 0.282051)
E	0	0.128205	0.000000	[0.000000, 0.128205)

Table 2.-1 A simple static model of Arithmetic Coding

Consider coding the word "BED". The initial range is [0,1). After seeing the first symbol, "B", the interval is narrowed to [0.435897, 0.615385) which is the range allocated to this symbol. The next symbol to be encoded is "E", the current range of the interval is δ=0.179487 (0.615385-0.435897). Using the recurrence relations we discover l'=0.435897 (0.435897+0.000000δ) and h'=0.615385-(1-0.128205)δ=0.45890913. The complete coding is shown in Table 2.-2.

symbol	interval after coding
B	[0.43589700, 0.61538500)
E	[0.43589700, 0.45890913)
D	[0.43884727, 0.44238759)

Table 2.-2 Illustration of the coding process

It is not necessary for the decoder to known both ends of the interval. Any value in the final interval suffices. In above example, 0.44 would be sufficed. The entropy of "BED" in this model is $-\log p_B - \log p_E - \log p_D \approx 1.43$ (logarithm to base 10). This is why it takes 2 ($2 = \lceil 1.43 \rceil$) decimal digits (44) to encode the message. In binary, 6 bits would be required which is a significant saving over the 24 bits used by ASCII.

Now consider the decoding process. Assume 0.44 was transmitted. We can tell immediately that the first symbol was "B" because 0.44 lies within the code space allocated to symbol "B". The decoder now simulates the action of the encoder by narrowing the range to [0.43589700, 0.61538500). Proceeding like encoding, the decoder can identify the whole message.

In the example, no mention was made of how the decoder knows when to stop. In practice either the message length must be transmitted prior to the actual message or a special end-of-file symbol must be added to the input alphabet. The latter option tends to be nicer because using an end-of-file character does not require the encoder to know the length of the message in advance.

It is possible to use various statistical-code techniques to encode any file using close-to-optimal code length, given a particular model of the data. It is desirable to use a good model of the

source of the data so that the statistical-code can divided into coding and modeling which used by Rissance and Langdon [RL81].

In data compression this is referred to as the model-coder paradigm [BCW90]. The idea is that a compressor should consist of two parts : a model which gathers statistics, uses prior information and heuristics to select a probability distribution; and a coder used to produce a compact representation of events generated by the model.

2.3 Modern Techniques for Compression

There are many possible models for generating the probabilities. Modern compression systems tend to use adaptive models (models in which the probability of a symbol can change during the encoding of a message). Adaptation makes a compressor suitable for a much wider class of inputs, even for messages in which the statistics vary over the course of the message. In contrast, a static model uses a fixed set of probabilities throughout the encoding of a message. The static model is useful when random access to the compressed information is needed.

In following sections I give a brief overview of three different kinds of modeling schemes, statistical modeling, dictionary-based compression and block sorting. In fact, all these three are context-driven algorithms, but processing with context in different approaches. Statistical compressors develop models of the statistics of the input text and use those statistical models to control the final compression. Dictionary compressors build explicit or implicit dictionaries of strings and replace entire strings or groups of symbols. Block Sorting gather the context with sorting, and then reorder the input sequence to high-skewed distribution. With collection of contexts, statistical compressors build an explicit probability model but dictionary-based or block sorting compressors build an implicit probability model.

2.3.1 Statistical Modeling

Statistical text compressors are traditionally regarded as a combination of a modeling stage and a following coding stage. The model is constructed from the already-known input and used to facilitate efficient compression within the coder (and matching decompression in the decoder). A good model will contain a few symbols with high probability (and preferably one dominant symbol), thus allowing very compact coding of those probable symbols.

2.3.1.1 Context Modeling

The idea of context modeling is based on the fact that distribution of possibilities in the alphabet depends on the nearest context, in other words, letters are more likely to appear in a particular pattern, that is, next to or near other particular letters. For this technology [CW84, BCW90], there also exist principal restrictions for the increase of a sliding frame, or a moving processor of data or letters, for which a context model is built. A small sliding frame with a corresponding merge context model is optimal on variable data, for which the problem of zero frequency is especially acute. In the course of increasing the size of a sliding frame, more and more various information is processed (e.g., a text in French, then a text in German and then a text in Russian enters this frame), with the result that the distribution of possibilities widens and the effectiveness of context modeling (with short context) decreases quickly. With the increase of the context length, costs also increase exponentially. The transition to context-mixed models has improved the situation to some extent; however, the absence of theoretically substantiated schemes of intermingling probabilities is compensated by a large number of heuristics, which takes us back to the times of alchemy.

In the prediction by partial matching (PPM) compression [MOF90], each symbol is predicted from its preceding contexts and then used to extend existing contexts and develop new ones. The multiple contexts develop in parallel and the relevant contexts usually change completely as each symbol is processed. For any particular input symbol the encoder and decoder can develop an appropriate collection of contexts of various coders, predict the probabilities of each possible symbol, and code according to these probabilities.

The accurate prediction of probabilities and especially of the probability of escaping into lower order contexts is the crux of successful PPM compression. Improvements generally follow from improved prediction of these probabilities. Most importantly, with PPM processing the input sequentially we always have precise knowledge of all possible contexts.

The order of a model is the maximum number of symbols used to predict the next symbol. In practice an order-*n* model will sometimes base its prediction on less than *n* symbols. By convention the order-(-1) model predicts each symbol with equal probability and the order-0 model predicts each symbol with probability proportional to the number of times it has occurred previously.

The general PPM method requires the predictions of all orders (up to some maximum) to be blended together to give an overall probability for the next symbol. The most general approach is

$$\Pr\left(s_j = \phi\right) = \sum_{i=-1}^{n} w_i p_i\left(\phi\right),$$

where the w_i are a set of weights normalized to sum to 1, s_j is the next symbol, and $p_i(\phi)$ is the probability of symbol ϕ according to an order-*i* model. Calculating the sum is computationally expensive and there is no single "right" way to determine the weights to use. The probabilities $p_i(\phi)$ are rational and based on the frequency counts :

$$p_i(\phi) = \frac{f_i(\phi)}{F_i} \text{ where } F_i = \sum_{\phi} f_i(\phi) .$$

However, the formula for $p_i(\phi)$ just given leads to the zero-frequency problem, citeWB, since symbols not previously encountered are given zero probability [WB91]. The formula for $p_i(\phi)$ must be modified so that symbols not previously encountered in a given context can be represented. If the context itself has never been seen before then all of the $f_i(\phi)$ and F_i will be zero. However, we are guaranteed that some shorter context has been seen before; in the worst case the order-(-1) model can be used. The order-(-1) model always predicts every symbol.

In practice full blending is not used; instead each context assigns a probability, called an escape probability, to a novel symbol occurring. The PPM variants differ in the way escape probabilities are assigned. When a novel symbol is seen the escape probability is used followed by the prediction of the next shorter context. Several escapes may be made before a context is reached which predicts the symbol. In the worst case the order-(-1) model makes the prediction. In other hands, escape probabilities are equivalent to weighting. We now mention how the probabilities are determined in few popular variants.

- **PPMC** : In this method no prediction is made unless the symbol has occurred more than once in the current context. This is done by subtracting one from all counts with the subtracted counts being combined to give the escape probability. The idea tries to filter anomalous event. Symbols are not predicted until they are seen twice. Thus, $e_i = \frac{q_i}{F_i + q_i}$ where q_i is the number of different symbols seen in the given context. The probabilities for the remaining symbols become $p_i(\phi) = \frac{f_i(\phi)}{F_i + q_i}$.

- **PPMD** : This method is a small modification to PPMC where each count is incremented by $\frac{1}{2}$. It sets $e_i = \frac{q}{2F_i}$ and $p_i(\phi) = \frac{f_i(\phi)+\frac{1}{2}}{F_i+1}$. Besides, using of deterministic scaling also leads to better performance.

The PPM method can be bounded if some maximum order is specified in advance or unbounded [TEA95] if contexts are allowed to be arbitrarily long. A tree representation is suitable for both approaches. Each node in the tree contains a frequency count $f_i(\phi)$ for the given context. To prevent overflow it is on occasion necessary to re-scale the frequency counts. Scaling often actually leads to improved compression as it gives increased importance to recently encoded text. Scaling is

appropriate for most text where the subject matter varies slowly over the course of the document.

Recent experiments have found that PPMC is better suited to the unbounded approach, whereas the PPMD is more effective with the bounded approach [CTW95]. Two new variances of PPM, PPM* of C method and PPMD+ of D method, are developed based on the experiments.

2.3.1.2 State Based Modeling

State-based modeling is another large class of statistical compression techniques. In principle this approach is more powerful than the context modeling used by PPM or dictionary coding as the general finite-state machine is able to capture some regularities (particularly those involving counting) that context models cannot.

One adaptive approach is dynamic Markov coding (DMC) which starts with a small initial model and expands by the addition of new states as they are required [BM89]. Probabilistic models with a finite number of states can be described by a finite automate. A set of states $S(i)$ and a set of probabilities of transition $\Pr(i, j)$ from the state i into the state j are called Markov's models. Frequency counts are maintained for each transition and new states are cloned when a transition becomes sufficiently popular (as governed by a heuristic).

Dynamic Markov Coding (DMC) is of practical interest, in that it works adaptively, starting from a simple initial model and adding, if necessary, new states. PPM technology is a particular case of the DMC approach. DMC allows the construction of context models not only for single symbols (as with PPM and consideration of letters of the alphabet), but also for phrases or lines.

2.3.2 Dictionary Based Compression

Statistical modeling is not the only popular approach to adaptive data compression. Another kind of compression uses a specially constructed dictionary to achieve compression. Comparison between two kinds of methods, statistical methods usually achieve better compression, dictionary methods use memory and processor time more efficiently.

In a dictionary based compression scheme, groups of consecutive symbols, called phrases, are replaced with indices into some dictionary. The dictionary is constructed so as to contain a list of phrases expected to occur frequently. To achieve compression the indices must occupy less space than the phrase they encode. Dictionary coding is also called macro coding and codebook coding. Since the number of bits used in indices can be chosen to align with machine

words efficient implementations are possible. Typically a multiple of four or eight bits is chosen for the indices. Since in some schemes the number of phrases can grow without bound it is necessary to be able to encode arbitrarily large integers. Methods for doing this are discussed in [ELI75].

These systems achieve good compression because a single dictionary reference may represent many characters. For every dictionary scheme there is an equivalent statistical scheme achieving the same compression [BCW90]. Eventually, dictionary methods will probably be completely replaced by statistical approaches. Dictionary schemes are currently still widely because they offer rapid decompression.

The construction of the dictionary is one of the more important aspects of the system. A dictionary that closely matches the text to be compressed will yield good compression. The dictionary can be static, semi-adaptive, or adaptive. The maximum length of phrases stored in the dictionary may be fixed or unbounded. Better compression is achieved by having an adaptive dictionary which allows longer phrases.

DEFINITION 2.1 : A dictionary $D = (M, C)$ is a finite set of phrases M and a function C that maps M onto a set of codes, where $M \subseteq A^*$ for an alphabet A. Without loss of generality the output codes are assumed to be over $\{0,1\}^*$.

DEFINITION 2.2 : The set M is complete if every infinite string over the input alphabet A is also in M^*; that is, any input string can be formed by the concatenation of phrases from M.

DEFINITION 2.3 : The function C obeys the prefix property if no string $C(m)$ is a prefix of another string $C(s)$, for $s, m \in M$ and $s \neq m$.

For reversible compression of any input to be possible the set M must be complete and C must obey the prefix property.

Aside from constructing the dictionary there is also the problem of matching the input to the dictionary. Optimal parsing is hard: it sometimes requires the entire input to be examined before anything can be output. There are reasonable heuristics which approach optimal parsing; frequently, greedy parsing is used. In greedy parsing the next longest match is found. The extra compression gain obtained by using optimal parsing is minimal and is at the expense of increased execution time.

Static dictionaries are constructed by considering a sample of representative text prior to actually compressing any messages. Such a static dictionary is assumed to be available prior to transmission to both the encoder and decoder. A semi-adaptive dictionary is constructed for the text to be compressed in an initial pass. Determining the optimal dictionary is NP-complete in the size of

the text. Further, the dictionary itself must be transmitted along with the compressed text. Most modern dictionary systems use adaptive dictionaries which are constructed incrementally (by both the encoder and decoder) as the message is transmitted.

The most popular dictionary schemes are from the Lempel-Ziv family of compressors. In the LZ77 coding scheme, text is compressed by providing references to earlier text. In fact, there are many variants of the LZ77 coding scheme. The coding scheme presented in LZ77 differs in several respects to the later scheme LZ78. In this section a brief overview of the different forms of Lempel-Ziv coding are given.

2.3.2.1 LZ-compression

Lempel-Ziv has two basic modifications, LZ77 and LZ78 along with a large number of variations. At present LZ models are widely used. The theoretical approach for LZ models was suggested in 1977 by Lempel A. and Ziv J., with software realization in the early eighties.

LZ-compression substitutes the initial text with references to a dictionary; it seems that this scheme resembles the methodology used by two interlocutors.

However, with the growth of the dictionary, the number of bits necessary for formulating the reference grows proportionally by a binary logarithm to the size of the dictionary. The length of the phrases in the dictionary (in the simplest case, a binary tree) at the beginning considerably surpasses the growth of the length of the references, but after some saturation, the speed of their growth asymptotically tends to logarithmic dependence. The optimal size of the dictionary varies for different types of data; the more variable are the data, the smaller the optimal size of the dictionary.

- **LZ77** : The first form of Lempel-Ziv coding [ZL77]. Pointers are used to denote phrases in a fixed-size window that precedes the coding position. Thus the window is essentially the phrases of the dictionary. There is a maximum length for sub-strings that may be replaced by a pointer (usually about twenty). The window is initially spaces. Matches may overlap with the text. The longest match is coded as a triple $(N,K,'A')$ where N indicates where in the just-encoded window to start copying, K is the length of the string to copy, and 'A' is the next symbol after copying. The window size is typically about eight kilobytes.

- **LZSS** : The same as LZ77 except pointer and characters are distinguished by a flag bit. Each codeword consists of $(f,N,K,'A')$ or $(f,'A')$. The one-bit flag $f = 0$ is indicating "copy", $f = 1$ is indicating "no copy". This avoids the presence of an explicit character in each triple.

- **LZ78** : The second form of Lempel-Ziv coding [ZL78]. The input text is broken into phrases where each phrase is the longest matching phrase seen previously plus one character. Each phrase is encoded as an index to its prefix plus the extra character. The number of phrases can grow unboundedly. In practical implementations when memory is exhausted the coding starts again from scratch. The LZ78 scheme is asymptotically optimal for a stationary ergodic source although convergence is relatively slow.

- **LZW** : Like LZ78 but the output consists solely of pointers [WEL84]. This is achieved by initializing the dictionary to contain every character in the alphabet. The LZW approach is perhaps the most common LZ variant in practice.

2.3.3 Other Compression Techniques

2.3.3.1 Block Sorting

A new class of compression technique uses sorting as the context collector. The algorithm works by applying a reversible transformation to a block of input text. The transformation does not compress the data, but reorders the input text and makes it easy to compress with simple algorithms such as move-to-front coding plus simple statistical coding.

This algorithm achieves speed comparable to algorithms based on the techniques of Lempel and Ziv, but obtains compression close to the best statistical modeling techniques. The size of the input block must be large (a few kilobytes) to achieve good compression.

More detailed reviews on such type algorithm (Block Sorting [BW94] and Context Sorting [YOK96]) will be presented in Chapter 3.1 and Chapter 4.2.

Why the transformed string compress well ?

Using the Block Sorting or Context Sorting as a front end for data compression has a result similar to that of simple statistical modeling programs. An order-n statistical model simply uses the previous n characters to predict the value of the next character in a string. Compression programs based on statistical modeling can give good results, at the expense of high memory requirements.

Because of the sorting method used, characters hundreds of bytes downstream can have an effect on the ordering of the sorting output. This characteristic of the sorting has another side

effect. It means that in general, the bigger the block size, the better the compression. As long as the data set is homogenous, bigger blocks will generate longer runs of repeats, leading to improved compression. Of course, the sorting and searching operations needed at the front end of the Block Sorting and Context Sorting will usually have $O(N \times \log(N))$ performance, meaning bigger blocks might slow things down considerably.

2.3.3.2 Context Tree Weighting

Yet another approach, Context Tree Weighting algorithm [WST95, WST96, VOL96], is an another important approach of data compression. In the recent results, Context Tree Weighting achieves some best compression results on the files of Calgary Corpus [WB].

Context Tree Weighting algorithm is a sequential universal data compression procedure for binary tree sources that performs the "double mixture". Using a context tree, this method weights in an efficient recursive way the coding distributions corresponding to all bounded memory tree sources, and achieves a desirable coding distribution for tree sources with an unknown model and unknown parameters. Computational and storage complexity of the proposed procedure are both linear in the source sequence length. Because of the shortage of time, I have not investigated this algorithm. However, Context Tree Weighting is another important algorithm as the Block Sorting.

3. Symbol Remapping

In this chapter, I describe a new avenue for improving the compressibility of text. I consider remapping the symbol set so that a representation of a character reflects its properies as a predictor of future text. In this chapter, I use this idea in conjunction symbol remapping method with block sorting algorithm. Then, symbol remapping method will be operating as a preprocessor of block sorting algorithm. This enables us to use an estimator from a restructed class to map contexts to predictions of upcoming characters.

3.1 Reviews on Block Sorting

Michael Burrows and David Wheeler released the details of a transformation function (Block Sorting [BW94])* on the text data set in 1994 that opens the door to some revolutionary new data compression techniques. The algorithm was improved in the work by Peter Fenwick on the locally adaptive entropy coding for output sequence [FEN95A, FEN95B, FEN95C, FEN96], Ziya Arnavut and Spyros S. Magliveras also extended the algorithm for optimization choices on the permutation [AM97].

The Block Sorting Algorithm has received considerable attention. Since it achieves as good compression rates as context-based methods, such as PPM, but at speeds comparable to those of algorithms based on Lempel-Ziv techniques.

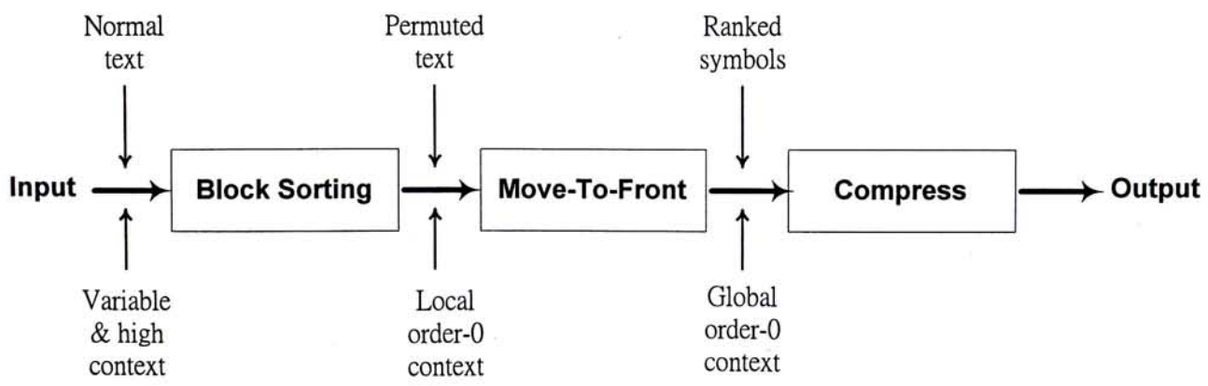


Figure 3.-1 Data flow in Block Sorting Compression

* also called as "block reduction" or "Burrows-Wheeler Transform".

The method is really a composite of three different algorithms: the Block Sorting main engine (a lossless, very slightly expansive preprocessor), the locally adaptive non-compressive coder (move-to-front coding method is used in original Block Sorting illustration) and a simple statistical compressor (first order Huffman is mentioned as a candidate) eventually doing the compression. Among three methods only the first one is discussed here as it is what constitutes the heart of the algorithm. The Block Sorting is based on a reversible sorting operation that brings together symbols standing in the same or a similar context. Since such symbols do correlate often this correlation can be utilized by simple coding algorithms. Intuitively speaking, the method transforms slack in the higher order probabilities of the input block (thus making them more even, whitening them) to slack in the lower order statistics.

3.1.1 Forward Transformation

Briefly, the algorithm transforms a string *S* of *N* characters by forming the *N* rotations (cyclic shifts) of *S*, sorting them lexicographically, and extracting the last character of each of the rotations. A string *L* is formed from these characters, where the *j*-th character of *L* is the last character of the *j*-th sorted rotation. In addition to *L*, the algorithm computes the index *i* of the original string *S* in the sorted list of rotations. Surprisingly, there is an efficient algorithm to compute the original string *S* given only *L* and *i*. The sorting operation brings together rotations with the same initial characters. Since the initial characters of the rotations are adjacent to the final characters, consecutive characters in *L* are adjacent to similar strings in *S*. If the context of a character is a good predictor for the character, *L* will be easy to compress with a simple locally-adaptive compression algorithm.

The forward transformation can be described as —

1. Sort the input symbols, using as a key for each symbol, to whatever length is needed to resolve the comparison. Key is symbols which immediately follow the symbol-to-be-sorted. The symbols are therefore sorted according to their following contexts.

2. Take as output the sorted symbols, together with the position in that output of the last symbol of the input data.

To illustrate the operations of block sorting algorithms we consider the *S* = [good, very good] be a given permutation as shown in Figure 3.-2. Constructing the matrix of consecutive cyclic left-shifts of the sequence *S* to form successive rows of *M*. By sorting the rows of *M* lexically, we transform it to *M'*.

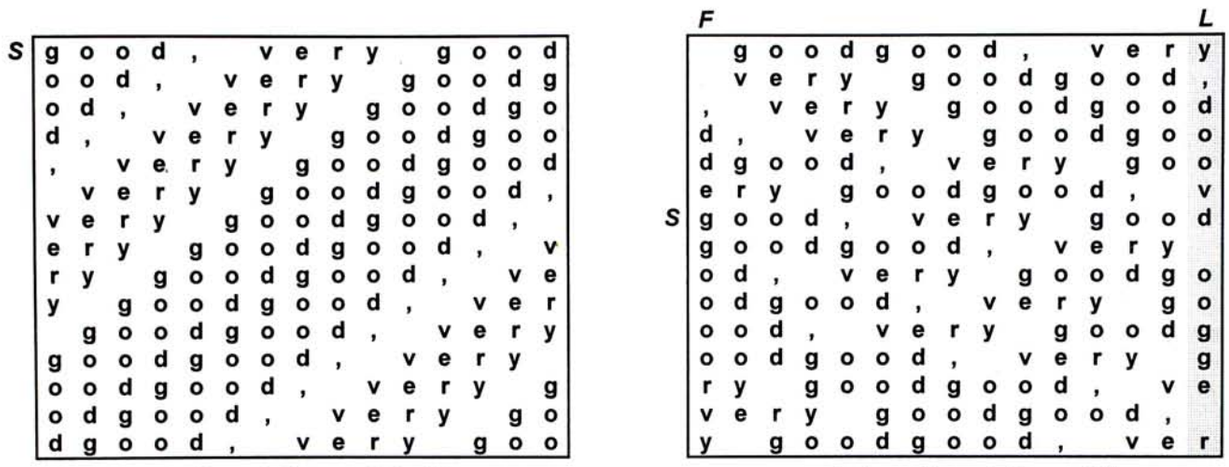


Figure 3.-2 The Forward Transformation of Block Sorting

The original sequence S appears in the $i = 7^{th}$ row of M'. Let F be the first, L be the last column vector of M'. For reconstruction the original sequence S uniquely, we should transmit the pair { 7, [y,doovd oogge r] } to decoder.

3.1.2 Inverse Transformation

Using the output (i,L) of the Forward Transformation to reconstruct the original input, the string S of length N, we requires the rules that defines the order in which the rotated strings are scattered throughout the rows of the matrix M'. As F in M' has an important characteristic: it contains all of the characters from the input string in sorted order. Since L also contains all the same characters, we can determine the contents of F by simply sorting L. Linking the correspond symbol of F and L and breaking ties by recency. Then receiver can construct the original sequence S by using the following procedure :

$$S[n] = L[i],$$

For $j = 1,\ldots,n-1$ **, let** $S[n-j] = L[S[n-j+1]]$.

A inverse transformation of the example is shown as following. Applying the linkage between L and F, and primary index i, into the above reconstruction procedure, the receiver can rebuild the original sequence.

	L	F		L	F													
	y			15	1													
	,			3	2													
	d	,		4	3													
	o	d		9	4													
	o	d		10	5													
	v	e		14	6													
S	d	g		5	7	7	11	9	4	3	2	14	6	13	15	1	8	12	10	5
		g		1	8	g	o	o	d	,		v	e	r	y		g	o	o	d
	o	o		11	9													
	o	o		12	10													
	g	o		7	11													
	g	o		8	12													
	e	r		6	13													
		v		2	14													
	r	y		13	15													

Figure 3.-3 The Inverse Transformation of Block Sorting

3.2 Ordering Method

In this section, I describe a new avenue for improving the compressibility of text. The main idea is that changing the representation of symbols may prove beneficial for compression.

For the Block Sorting algorithm, lexical sorting is used due to the convenience for comparison. Alternatively ordering on the symbols is attempted as following. The first ordering method I have used is the empirical frequency-ordering. Therefore, $a \prec b \prec c \prec \cdots$ is instead by $e \prec t \prec o \prec a \prec i \prec \cdots$.

The Block Sorting algorithm works in three stages : first it try to reorder the symbols in a block (a list) based on a reversible sorting, and then proceed the ranked list with locally adaptive non-compressive coder, finally it uses a simple statistical compressor doing the compression. Symbol ordering (or mapping) method is focused on the second step. Analysis the Move-To-Front coding (in Chapter 2.2), we can find that the main factor to the compression performance is cost for the symbol changing. For each symbol changing, the cost is relied on the position of the new symbol in the MTF list.

Current methods do not consider geometric information for prediction purposes. For instance, the letter *e* tends to predict the letter "*space*", *r*, *n* and *d*. However, in the ASCII Table, the distance among those character are very difference. Be a pre-processor, empirical frequency-ordering method reorders all the symbols in the block based on their frequency. The basic idea behind it is to keep the cost of the ranks changing in the MTF list minimize. As the highest frequency

symbol has been assigned with smallest value, the cost for the ranks changing in the Move-To-Front Coding part can be smaller.

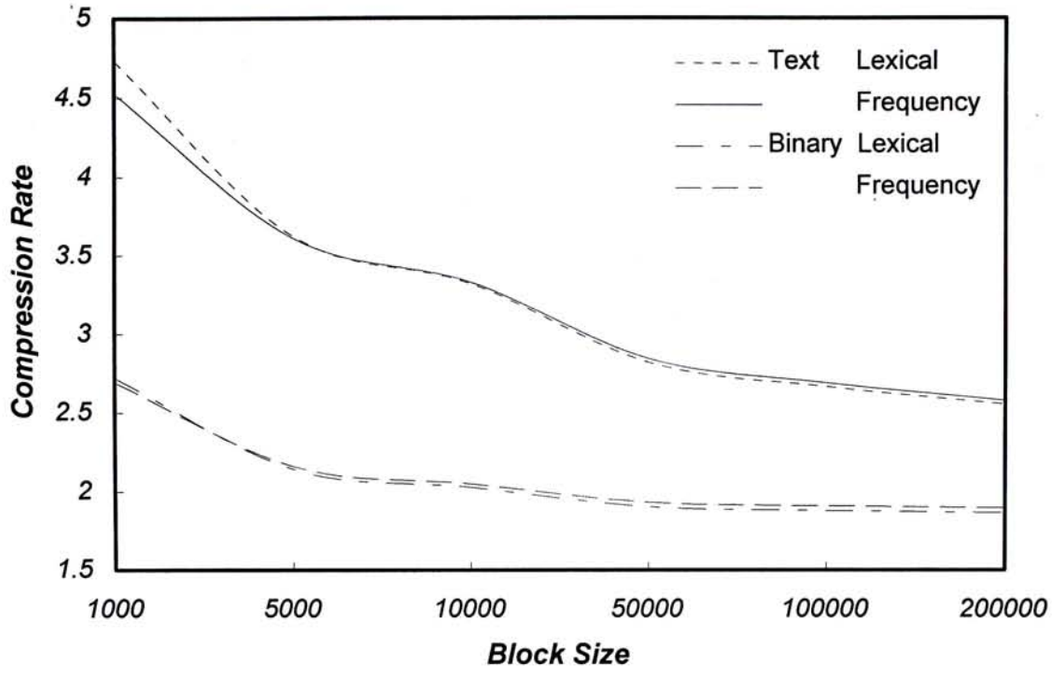


Figure 3.-4 Empirical Frequency Ordered Block Sorting

Checking the above Figure 3.-4 or Table 3.-1 in Appendix A, the Block Sorting Lossless Compression Algorithm can be improved by a Empirical Frequency Ordering preprocessor. With the simple preprocessor, the gain in the compression rate is particularly significant when the block size is small. When the block size is large, this advantage vanishes. Our result is useful because its advantage can be combined with the lower complexity of smaller block sizes. One thing needs to be focused is that the improvement on compression rate* owing to the empirical frequency ordering becomes worse with increasing the block size.

As empirical frequency ordering filter is helping in the symbol ranking compression, a further development can be centered on the high order context frequency ordering, or a fixed, pre-set ordering type for saving the header.

There are two methods can adapt the fixed ordering type. Either a grouping method collects the symbols with the similar properties together, or re-order the symbols with appearance frequency of the natural language. Both methods give almost same phenomenon on the files (Figure 3.-5) as empirical frequency ordering method mentioned before. In Figure 3.-5, a relative compression rate

* compression rate = bits per input symbol

based on lexical ordering is used. Grouping method has a better performance than the Language Style method (pre-set frequency ordering) in small block size.

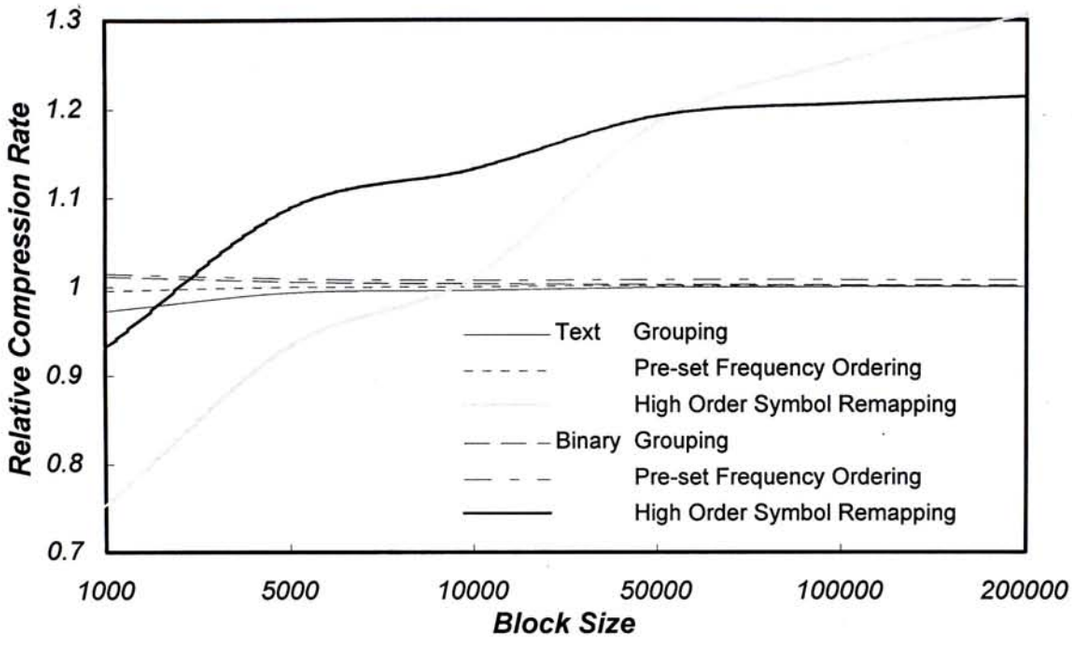


Figure 3.-5 Alternatively Frequency Ordering in Block Sorting Compression

As the baseline Content Prediction method (see Chapter 4.2) do the similar job as the high order modeling, we can use the baseline Content Prediction as the pre-processor of the Block Sorting. Reference to the results on the empirical frequency ordering method, we know the improvement only appears in the small block size, but worse case appears in the large block size. A high order frequency ordering enlarges such phenomenon. On the other hand, such phenomenon can be a good choice for helping the block sorting algorithm to compete with other compression methods under the limited resource environment.

3.3 Discussion

In the above sections, I have reviewed the Block Sorting lossless data compression algorithm. The basic idea of the system is to do ranking (reversible sorting) on a list and then operate with a simple statistical compression algorithm. The Block Sorting Algorithm has received considerable attention. Since it achieves as good compression rates as context-based methods, such as PPM, but at speeds comparable to those of algorithms based on Lempel-Ziv techniques.

Further improvements on the Block Sorting algorithm, some alphabet remapping methods are suggested for reducing the cost of symbol changing in MTF. With a simple preprocessor, Block Sorting can perform better at the limited memory situation.

4. Content Prediction

In this chapter, I present a general-purpose text compression algorithm. It is a codebook based algorithm where the codebook is derived from a combination of sliding windowed context. By suitable adjustment of design options, this algorithm includes both the LZ77 algorithm and the context coding algorithm by Yokoo as extreme special cases. The codebook is derived from a fixed-length window of the just-encoded text at any given time. The symbol content predictor and symbol ranking scheme are the main structure in this algorithm. Thus, I will introduce the concept about the prediction and ranking schemes at section 4.1. Then a reviews on the Yokoo's Context Sorting algorithm will be given in section 4.2. Finally, I will acquaint my proposed Content Prediction method.

4.1 Prediction and Ranking Schemes

A simple concept about the prediction and ranking schemes is explained at this section. Prediction and ranking is a finite context modeling method. It is based on the observation that a symbol in a sequence is often determined by the symbols that immediately precede it. For example, in English text, the letter *q* determines that the next symbol is most likely to be *u*. In a less extreme example, the letters *th* usually precede the letter *e*, but may also precede *a*, *i*, *o*, *u*, or *r*. In fact, the most successful predictors used for data compression schemes are based on this principle.

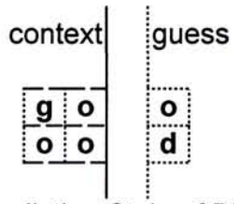
The prediction and ranking scheme can be divided into two parts, prediction and ranking. The idea is learning to do the prediction. At first, a symbol predictor determines a probable next symbol, which may be accepted or rejected by a compactor which can see the incoming text. The correct answer is either guessing by the predictor or providing by compactor after several unsuccessful attempts which is controlled by ranking scheme.

4.1.1 Content Predictor

For the prediction, the modeling system usually predicts future events based on past events in the same context. PUSS is a prediction system that was developed by Andreae in 1977. PUSS stands for Prediction Using Slide and Strings, the 'slide' referring to the window of context preceding the current prediction, and 'strings' to the table of previous contexts and subsequent symbols. PUSS predicts future events based on past events in same context [AND77].

The form of this system is very similar as LZ-77 compression method. Instead of finding the repeated phrases, it uses the context for prediction of coming symbols. The text is processed sequentially; as learning proceeds the predictor builds a sorted dictionary of all encountered context. Each context is followed in the input text by its corresponding symbol. Prefixes of the to-be-processed symbol are tested against sorted dictionary to find matching phrases.

Assume we predict the coming symbols after we read the string "good, very go". Use the method describes above, we can find the same pattern as the current context. Matching more in the context, more correctness of prediction.



Prediction Style of PUSS

4.1.2 Ranking Technique

Symbol ranking method is essentially a transformation of the input symbols, with the transformation usually dynamic according to the symbol context. There is an output symbol for each input symbol, and compression relies on the very skew output distribution, with most output symbols being 0 and able to be emitted with very short codes.

A simple example of symbol ranking is Move-to-Front coding. MTF coding assumes that symbols can be ranked according to the closeness of their last occurrence (see Chapter 2.2). The effect is that more frequently-used symbols stay closer to the front of the list, while less frequently-used symbols drift to the back of the list. Smaller list ranks tend to be more frequent and can be emitted with shorter codes, while the less frequent larger ranks require longer codes.

In general the transformation produces an output symbol for each input symbol, with the output alphabet being dominated by a few symbols. It is the responsibility of the following compressor to handle the transformed data as efficiently as possible. Block Sorting algorithm (in Chapter 3.1) is one of the best of such transformation nowadays.

As the result of prediction is an ordered list of symbols, from the most probable to least probable. Each input symbol is re-coded into its rank in the list and that rank is output as the code for the symbol. Input string is just re-coding at prediction stage, and no compression is done. Then ranking scheme control the output of predictor that making the re-coded alphabet has a highly skewed frequency distribution, implying good compressibility.

4.2 Reviews on Context Sorting

Context Sorting [YOK96] is a method more related to the Block Sorting algorithm by Burrows and Wheeler. The only difference between two algorithms is the style of sorting. Block Sorting does not process its input sequentially, but instead processes a block of text as a single unit. Context Sorting does this by adjusting sorting on the fly, based on prediction and sliding. Context Sorting preprocessor can be separated into two different parts. One is a prediction operation and another is a ranking method. The concept about the prediction and ranking schemes can be found in section 4.1. Actually, Context Sorting can be counted as a application of PUSS on compression.

4.2.1 Context Sorting basis

The basis structure of Context Sorting is similar as the Lempel-Ziv '77. Both algorithms are using context as a content predictor. However, LZ77 uses context for the longest matching on the content and Context Sorting uses context as a hint to guess an inputting symbol. The goal of Context Sorting is to sort a set of contexts in ascending order. Yokoo's context coding method is given by :

1. sorts the so far observed contexts in lexical ordering
2. finds the best matching context, using this context index as an anchor into the context dictionary
3. searches neighboring (similar) contexts of anchor for matching to-be-processed symbol
4. If to-be-processed symbol is found and occurs a distance *d* contexts from the anchor, the just inputting symbol is represented by *d*
5. compresses the ranks with statistical coding

In spite of the simplicity of Yokoo's Context Sorting mechanism, the performance of algorithm is worse than GZIP for the same situation (memory consumed). In the following sections, some more extensions are suggested for improving the Context Sorting mechanism.

4.3 General Framework of Content Prediction

A general framework of extended context sorting mechanism [WWY97] is introduced here. To distinguished from the Yokoo's context sorting algorithm, this extended context sorting algorithm

is named as Content Prediction. With using sorted context codebook, the Content Prediction system predicts more symbols in content rather than one symbol prediction in Yokoo's coding method.

At time t, let the just-encoded window (JEW) is consisting of the source symbols $x_{t-\ell}, x_{t-\ell+1}, \cdots, x_{t-1}$. The remaining source symbols to be encoded (TBE) are $x_t, x_{t+1}, x_{t+2}, \cdots$. At this time, the compressor outputs the token is (Go back to the n-th longest context match, copy k symbols) or, equivalently, (n,k).

Let me explain the meanings of token entries. The context of the TBE is the suffix of the JEW. The suffices of the JEW below a certain length have appeared in the JEW before. We rank these prior appearances according to the length of the suffix, and breaking ties according to recency. Assume the n-th context is

$$\left(x_{t-\ell}, x_{t-\ell+1}, \cdots, x_{t-\ell+s-1}\right) = \left(x_{t-s}, x_{t-s+1}, \cdots, x_{t-1}\right) \qquad (4.\text{-}1).$$

Furthermore, assume that

$$\left(x_{t-\ell+s}, x_{t-\ell+s+1}, \cdots, x_{t-\ell+s+k-1}\right) = \left(x_t, x_{t+1}, \cdots, x_{t+k-1}\right) \qquad (4.\text{-}2).$$

Then the token (n,k) means go back to the n-th context, which (4.-1), and copy the subsequent k symbols, which are (4.-2). For practical implementation, I restrict $k \le K$ and the context length to less or equal to N.

The context ranking is completely determined from the JEW. Therefore, the decoder can reconstruct the context ranking and retrieve the n-th context, then copy the subsequent k symbols.

The Context Sorting algorithm is a symbol ranking based compression algorithm. It re-encodes the input symbol s_t into an alphabet set with different frequency distribution. The skew of the frequency distribution is mainly depended on the accurate of predictor. More accurate prediction, higher skewed in frequency distribution. In the extreme case, as $N = 0$, (no context), then our algorithm reduces to Lempel-Ziv '77 (LZSS variant) with the same token streams. If we fix $N = 0$ and $k = 1$, then the algorithm reduces to Move-To-Front coding (and its equivalents).

4.3.1 A baseline Version

In the baseline version, I try to do a simplest testing by fixing K to be 1. Then Content Prediction algorithm predicts only one symbol as Yokoo's scheme at this time.

Assume $s_{t-W-M+1},\cdots,s_{t-W+1},\cdots,s_{t-1}$ are the $W+M-1$ most recently encoded source symbols, and $s_t, s_{t+1}, s_{t+2},\cdots$, are the source symbols to be encoded. For $i=1,2,\cdots,W$, let S_i be the M-symbol word

$$S_i = s_{t-i}s_{t-i-1}\cdots s_{t-i-M+1} \tag{4.-3}$$

Sort the words $S_1, S_2,\ldots, S_w$ in alphabetical order. Ties are broken by chronology. Denote the sorted result by

$$S_{\sigma(1)} \prec S_{\sigma(2)} \prec\cdots\prec S_{\sigma(w)} \tag{4.-4}$$

Note that $\sigma(*)$ is a permutation on the set of integers $\{1,2,\cdots,W\}$.

Let p be such that $\sigma(p)=1$, or equivalently $p=\sigma^{-1}(1)$. Let r be such that the symbol S_t is the r-th distinct symbol to appear in the list of symbols

$$s_{t+1-\sigma(p-1)}, s_{t+1-\sigma(p+1)}, s_{t+1-\sigma(p-2)}, s_{t+1-\sigma(p+2)}, s_{t+1-\sigma(p-3)}, s_{t+1-\sigma(p+3)},\cdots \tag{4.-5}$$

Note that these are exactly the symbols immediately "following" the words

$$S_{\sigma(p-1)}, S_{\sigma(p+1)}, S_{\sigma(p-2)}, S_{\sigma(p+2)}, S_{\sigma(p-3)}, S_{\sigma(p+3)},\cdots \tag{4.-6}$$

in the just-encoded window of source symbols. Note that the list (4.-5) is known to the decoder.

The Content Prediction algorithm re-codes the symbol s_t into a binary sequence which is the Elias coding of the integer r. In the event that the symbol s_t is not found in the List (4.-5), the encoder outputs an escape token, such as $r=W+1$. Then the window of just-encoded symbols are shifted by one symbol, and the algorithm iterates.

Instead of the simple alternate merge use in (4.-6), other principles of merging the two lists

$$S_{\sigma(p-1)}, S_{\sigma(p-2)}, S_{\sigma(p-3)},\cdots \tag{4.-7},$$

$$S_{\sigma(p+1)}, S_{\sigma(p+2)}, S_{\sigma(p+3)},\cdots \tag{4.-8}.$$

can be used. We will in fact use one such variation in the proof of theorems concerning asymptotic information-theoretic compression performance.

When s_t is not found in List (4.-5), one method is to output an escape code, say the token $r = W + 1$, followed by the symbol s_t uncompressed. A more efficient method is maintaining a move-to-front list, MTFL, of the all possible symbols. As we proceed down List (4.-5), perform move-to-front of the "new symbols" we encounter as we proceed down the List (4.-5) until we find s_t. If the s_t cannot be found in List (4.-5), then the rank of s_t in MTFL at that time is outputted by the compressor.

4.3.2 Context Length Merge

During the operation of the prediction, we get a list of probable symbol for guessing. To order the list of symbols from the most probable to least probable, our ordering scheme select the symbol due to the similarity of its context and context of to-be-processed symbol. Therefore, matching more in context, guessed symbol assumes be more probable. The probable symbol is no longer relied on the frequency as the order compression models. Then, this scheme uses less memory and processing time. Followings are the details about our scheme.

Let the function contextlen(X) return the length of the prefix match between $X = (x_1, x_2, \cdots x_M)$ and $S_1 = (s_{t-1}, s_{t-2}, \cdots, s_{t-M})$. I.e.

contextlen(X) = ℓ, such that $x_1 = s_{t-1}, x_2 = s_{t-2}, \cdots, x_\ell = s_{t-\ell}$, and $x_{\ell+1} \neq s_{t-\ell-1}$.

Note that either list (4.-7) or list (4.-8) is already sorted in the order of non-increasing values of contextlen.

Instead of the simple alternate merge in (4.-6), we use the following merge of two lists (4.-7) and (4.-8) : Merge according to contextlen, where ties are broken by preferring list (4.-7) members to list (4.-8) members. The merge result is denoted by

$$S_{\mu(1)} \lhd S_{\mu(2)} \lhd \cdots \lhd S_{\mu(W-1)} \qquad \text{(4.-9).}$$

where $S_{\mu(i)} \lhd S_{\mu(j)}$ if and only if

contextlen$\left(S_{\mu(i)}\right)$ > contextlen$\left(S_{\mu(j)}\right)$; or

contextlen$\left(S_{\mu(i)}\right)$ = contextlen$\left(S_{\mu(j)}\right)$, $S_{\mu(i)}$ in List (4.-7) and $S_{\mu(j)}$ in List (4.-8) ; or

contextlen$\left(S_{\mu(i)}\right)$ = contextlen$\left(S_{\mu(j)}\right)$, $S_{\mu(i)}$ in front of $S_{\mu(j)}$ in same List .

Note that $\mu(*)$ is a permutation on the $W-1$ integers $\{2,3,\cdots,W\}$. In each iteration, the encoder transmits integer r such that s_t is the r-th distinct symbol to appear in the list

$$s_{t+1-\mu(1)}, s_{t+1-\mu(2)}, s_{t+1-\mu(3)}, \cdots \qquad (4.\text{-}10).$$

Note that these exactly are the symbols that immediately "follow" the words of (4.-9) in the encoder's input sequence.

In the above, a sub-list from (4.-7) consisting of words of the same contextlen value is entirely in front of its counterpart sub-list from (4.-8) with equal contextlen value. A variation is to use simple alternate merge on such a pair of counterpart sub-lists.

The rules for context merge can be summary as three steps : *1.) longer context matching gets a smaller rank*, *2.) break ties by lexicography*, and *3.) break further ties by recency*. Compared with Burrows-Wheeler Transform, one more rule, "longer context matching gets a smaller rank", are included. Such addition makes the Content Prediction processing with an adaptive model. The algorithm is suitable for a much wider class of inputs.

Therefore, baseline version of Content Prediction algorithm can be describe as :

1. sorts the so far observed contexts in lexical ordering
2. finds the best matching context, using this context index as an anchor into the context dictionary
3. searches neighboring (similar) contexts of anchor for matching to-be-processed symbol
4. context length merge
    a. *longer context matching gets a smaller rank*,
    b. *break ties by lexicography*,
    c. *break further ties by recency*.
5. If to-be-processed symbol is found and occurs a distance *d* contexts from the anchor, the just inputting symbol is represented by *d*
6. compresses the ranks with statistical coding

For prediction more than one symbol, some more rules are needed to include in the context length merge scheme. For example, an additional rule is “longer context matching gets a smaller rank”.

The baseline version of Content Prediction algorithm with the context length merge is used in proving information theoretic results as Figure 4.-1.

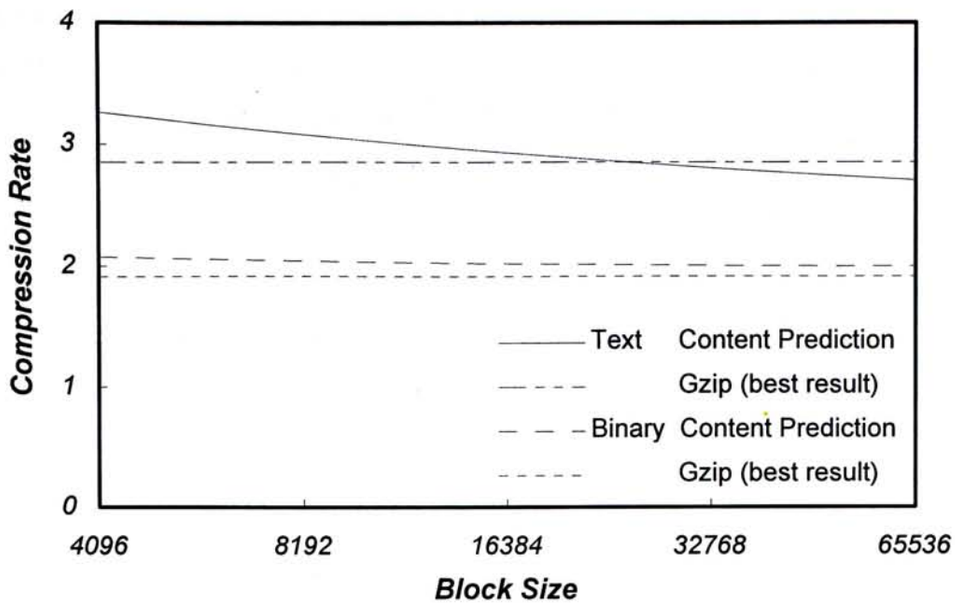


Figure 4.-1 Performance of Content Prediction for Different Buffer Size

Table 4.-1 in Appendix A shows up the context length is not the main factor in compression results. Along the context length, compression results are almost similar. Changing the option of the context length is only works at extreme cases (context length = 1, 2). The main influence for Content Prediction is appeared in the memory provided. Increasing the memory buffer, the compression rate* of Content Prediction are also increasing. In above results, For same memory consumed as GZIP, baseline Content Prediction versions leads GZIP for text data.

4.4 Discussions

In the above sections, I have presented the concept of the prediction and ranking schemes, and have introduced a general framework of Content Prediction lossless compression algorithm. The basic idea of the system is attempted to do learning and prediction like humans. Working in a sliding-window model, a simple Content Prediction implementation performs better than the GZIP algorithm. I think it is possible to achieve the result like PPM compressor by including more coding skills in Content Prediction algorithm.

There are three main approaches to improve compression performance of Content Prediction. These approaches rely on the modeling, prediction and coding. All these three

* compression rate = bits per input symbol

approaches have relations with each other. For modeling, the type of block sorting algorithm can be used in our algorithm and more detail is presented in next chapter. Another possible modeling choice is enlarger the prediction range, we can predict more than one symbol in each guess. More correct prediction, then we can reduce more redundancy. The methods of prediction are highly depended on the modeling.

As the result of prediction is an ordered list of symbols, from the most probable to least probable. Each input symbol is re-coded into its rank in the list and that rank is output as the code for the symbol. The quest now is to select symbol building the skewed frequency distribution, implying good compressibility. There are several methods which can be used to build the skewed frequency distribution. Shannon's results on English symbol prediction by human can be a good reference for further development [SHA51].

Guesses	1	2	3	4	5	>5
Probability	79%	8%	3%	2%	2%	5%

Table 4.-3 Shannon's results on human guess

According to Shannon's result, the first guess always does a very good prediction. For a less accuracy guess, the first answer is enough. Then no more prediction will be going on after the first prediction. The output is "true", or "false" plus the correct answer.

In a highly accuracy model, few more predictions can be proceeded for the correct answer. Failure of last guess causes the predictor output the correct answer. Reference to Shannon's result, two to five guesses are enough. The ranking scheme mentioned before uses all possible symbols. Maybe another ranking method based on the accuracy of predictor can improve the coding performance. Besides, some coding techniques for ranking list are proposed in next chapter.

PS. Some experiments for enlarge the prediction range have done. The system sets $N = 17$, $K = 8$, and uses ranking method that further prediction is going on until the correct answer be obtained. The compression results is similar as the baseline version. For better result on the case that enlarge the prediction range, we need to find a cutting-edge on the content matching and prediction.

5. Bounded-Length Block Sorting

At this chapter, I attempt to make a linkage between Content Prediction algorithm and Burrows-Wheeler Transform. Mixing the modeling schemes of such two algorithms, I use a bounded context length sorting as a context collector.

In following sections, I will describe the Block Sorting with a bounded context length. Such algorithm is a similar method to the static Content Prediction in block basis. Next section, I will present some ideas to improve the compression rate with symbol remapping in alphabet set, which are also applicable to the Burrows-Wheeler Transform.

5.1 Block Sorting with Bounded Context Length

Using a bounded context length sorting as a context collector, the system can use few memory and processing time. Before describing the operation of coding and decoding of the Bounded Length Block Sorting, I analyze the effect of context on the compression performance.

In normal statistical compression we consider each symbol of the file in relation to its preceding symbols or context. The inter-relations between symbols in the file mean that it is possible to predict most symbols with a high degree of confidence. The limited choice of possible symbols within the context means that few bits are needed for the encoding and considerable compression is achieved. In general, increasing the context (or number of preceding symbols being considered) narrows the choice of possible symbols and improves the compression. A maximum context of about 4-8 symbols is appropriate for most files. Above that length, any improvement in the actual coding of the symbols tends to be offset by the overheads in controlling and specifying the context; the compression remains constant or even deteriorates slightly.

Moreover, symbol context is scanned from right-to-left in most compression algorithms. But sometimes, it is possible to reverse the direction of scanning, left-to-right. These two scanning methods will consider in the following sections.

5.1.1 Forward Transformation

Following is the coding part of Block Sorting with a bounded context length. Block Sorting

with a bounded context length proceeds in the same pattern as the Burrows-Wheeler Transform. The only difference appears in the sorting preprocessor. Rather using the whole string as context, Bounded-Length Block Sorting use context in a bounded length. Therefore, the system can operate faster with less memory consumed.

For a bounded context length, system will order symbols due to their preceding context and recency. Compared with Content Prediction ranking scheme, Bounded-Length Block Sorting only misses the rule, "longer context matching gets a small rank". This is because Content Prediction is on-line adaptive context-driven algorithm and needs to reorder the codebook dictionary after each encoding, but it's static version in a block basis treat a block as a unit.

According the properties, Block Sorting with a bounded context length is a special case in Burrows-Wheeler Transform or static Content Prediction in block basis, and vice versa. Rather sorting all elements in block, Bounded-Length Block Sorting is possible using the context in any length. However, such additional characteristic causes the prefect reconstruction rule of Burrows-Wheeler Transform algorithm no longer works.

The forward transformation of Block Sorting with a bounded context length can be described as —

1. Sort the input symbols, using as a key for each symbol, to a fixed length is needed to resolve the comparison. Key is a bounded length context of the symbol-to-be-sorted. The symbols are therefore sorted according to their preceding contexts.
2. Take as output the sorted symbols, together with the position in that output of the second symbol of the input data.

5.1.2 Reverse Transformation

The decoding operation of Bounded Length Block Sorting is explained here. Before introduction the reverse transformation of Bounded-Length Block Sorting, I will introduce its extreme case, Block Sorting. To illustrate the operation of encoding and decoding we consider the string "good, very good" as shown in Figure 5.-1.

In Figure 5.-1, Block Sorting is implemented as static Content Prediction in block basis, encoding uses the whole block as the context. But in Figure 5.-2, normal case of Bounded-Length Block Sorting, much shorter context is used. The output of Block Sorting and the output of Bounded-Length Block Sorting are much similar to each other. However, using the context less than the length of block, Bounded-Length Block Sorting cannot reconstruct by using the simple rule likes Burrows-Wheeler Transform.

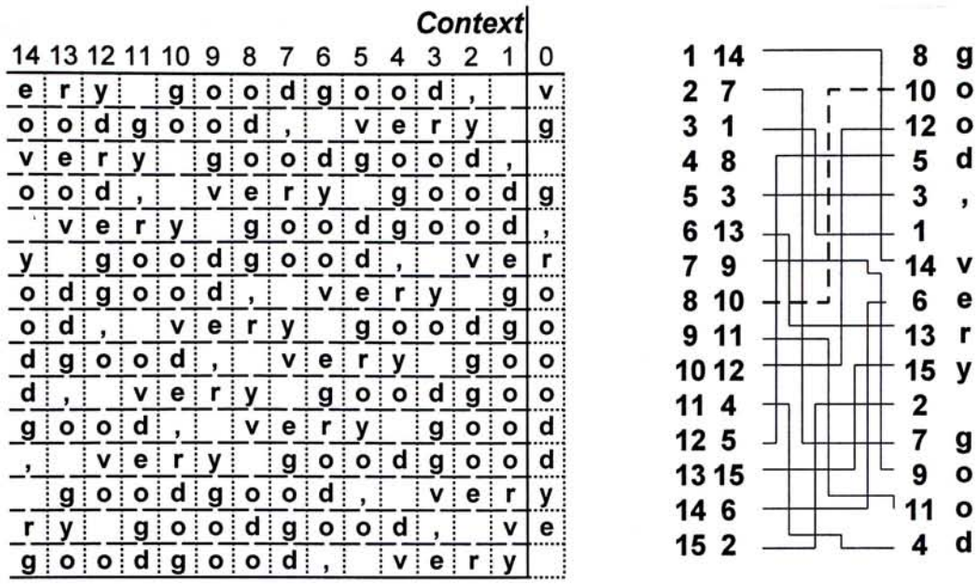


Figure 5.-1 The operation of coding and decoding of right-to-left scanning Block Sorting Algorithm

Analyzing the reconstruction rule of Burrows-Wheeler Transform algorithm, the main elementary property is that reverse transformation requires the character following each context can be uniquely identified. Using contexts less the length of block, the uniqueness of each context can be guaranteed if and only if we known all these contexts at decoding. In the following, I will introduce a method to find out contexts used in coding with the primary index and the copy of the ranked list.

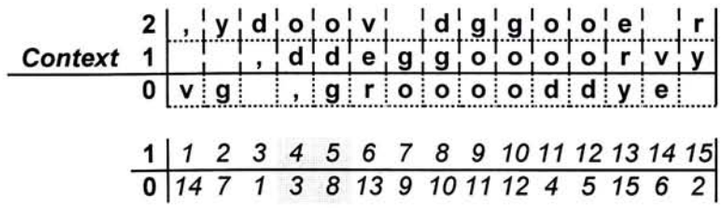


Figure 5.-2 Block Sorting with an order-2 context

Finding out all contexts of the transmitted symbol, we can starting at column 1. As all elements of permutation are sorted in this column. It is easier for us to rebuild it by sorting transmitted symbols. The order-1 contexts plus transmitted symbols, bi-symbol pairs are formed. Actually, these bi-symbol pairs in sorted order are the order-2 contexts of transmitted symbols. By using the same method to find out the column 2, we can find out all contexts used in the coding. With the knowledge of contexts of transmitted symbols and index to the starting point of original string, we can construct the original string. During the decoding, we should output the early symbol first if same context is using among symbols. It is because Bounded-Length Block Sorting will keep the

early symbol ahead for same context cases.

Alternatively, the decoding can also operate in another approach. The differences between output of Burrows-Wheeler Transform and output of Bounded-Length Block Sorting are appeared only in cases that same context are used in coding. Although this discover cannot help to change the output of Bounded-Length Block Sorting to the output of Burrows-Wheeler Transform, it tells us a fact that the prefect reconstruction rule in Burrows-Wheeler Transform can use backward rebuild the context used in coding correctly.

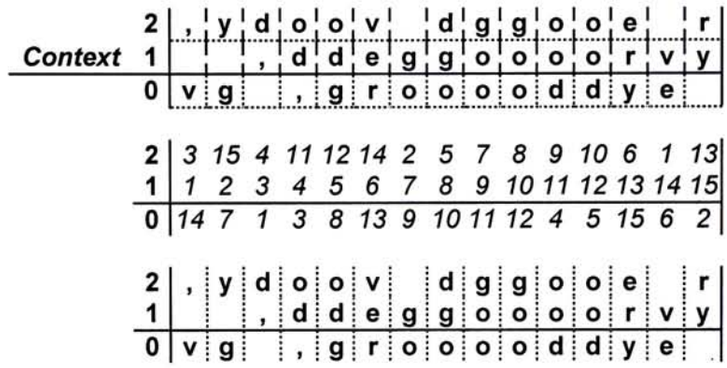


Figure 5.-3 Context rebuilding by the reconstruction rule of Burrows-Wheeler Transform

Same as decoding method mentioned above, we use the index to the starting point of original string and output symbols rely on their contexts. Another method also can be used. Since the prefect reconstruction rule only has error in cases that same context are used, symbols that not following same context are still correct by using this rule.

As shown in Figure 5.-3, the prefect reconstruction rule is still work in some cases. After the generation of contexts, we mark same context cases be wrong and do not further processing in the rule. Some partial sequences of the original string still can be reconstructed by passing the rule on unmarked cases. After processing unmarked cases, just left the first one to be a pointer on that sequence and erase the others. Plus correct elements in the context of marked row that reverse by prefect reconstruction rule. Then we can start at the index that indicated the original string. We go into a site either marked wrong relationship row, or a partial sequence of the original string. Meeting the partial sequence, we get a short cut in the decoding, just output that sequence. At the marked wrong relationship row, with elements in the context, output the symbol due to the order of sorted directory.

In short, the reverse transformation of block sorting with a bounded length context can be summary as following steps.

1. Generate all contexts used in forward transformation with the prefect reconstruction rule in Burrows-Wheeler Transform.
2. The further procession is divided into two cases.
   a. more than one symbol use the same context in coding
      Find the symbol due to its context, and break ties by recency.
   b. the context is only appeared one in coding
      Reconstruct the symbol by rule in Burrows-Wheeler Transform.

Actually, two steps can be combined together and operated in parallel. Therefore, the system can check the context and rebuild the original sequence during generation of the contexts. One illustration is shown in Figure 5.-4.

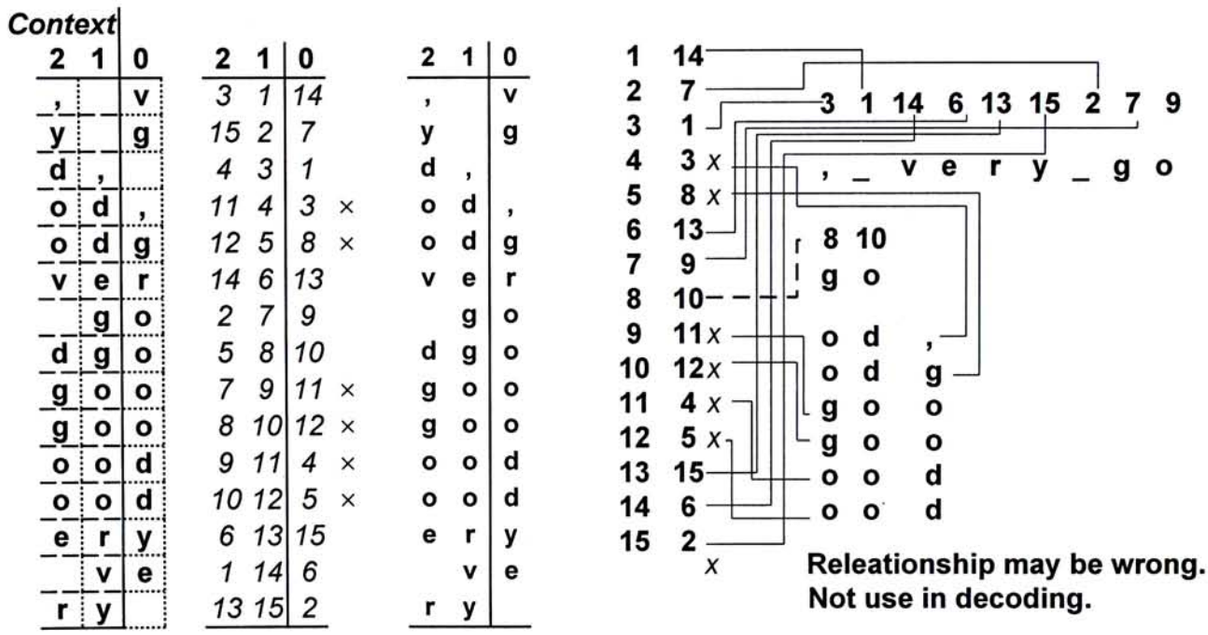


Figure 5.-4 Partially reverse transformation by the reconstruction rule

In Figure 5.-4, decoding with the sorted directory, the starting point is 8 and "go" is output. Based on the context "go", "o" is output. Then "d" is output. At the row with context "od", "," should output first, so sequence ", very go" will copy out. Following "go", "o" and "d" can be decoded.

Prediction of symbol ranking is essentially what the context collection (sorting) algorithms do, although with a permutation of the input text to increase locality effects. The Move-To-Front list approximates an ordering in symbol frequency, and the emitted index is simply an error indication. The Bounded-Length Block Sorting compressor with MTF processing and Arithmetic Entropy Coding is performed as following.

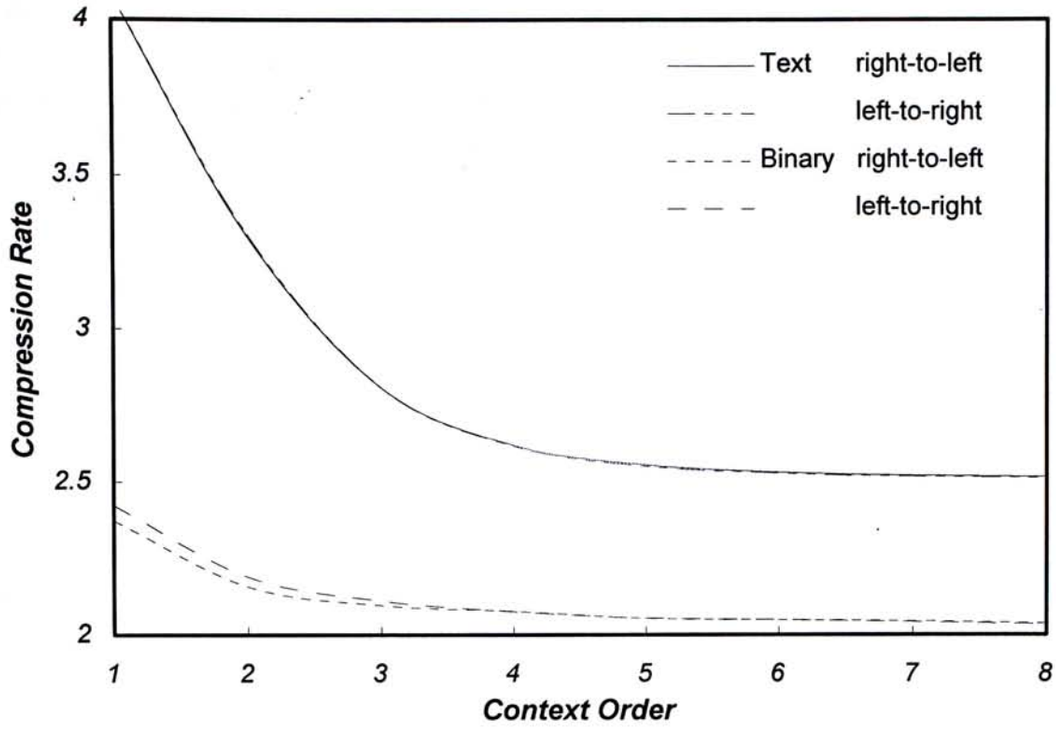


Figure 5.-5 Performance of Bounded-Length Block Sorting

Beside testing how the order of context affects performance of Bounded-Length Block Sorting, the context scanning method is also considered. The order of context will help in the compression at extreme cases (the length of context is very short). After order-4, compression rates in Figure 5.-5 become flat. The most detailed values can be find in Appendix A, Table 5.-1. One more interesting thing is that right-to-left context scanning is better at low order context cases. Left-to-right context scanning is better at high order context cases.

5.2 Locally Adaptive Entropy Coding

In additional to Block Sorting with a bounded context length, some coding techniques can improve the compression performance. Modifies on the MTF and arithmetic coding are main contributions in followings.

The order-0 statistical model of Move-To-Front output is at best only an approximation or averaging-out of local contexts, probably with considerable local deviation from that model. However, during the processing on the symbol ranking sequence, we know about the frequency distribution of outputs. We should be coding more efficiently.

In the symbol ranking output, first few MTF ranks contain almost all elements. Improving compression over that achieved by the order-0 model requires models which can adapt quickly to local changes in frequency. With adaptive arithmetic coding this requires a model containing only a few symbols and with a small count limit, so that statistics are sensitive to just a few added symbols and there is frequent rescaling to provide locality. One good starting point is applying different weighting on the ranks, e.g. {0, 1~4, 5~255} likes Shannon's human results on symbol guess (Table 4.-2). Moreover, an adaptive frequency ordering method is included in Move-To-Front Coding. If the first symbol in the MTF list is appear continuously, the new symbol is inserted into position 1 rather than position 0 as MTF.

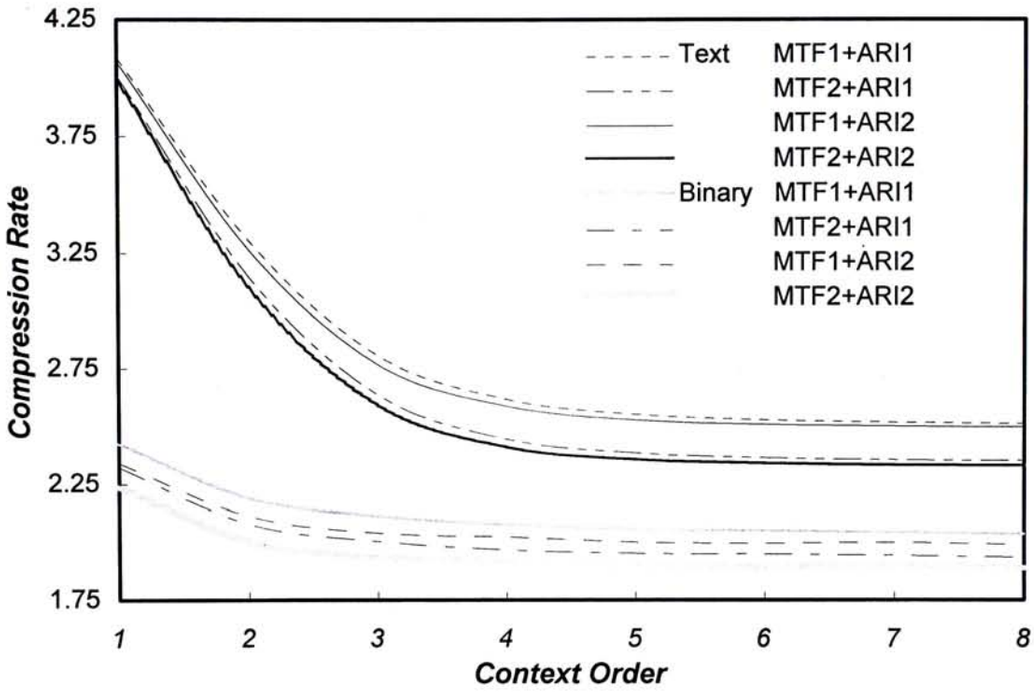


Figure 5.-6 Extensions for the Bounded-Length Block Sorting

In experiments, both context scanning methods are processing with new MTF and new Arithmetic coding. The results are mentioned in Figure 5.-6 and Table 5.-1 at Appendix A. The new MTF and new Arithmetic coding are marked as version 2 in above figure. Adaptive frequency ordering on MTF for the ranking list is always improved the performance about 5~6%. Plus the proposal Arithmetic Coding based on Shannon's human guess, compression rates can reduce near 7.5%.

Some good ideas for improving compression results for the symbol ranking list are published recently. A new technique is Inversion Frequencies [AM97] by Ziya Arnavut and Spyros S. Magliveras. One best approach is using caches and run-length coding [FEN96] by Peter Fenwick.

I have got a few more improvements by using it as locally adaptive entropy coding to Bounded-Length Block Sorting algorithm. The method is to use a small "cache" model which holds only the first few, or most probable, MTF codes, escaping to a complete, background, model to other values. With much of the MTF output being simply runs of zeros, it seems reasonable to try run length coding of the zero values.

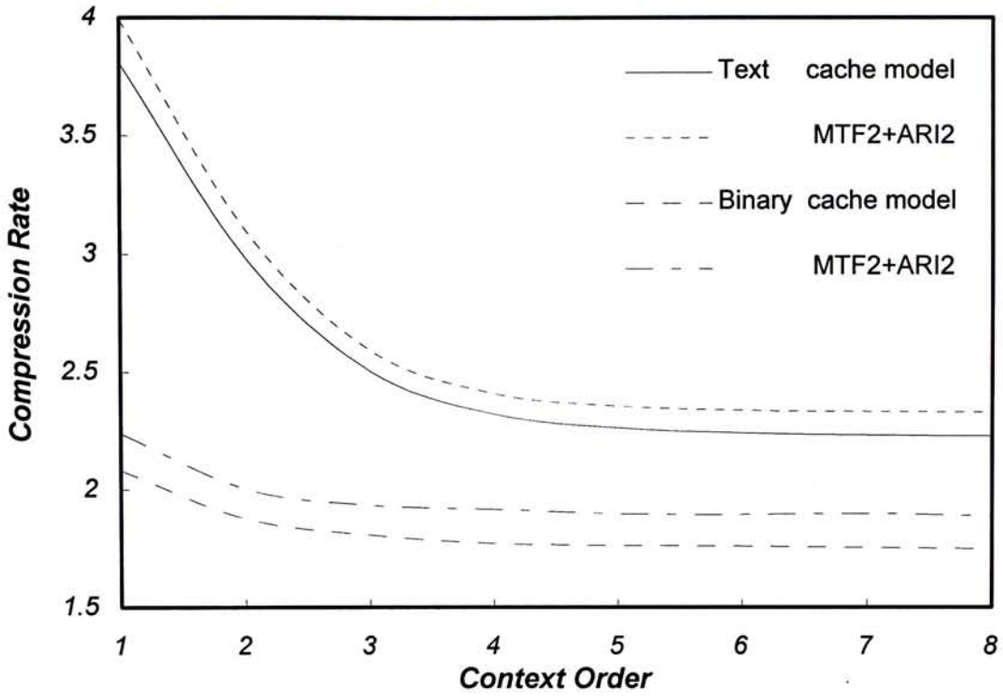


Figure 5.-7 Locally adaptive coding improvement by using cache model

Cache model can farther improve the ranking list from Bounded-Length Block Sorting. As the performance curve is similar to my proposal methods, I think the gap is due to modeling updates or the different between Shannon's human guess and machine learning. Some more experiments are need to done for a better update step for fitting the real world cases.

5.3 Discussion

In above sections, I have presented another possible application of Content Prediction framework. This application, Bounded-Length Block Sorting, is a linkage between context sorting and block sorting. With sorting on the low order context is possible to archive the similar performance of Burrows-Wheeler Transform, but consumes fewer time. In addition, the adaptive

frequency ordering in MTF and arithmetic coding with 3-ary model are proposed for locally adaptive coding on the ranking list. Caches and run-length coding is testing for Bounded-Length Block Sorting too.

Bounded-Length Block Sorting has a very different kind implementation rather than Burrows-Wheeler Transform and baseline Content Prediction. The extensions of Content Prediction that I present in Chapter 4.3 may not be used in Bounded-Length Block Sorting. As the prediction style of Bounded-Length Block Sorting is no longer operation adaptive, the system may not be possible to modify for further prediction. But system ranks with a locally adaptive entropy coding is another promise techniques for static Content Prediction.

The next stage for static Content Prediction can be divided into two different approaches. One is improvement on the sorting style, another is using a different set of symbols in the rotation matrix. In the sorting part, as the prior and follow symbols are important to prediction in the same weight, we could construct a sorting algorithm that can use prior and follow symbols in the same time. At rotation matrix, we could use other columns to instead of the current one. May be a mixture of second column, third one, and so on can be a good choice.

6. Context Coding for Image Data

In this chapter I will review some common image compression techniques and proposed some context model or coding techniques for improvement on the embedded zerotree wavelet image coding.

Image is very different kind of data to the text. In fact, over the long run, histograms for live images from sources such as television tend to be flat. This means that each pixel code has approximately the same chance of appearing as any other, negating any opportunity for exploiting entropy differences. Besides, the length of matching strings tends to be small because of the vagaries of the real world. All these limits the effectiveness of compression by using the lossless coding methods mentioned before.

However, a small alteration on pixel could be undetectable or meaningless to the human eye. Thus, a lossy compression has an advantage on image data files. They can be slightly modified during the compression or expansion cycle and unnoticed if the modifications are done carefully. In section 6.1, basic characteristics of image data are introduced. The model of image coding system, aimed at reducing image data redundancies, is described in section 6.2. The following sections are addressed on the zerotree data structure coding and possible extensions.

6.1 Digital Images

Before describing the context coding modeling, I like to introduce some basic characteristics of image data.

A two dimensional image can be represented by a light intensity function $f(x,y)$, where x and y denote the spatial coordinates and the value of function $f(x,y)$ that is proportional to the brightness of image at point (x,y). A digital image is formed by discretizing both the spatial resolution and the brightness function. The picture elements in a digitalized image are called pixels. The performance of an image compression method must be revised with three aspects in mind, *Compression ratio*, *Image quality*, and *Computational cost*.

In general it is always possible to improve one aspect at the cost of degrading the performance of the others. A good compression method takes all three aspects into account, and it is essential to make a survey of all aspects together when evaluating a specific compression method. The performance of the proposed image coding algorithm will be compared with two

popular image compression techniques, Discrete Cosine Transform and Wavelet Transform. A brief summary of these two techniques is shown in Table 6.-1, and detailed introduction can be found in Appendix B.

Table 6.-1 Summary of some popular image compression techniques

Discrete Cosine Transform

widespread standard (JPEG)
fast
excellent compression-quality tradeoffs at medium to high rates
errors affect small areas

Wavelet Transform

no standard (yet)
fast
excellent compression-quality tradeoffs at all rates
errors affect entire image

6.1.1 Redundancy

In compression of digital images there exist three basic image data redundancies :

- ***Coding redundancy*** consists of a nonoptimal way of describing an image. The typical approach to eliminate coding redundancy is to examine the probabilities of the pixel values in the image and then assign a variable length code to each pixel value according to its probability ; the higher probability the shorter code.

- ***Interpixel redundancy*** stems from the fact that there in most images is a correlation between these pixels. A real world image can be modeled as Markov process where the value of the current pixel in some way has dependency with the *n* previous pixel. This kind of redundancy can be removed by transforming the image to a state where the interpixel redundancy can be discovered and eliminated, and this kind of transformation process is called a mapping. It is referred to as reversible mapping if the original image can be reconstructed from the transformed data set without any loss. Interpixel redundancy is also know as spatial, geometric or interframe redundancy.

- ***Psychovisual redundancy*** The human perception is not a constant pixel oriented mechanism. This implies that every area in a visual field is not processed with the same amount of sensitivity, and that areas in the image which do not contribute with valuable visual content possible can be removed without major loss in quality for the human perceiver. This

unnecessary image content is called psychovisual redundant information, and elimination of these redundancies is a sort of quantization, which is an irreversible process.

6.2 Model of Image Compression System

To reduce three kinds of redundancy that mentioned before, a general model of image coding system can be found here.

Figure 6.-1 model of image compression system

A model of a general compression system (Figure 6.-1) consists of three stages : representation, quantization, and finally a lossless encoding, all aimed at reducing a specific redundancy. Block-based transforms (such as the DCT) and wavelet decompositions are commonly used to convert an image into a representation with good energy compaction. The transform coefficients are then quantized to reduce the information and achieve the desired bitrate. The quantized coefficient image is then entropy encoded in a lossless fashion. We have shown (for wavelets) that good results can be obtained with such a framework. Recently, however, more sophisticated techniques have surfaced which, in some sense, analyze the image to exploit higher level correlations that exist in the transform domain. One such technique, which has received a lot of attention, is the embedded zerotree wavelet (EZW) code.

In the following section, after introducing lossy image compression, I discuss the EZW code, analyze its behavior, and then propose a context modeling techniques to achieve better performance.

6.2.1 Representation

The representation operation maps the image from one format to another. The idea behind this operation is to represent the image in a format that interpixel redundancy can be discovered. Common methods include block based transforms, such as the DCT used in JPEG, and joint spatial

frequency based decompositions as used in subband and wavelet coding. From an appropriate viewpoint, block based DCT codes can be interpreted as a subband coding techniques — all one has to do is reorder the data so that coefficients which share the same frequency band are grouped together.

A major problem, however, is that in block based codes there is no interaction between pixels in different blocks which, when coupled with coarse quantization, results in blocking artifacts. Wavelet techniques decompose the image into frequency bands and, because they are filtering based approaches, do not suffer from blocking artifacts and typically generate higher quality images at low bitrates. In wavelet transforms, representations differ in their choice of wavelets. I will try orthogonal wavelet and biorthogonal wavelet filters and the associated representations in the context of image coding.

6.2.2 Quantization

Typically, the number of samples resulting from image transformations remains the same, but the precision required to specify the transform coefficients increases. Often, the output of the representation is a set of real-valued coefficients, which we cannot encode with a finite number of bits. Thus, quantization has the job of degrading the accuracy of the mapping operation according to a given fidelity criterion. This job reduces both of the interpixel and psychovisual redundancy. Furthermore, quantization is often the only way we can reduce the information content of the source in a controlled fashion. In all common transformations, there is some notion of frequency in the transform domain, and better quantizers exploit the human visual system by quantizing higher frequencies, where errors are less visible, more coarsely than lower frequencies.

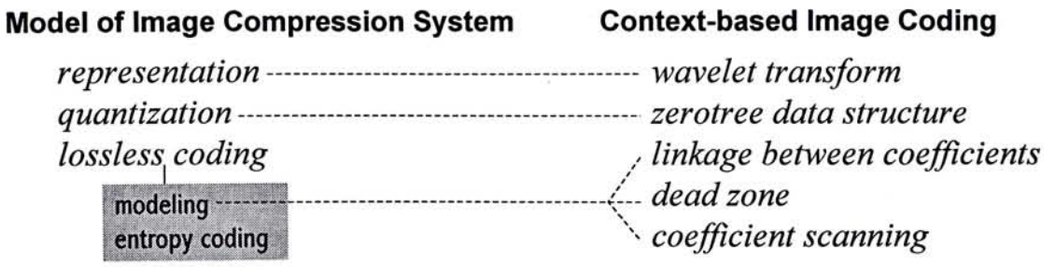
Suppose we have decomposed an image to N dyadic scales using a wavelet transform, either orthogonal or biorthogonal. This will yield $3N+1$ wavelet subbands. Since the variance of each subband is generally different, we need to design a quantizer for each subband. Scalar quantization is a solution with much simpler implementation. Doing uniform quantiztion among the value of pixel can be worked satisfactorily in most cases. This design is obviously nonoptimal, one better quantizer is Vector Quantization. VQ is a generalization of scalar quantization on which vectors, or blocks, of pixels are quantized instead of the pixels themselves. To apply VQ to wavelet image coding, the common approach is to consider each subband individually. Since subbands are a hierarchical organization of oriented frequency bands, it is intuitive to consider quantizing a vector whose elements span subbands of the same orientation. Such idea is referred as Space-Frequency Quantization [XOR96, XOR97]. For better visual effect, Human Visual System Characteristics are always considered in the image coding. A general description of HVS can be found in Appendix C.

6.2.3 Lossless coding

Lossless coding is responsible for suppressing the coding redundancy from two previous stages. This stage of encoding is reversible in contrast to the output of the quantizer. Huffman or Arithmetic coding are commonly using in the this stage. Some universal code based on an ensemble of typical images plus the encoded image data are always saving more space. For highly skewed sources, such as abundance of zeros in quantized wavelet transformed images, Run-Length coding can be combined to remove the zeros. The locations of non-zero pixel are specified by encoding a binary activity mask (all non-zero values are set to 1) with standard binary image compression techniques, such as JBIG. The non-zero pixels are mapped through a balanced binary tree and encoded. Some alternative, efficient representations of the zeros in the source are Shapiro's zerotree coding [SHA93] and stack-run coding by M.J. Tsai, J.D. Villasenor and F. Chen [TVC96].

After transmission through the channel the source decoder reconstructs the original image. It consists of an entropy decoder and an inverse transformation. The decode process corresponds to the inverse action of the source encoder, except that an inverse quantizer is not possible due to the irreversible process of the quantization.

In this thesis, the model of image compression system is designed as followings. I use wavelet transform as the image representation is due to its excellent compression-quality tradeoffs at all rates. Zerotree data structure is chose because of simplicity and adaptiveness. The main issue of context-based image coding is central on the modeling part.



6.3 The Embedded Zerotree Wavelet Coding

Since wavelet is a relative new and promising development in the area of lossy image compression, we will develop the context coding model for this area. Among different wavelet

coding algorithms, we choose the Embedded Zerotree Wavelet Coding because of simplicity and adaptiveness.

The EZW Coding encodes images, in an embedded fashion, from their dyadic wavelet representations. The goal of embedded coding is to generate a single encoded bitstream which can be truncated, to achieve any desired rate, and used to reconstruct the best possible rendition at that rate. Since wavelet representations have both scale (frequency) and space contents, spatial grouping of data and quantization are possible. Qunantization in the EZW is done by successive approximation across the subbands with same orientation. This results in an efficient data structure for encoding zero and nonzero quantized value.

The main idea of EZW is the wavelet transform coefficients with the same magnitude are assumed to have equal importance, and should be transmitted before coefficients with smaller magnitudes. This technique is based on three concepts: *1.) partial ordering of the transformed image elements by magnitude, with transmission of order by a subset partitioning algorithm that is duplicated at the decoder*, *2.) ordered bit plane transmission of refinement bits*, and *3.) exploitation of the self-similarity of the image wavelet transform across different scales.*

As to be explained, the partial ordering is a result of comparison of transform element (coefficient) magnitudes to a set of octavely decreasing thresholds, $T_0 > T_1 > \cdots > T_{N-1} = T_{\min}$. We say that an element is significant or insignificant with respect to a given threshold, depending on whether or not it exceeds that threshold. Significance pixel are processed during each pass. Typically, $T_i = T_{i-1}/2$ just like the bitplane coding.

As described by Shapiro, the zerotree coder maintains two lists, a dominant list and a subordinate list. Initially, all pixels are placed on the dominant list in a predefined order. In the dominant pass, this list is scanned and the location of all pixels whose magnitude exceeds the first significance threshold, T_0 , and their signs, are encoded using the zerotree data structure. These pixels are then transferred to the subordinate list, and the corresponding coefficients in the wavelet transform image set to zero so that their location is not encoded again in later passes. In the subordinate pass, the next bit in the representation of each pixel on the subordinate list is encoded. The subordinate list is then sorted (using only the information that is known at the decoder), and the process repeated for each threshold until a bitrate target is met or $T_{\min}$ is reached.

Shapiro elected to encode the sign bits as part of the zerotree and chose a 4 symbol representation to encode it : zerotree root (ZTR), isolated zero (IZ), positive significant (POS) and negative significant (NEG). A zerotree root is used to indicate that the entire subtree rooted at the corresponding node is zero (or insignificant), and allows an efficient description of large all white blocks. The isolated zero symbol is used to indicate that a pixel is not significant, but that one of its children is. Significant symbols are classified as positive or negative significant. Then EZW symbols

are drawn from the three different alphabets $S_0 = \{0,1\}$, $S_1 = \{ZTR,IZ,POS,NEG\}$, and $S_2 = \{Zero,POS,NEG\}$. For processing the quantization precision in the subordinate pass, 0 and 1 in S_0 are used to the significant coefficients' interval. The alphabet set S_1 is used to encode the sign bits in the dominant list. As there are 3 subbands (HH_0, HL_0, LH_0) not need to distinguish between the zerotree root and isolated zero, a simpler alphabet set S_2 is designed for these cases.

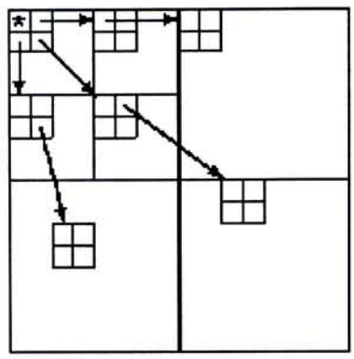


Figure 6.-2 zerotree data structure in wavelet decomposition

The zerotree data structure is of central importance in the EZW code. Its importance lies in its ability to efficiently encode large blocks of zeros in the significance maps, and its exploitation of the hierarchical correlation in wavelet transformed images. An example significance map (augmented with isolated zeros, zerotree roots and sign information) and its corresponding zerotree are shown in Figure 6.-2. All the zeros are represented by the zerotree root at their parents.

For entropy encoding, Shapiro conditions zerotree symbols using the significance of a pixels' parent and the previous pixel in the defined ordering. An isolated zero cannot occur at the leaf nodes, so that a ternary alphabet can be used for the highest frequency bands. The bits encoded in the subordinate pass are encoded in a single context, without any conditioning. All subsources are encoded using an adaptive arithmetic code 8 with a maximum frequency count of 256.

6.3.1 Simple Zerotree-like Implementation

The following sections are the context coding algorithm with EZW structure. At first, we analyze some basic components of wavelet coding. And then we will check the internal regularity of the Embedded Zerotree Wavelet image coding. A context coding method based on the experiments are acquainted in the section 6.4.

At first, I have implemented a simple zerotree-like coder without ordering on the lists. Besides, the LL band is also encoded separately in lossless for preserve the lowest frequency components, and the coefficients of each subband are rescaling to range of -128~127. I use a 6-level pyramid wavelet decomposition of Lena*, based on the orthogonal 6 wavelet filter of Daubechies and biorthogonal 7/9 wavelet filter of Barlaud [ABMD92, BMH92]. The implementation is mainly designed for testing different options in zerotree-like wavelet coding rather than effectiveness.

	Daubechies 6	Barlaud 7/9
Bit Rate	0.56	0.47
PSNR	33.3622	33.3056

Table 6.-2 Comparison between Orthogonal and Biorthogonal wavelet filters

Operation with Daubechies 6 and Barlaud 7/9, the result is shown in the Table 6.-2. For the similar PSNR, Barlaud 7/9 leads Daubechies 6 approximately 0.1 bits/pixel. The result forces me using the Barlaud 7/9 wavelet filter in the followed cases.

6.3.2 Analysis of Zerotree Coding

Using the basic zerotree-like wavelet coder implemented in the last section, I have done some experiments for finding useful context elements for zerotree coding. Typically the coding procession can be divided into two components : modeling and entropy coding. The goal of modeling is to predict the distribution to be used to encode each pixel and context information is included in the models in this part.

In coding, the goal is to encode each sequence of symbols $\{s_1, s_2, \cdots, s_n\}$ with $-\log_2 p(s_1, s_2, \cdots, s_n)$ bits. Without any prior information (context) about the symbols, the entropy reduces to $-n \log_2 p(s)$, where $p(s) = p(s_i)$ for all *i*. If not, modeling is the task of estimating $p(s_1, s_2, \cdots, s_n)$. Reformulating the problem as that of estimating a set of conditional probabilities, i.e.,

$$p(s_1, s_2, \cdots, s_n) = \prod_i p(s_i | s_1, \cdots, s_{i-1}) \qquad (6.-1)$$

Our goal is to reduce the problem of estimating the symbol distributions to a manageable size. This is accomplished by restricting the range of context to a small set of states. Now, we can

* The Lena I used for testing may be Lean-Y.

associate with each state a conditional source comprised of all symbols which occur in that state.

It is possible using all previously transmitted symbols as context. Owing to complexity constraints, we must reduce the range of it. This is done by choosing a set of pixels from the set of all previously transmitted pixels. This choice is of utmost importance, since our prediction (classification) is based entirely upon these pixels. In the Shapiro's zerotree coder, this corresponds to previous pixel in define order and the parent pixel. These should be the pixels that supply the most information about the current pixel. Instead of following the Shapiro's step on the context choice, I developed a set of conditional probabilities on the different context for coding. Before I describe the context-based coding I used, I will present some experiments for finding the context.

6.3.2.1 Linkage between Coefficients

Now consider the relationship between wavelet subbands. Figure 6.-3 shows the magnitudes of wavelet coefficients in a 2-level pyramid decomposition. It is visually apparent that the coefficients with large magnitude tend to occur at the same relative locations in subbands at different levels. This is true when comparing subbands of the same orientation, and also holds across orientations.

Figure 6.-3 2-level pyramid decomposition of wavelet coefficients

To capture this relationship more precisely, I consider the joint statistics for two coefficient subbands. Figure 6.-4 shows the conditional statistics on wavelet coefficients of a 6-level pyramid decomposition. That is, we plot the conditional histogram of most-significant-bit of the parent coefficient against children coefficient over the 6-level pyramid decomposition. When the magnitude of the parent coefficient is large, the expected value of the magnitude of the child appears to be linearly related to the magnitude of the parent. When the parent magnitude is small, the magnitude of the child is independent of the parent magnitude.

Although the parent pixels provide some information related to the magnitude of the child pixels, it is not always correct. The solutions for solving this problem either not use parent pixels for context or coding with parent pixels separately based on their magnitude.

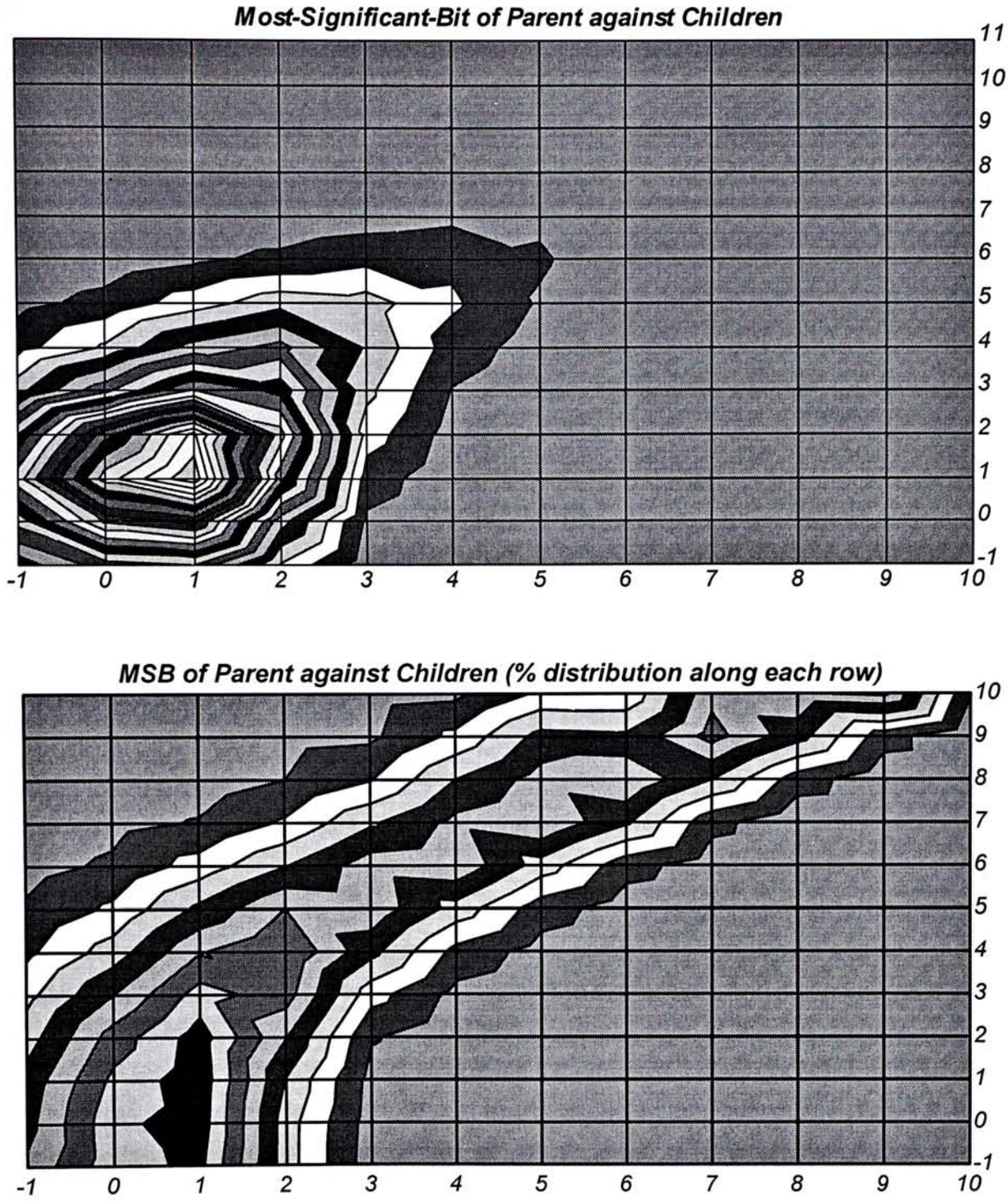


Figure 6.-4 Conditional histogram and distribution of coefficient magnitudes of "Lena"

In my first context-based zerotree-like wavelet coding, I do not use the parent pixels in most cases. The parent pixels only applies in case that the pixels of three subbands adjacent to the LL band are need to encode. The previous pixels are also not include in the context. Instead of parent and previous pixels, I use the pixels in the same relative position at other two subbands on the same level. Thus, pixel in HH band will take the pixels at same position in LH and HL bands. If there is no such two pixels to be the context, then previous pixels are used for coding. For easier

implementation, I sort the lists for handling the above requirements.

Threshold	16	32	64	128	256	512	1024	2048	4096	8192
PSNR	35.8294	32.6629	29.4352	26.4111	23.5983					
Raw Size	129019	64947	29563	12462	5048	2346	1195	741	459	169
Context Order										
0	15198	7616	3622	1673	739	362	187	117	79	34
1	15252	7656	3609	1606	636	276	120	65	44	22
2	15102	7417	3402	1480	587	256	111	63	44	24
3	14890	7295	3354	1467	589	266	117	68	47	26
4	14927	7381	3434	1516	621	284	128	74	52	28
Bit Rate	0.4544	0.2226	0.1024	0.0448	0.0179					

Table 6.-3 Performance of context-based wavelet image coding (using zerotree data structure)

The experiments' results are shown in Table 6.-3. I use 6-level pyramid wavelet decomposition of a 512 x 512 gray scale image Lena* based on Barlaud 7/9 wavelet filter. In comparing the results of Shapiro's and my implementation, my wavelet coding with order-3 context model always achieve better or same performance as EZW in all cases.

At each threshold pass, I have tried to use different length context to do the modeling. Without mainly depending on the parent and previous pixels, I do conditional coding based on the pixels in the same position at two adjacent bands on the same level. And it can still get a very good result as the Shapiro's. In general, short context perform better at the low bit rate coding, long context lead at the higher bit rate coding. During the experiments, I have got some additional results. Rather coding for the whole list, I do coding at each threshold pass individually (Table 6.-4).

Threshold	8192	4096	2048	1024	512	256	128	64	32	16
Raw Size	169	290	282	454	1151	2702	7414	17101	35384	64072
Context Order										
0	34	51	46	74	187	360	923	1921	3965	7555
1	22	22	27	61	163	368	960	1955	3972	7525
2	24	23	27	58	157	344	917	1958	4046	7593
3	26	26	30	62	166	350	921	1957	4034	7699
4	28	29	33	69	179	373	964	2028	4131	7808

Table 6.-4 Compression results for coding individually at each threshold pass

* The Lena I used for testing may be Lena-Y.

Coding individually has the similar phenomenon as coding the whole list, short context perform better at the low bit rate coding and long context lead at the higher bit rate coding. Some extreme cases happen after threshold = 64. Without using any context, the coder get the similar result as the using context one. This phenomenon is happened at the bitplane coding. At the less significant bit, the image data is become random.

According to above testing, a better compression scheme for zerotree-like image coding is refreshing context model after each threshold pass. Besides, short context can be used in low bit rate coding for saving modeling cost. The long context estimates more accuracy probability model and applies in high bit rate coding. Further, as random characteristics of less significant bits in the Image data, we can just use a memoryless entropy coder in the threshold pass which threshold value is less than 128.

6.3.2.2 Design of uniform threshold quantizer with dead zone

Another special interest in the testing is a similar skill of "dead zone". The "dead zone" is a technique used in uniform threshold quantization that produces data closely approximated by a Laplacian distribution. In my implementation, I just assume the middle of the integer interval be the boundary without computing any probabilities. This approach can bring the benefit of "dead zone" in my wavelet coding, and do not increase any complexity. Then the uniform threshold quantization for the above implement has a fixed step size (expect for the "dead zone" around zero).

To apply the idea of "dead zone" in above zerotree-like wavelet compression, two possible ways we can do. One is using in the decode part only, and another can use in both encode and decode parts. I use the case where threshold = 64 in the simulation.

Baseline	
Raw Size	29563
Coded Size	3354
Bit Rate	0.102
PSNR	29.4352

Use "dead zone" in decode	
Raw Size	29563
Coded Size	3354
Bit Rate	0.102
PSNR	29.4518

Use in encode & decode	
Raw Size	29757
Coded Size	3375
Bit Rate	0.103
PSNR	29.4889

Table 6.-5 Compression results for coding with "dead zone"

"Dead zone" is a minor skill for increasing the PSNR (about 0.05 dB). Generally, it does not affect the image quality in the coder with uniform classification, it just makes the result seems better. However, such skill may be useful in the other non-uniform classifications, like Estimation Quantization [LRO97], Space-Frequency Quantization and so on.

6.4 Extensions on Wavelet Coding

Due to the experiments' results of above sections, an advanced context model for image coding is proposed here. Further improvement on the wavelet coding, there are various basic components in a wavelet-based image coder can focus on, namely, the tree-structured filterbank, the filters themselves, the quantizers and the entropy coders.

An extensive analysis of correlations in the wavelet transformed image would be required to determine the optimum contexts for the encoding. In this thesis I concentrate on the issue of adaptive quantization and entropy coding for a fixed filterbank (Barlaud 7/9). The issues for other possible improvements are not considered here.

There are two main approaches for context-based image data compression. The first approach relies on a fixed quantization for all coefficients in a given band and a layered transmission of the coefficients using binary or low order arithmetic coding. A best example is SPIHT by Said and Pearlman. Context information is taken into account by using the zerotree data structure which enables the joint transmission of zero-valued coefficient present at the same spatial location across several frequency bands. The second approach is relied on using different quantizers for different subbands. With the knowledge of minimum, maximum and variance in each band, the most effective quantization method and entropy coding scheme are selected.

I am considering a context-based adaptive arithmetic coding for the wavelet image coding. In additional to pixel at adjacent frequency bands and parent, I add neighboring four pixels and same pixel at previous bitplane into the model. If the context which needs a pixel is still not encoded, that pixel in the previous bitplane is used. Moreover, some skills mentioned before also add into the extended version. Just like sorting the subordinate list and reset the arithmetic coding after each threshold pass are proved useful for coding. Contexts used for the wavelet coding is shown in Figure 6.-4.

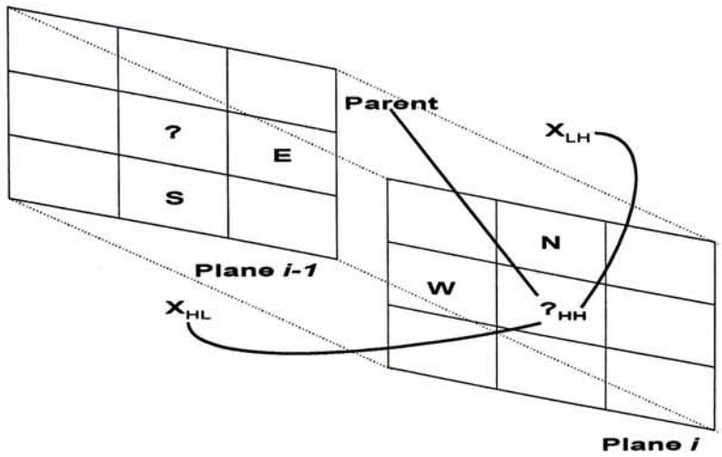
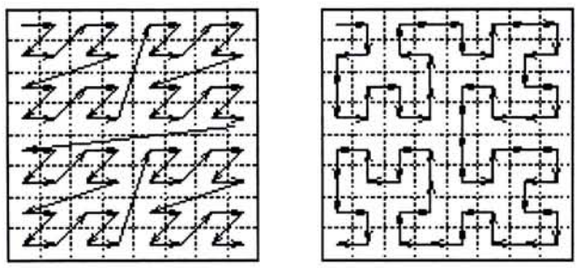


Figure 6.-4 Context model for estimating a pixel in HH band.

6.4.1 Coefficients Scanning

Coefficient scanning is another skill to help in the coding. The zerotree coding method takes into consideration that the edge energy is spread out in a Wavelet representation in an ordered manner. Since the original image is recursively down-sampled at a 2 by 2 ratio, each representation coefficient in a lower band is spatially related to 4 other coefficients in the immediately higher band. This relation structure can be carried out another skill for the modeling, coefficient scanning. A zerotree is encoded by encoding the symbols encountered on a predetermined path through the corresponding augmented significance map. In the pyramid decomposition structure, 4 child coefficients under a same parent have correlation with each other. Instead of raster scan, some specific scanning methods like Morton scan and Peano-Hibert scan in Figure 6.-5 can take advantage on the wavelet structure. The chosen scanning is important consideration with respect to the embedded nature of the algorithm, and emphasizes the fact that, for images, there really is no well-defined notion of causality. Intuitively, is would seem that we should transmit the lower frequency components first.



Morton Scan Peano-Hibert Scan

Figure 6.-5 Different coefficient scanning method for a 8×8 image

Context selection is another small skill I use in my final implantation. I use different context for different part of the zerotree data structure. Different part in zerotree data structure is sensitive to different context. After the experiment, I find that significant bit can be modeled better by four pixels surround it and dominant and refinement are mainly relied on their parents.

	Lena ¤	Lena	Lena-G	EZW	state-of-art •
Bit Rate					
0.25	33.7855	33.7903	33.8059	33.17	34.57
0.5	36.8438	36.8523	36.8749	36.28	37.69
1	39.938	39.9384	39.9889	39.55	40.88

Table 6.-6 Coding result for "Lena"

¤ This image may be Lena-Y.

• Image coding based on mixture modeling of wavelet coefficients and a fast Estimation-Quantization framework.

I have done encoding on three images of "Lena" that I have with Peano-Hibert Scan, and the result is put at Table 6.-6. The Lena with best result is the Green part of color image, the worst one may be the luminance (Y) from the color image. Three coding results of Lena are shown out is because I am not sure which one is used by others. Some more testing results can be find in Table 6.-7 at Appendix D. Using the proposed context model, the context-base wavelet coding algorithm always leads EZW over 0.5 dB at the same bit rate.

However, proposal algorithm is still far from the state-of-art (about 0.7 dB). Beside the lacking of optimal quantization technique and advance filterbank, cost of model is another factor affect the compression performance. The higher-order modeling context would lead to a shorter codelength. But the number of possible conditions states grows exponentially with the order of context. Since the conditional probabilities $\Pr(s_i|s_1,\cdots,s_{i-1})$ have to be estimated on the fly by corresponding symbol histograms in different conditioning states, an image may not provide sufficient samples for the convergence of too many symbol histograms to $\Pr(s_i|s_1,\cdots,s_{i-1})$. In other words, a large modeling context spreads counting statistics to thin among all possible modeling states to reach good conditional probability estimates. The codelength will actually increase when the order of modeling contexts gets too high. Thus high order of context modeling is more than a problem of large time and space complexities, it can reduce coding efficiency as well. This problem is called "context dilution" and formulated by Rissanen analytically as so-called "model cost" [RIS84, RL81]. Solving this problem, we need to do more optimization on context and modeling.

6.5 Discussions

In above sections, I show that it is possible to use prior information to boost the performance of embedded zerotree wavelet compression algorithm. It is possible to predict the significant bit, dominant and refinement more accuracy by using more prior information. I presented a new context-based model for EZW image coding algorithm. It always leads the EZW image coding algorithm over 0.5dB. In the proposed scheme, the pixel can be estimated by more accurate with a context in a reasonable size. In addition, some techniques helping the compression rate are also introduced.

In the sequel, I can suggest some possible directions of further research for competition with the current champion in image compression. For the further improvement on the wavelet coding, there are various basic components in a wavelet-based image coder which can be focused on, namely, tree-structured filterbank, filters themselves, quantizers and entropy coders. Some filters can do a better job for image compression, but few modifications are need to arrange in the proposed algorithm for new filter. Estimation Quantization or Space-Frequency Quantization is a

better quantizer for the proposed algorithm. At the modeling part, wavelet coding with Set Partition In Hierarchical Tree is an alternative of the embedded zerotree. Context-driven algorithms that proposed in the lossless compression (Chapter 3, 4 and 5) should contribute in the image coding, although it is still under construction.

7. Conclusions

The objective of this thesis is to study context modeling and its application to the text and image data. For the text data, I introduce a new algorithm based on Context Sorting and Content Prediction. With the use of prior information, the system is try to learn and make prediction as human guess. More accurate guesses will reduce the redundancy of the data and make compression.

Image data is a very different kind of data from the text. The correlation in image is no longer one dimension as text, multi-dimension's correlation need another context modeling type. Moreover, image data can also accept the data loss during the compression. A new context-based compression method for image data is developed in this thesis.

The results and conclusions can be divided into three main groups : framework, modeling and coding. The proposed algorithms for text are covered by all these three groups. In the image part, I focused on the modeling area.

For the text data, I present a general-purpose text compression algorithm. The baseline version is a codebook based algorithm where the codebook is derived from a combination of sliding windowed context. The general framework for the Content Prediction is described, the possible construction of codebook and the context merging method are also presented. For better compression, symbol remapping methods are experimented with the Block Sorting algorithms. Besides, a static Content Prediction algorithm in a block basis is expressed as a real possible production for competition with the other best algorithms. During developing the Bounded-Length Block Sorting, adaptive frequency ordering and locally adaptive entropy coding are also investigated.

Wavelet is a relative new and promising development in the area of lossy image compression. I try to include context information into zerotree data structure for improvement on the image coding. Main area of image coding in this thesis concerns about the modeling of the context. Apart from the parent and previous pixel using by the Shapiro, I shift the views on the other potential pixels. Without strong dependence on the pixels suggested by Shapiro, I still can get a similar result as his. Some more pixels are proposed for the final implementation to get a good result. Moreover, a few more skills are suggested for a better performance.

7.1 Future Research

As sorting permutations can be handled from a mathematical point of view, this makes it possible to analyze or study the characteristics of the sorting, ranking and output distribution. Context Sorting may construct into a two dimension model for lossy coding or remodel to be a entropy coder at the final stage of image compression.

In the sequel, possible directions for future research are suggested, as a continuation of this work.

Content Prediction

A possible modeling on the Content Prediction is enlarger the prediction range, which is predicting more than one symbol in each guess. If average coded symbols at each prediction are more than one and the output distribution is not changing dramatic, we can get a better performance. For content prediction, ranking techniques (in Chapter 4.4) should be carefully selected under consideration for changing the accuracy rate of prediction.

Mixture sorting on both Context and Content

Bounded-Length Block Sorting has very different kind implementation rather than the baseline Content Prediction. Content prediction may not be used due to the prediction style is no longer operation adaptive. As both prior and follow information can do prediction on a symbol, a mixture that combined both Context and Content can use as context in Bounded-Length Block Sorting.

Extensions on basic components in wavelet-based image coder

Further improvement on the wavelet coding, there are various basic components in a wavelet-based image coder can focus on, namely, the tree-structured filterbank, the filters themselves, the quantizers and the entropy coders. Some filters can do a better job for image compression, but few modifications are need to arrange in the proposed algorithm for the new filter. Estimation Quantization or Space-Frequency Quantization is a better quantizer for the proposed algorithm. At the modeling part, wavelet coding with Set Partition In Hierarchical Tree is an alternative of the embedded zerotree. Context Sorting that proposed in the lossless compression (Chapter 3, 4 and 5) should contribute in the image coding, although it is still under construction.

Appendix - A

Lossless Compression results

Table 3.-1 Block Sorting with Alternatively Symbol Ordering Method

File	Size		Block Size 1K	5K	10K	50K	100K	200K
Text								
bib	111261	*L*	67550	48470	43015	33489	30486	28738
		F	65857	48949	43610	33849	30763	28930
		S	66782	48405	43012	33573	30533	28766
		G	65377	47938	42814	33462	30493	28755
		C	42259	39751	39348	39081	39013	39013
book1	768771	*L*	462644	372557	347926	302400	286737	273036
		F	439734	368052	345939	303097	287834	274205
		S	461125	371604	347260	302513	286867	273180
		G	448995	369197	346004	301984	286500	272927
		C	379067	361996	359728	357625	357339	357202
book2	610856	*L*	354585	268135	244732	204624	192021	182152
		F	337746	267463	245811	207051	194384	184236
		S	352493	268240	245035	205097	192388	182533
		G	343974	266821	244155	204576	192067	182282
		C	259218	245474	243803	242388	242172	242077
news	377109	*L*	242198	185789	171448	147550	138746	131465
		F	234529	186718	173332	149552	140673	133210
		S	240582	186168	172042	148055	139178	131817
		G	236945	184960	171116	147585	138913	131583
		C	179612	170238	168993	168139	168103	168014
paper1	53161	*L*	31625	23219	21300	17714	17165	17165
		F	30044	23198	21443	17943	17398	17398
		S	31483	23210	21305	17734	17182	17182
		G	30649	23121	21300	17768	17214	17214
		C	23704	22449	22301	22233	22233	22233
paper2	82199	*L*	47752	36492	33537	28361	26510	26510
		F	45226	36043	33348	28508	26632	26632
		S	47588	36464	33490	28363	26507	26507
		G	46569	36321	33425	28390	26525	26525
		C	37156	35365	35190	35142	35111	35111
paper3	46526	*L*	28077	21506	19823	16615	16615	16615
		F	26577	21336	19819	16713	16713	16713
		S	27867	21446	19807	16605	16605	16605
		G	27201	21350	19743	16584	16584	16584
		C	22649	21588	21472	21486	21486	21486
paper4	13286	*L*	7745	5994	5587	5256	5256	5256
		F	7413	5970	5608	5324	5324	5324
		S	7691	5985	5591	5268	5268	5268
		G	7504	5938	5558	5263	5263	5263
		C	6854	6638	6618	6589	6589	6589

paper5	11954	*L*	7021	5449	5126	4860	4860	4860
		F	6714	5469	5149	4913	4913	4913
		S	6961	5453	5140	4875	4875	4875
		G	6799	5406	5095	4842	4842	4842
		C	6383	6082	6101	6069	6069	6069
paper6	38105	*L*	21995	16222	14981	12616	12616	12616
		F	20965	16198	15057	12803	12803	12803
		S	21857	16244	14996	12664	12664	12664
		G	21289	16124	14919	12633	12633	12633
		C	17247	16356	16312	16278	16278	16278
progc	39611	*L*	22363	16665	14947	12794	12794	12794
		F	21834	16788	15166	13003	13003	13003
		S	22250	16609	14934	12793	12793	12793
		G	21941	16585	14930	12835	12835	12835
		C	17666	16622	16520	16617	16617	16617
progl	71646	*L*	33140	22153	19255	16849	16003	16003
		F	31873	22210	19466	17115	16261	16261
		S	33226	22101	19257	16885	16016	16016
		G	32178	21897	19138	16831	16006	16006
		C	22238	20679	20569	20500	20500	20500
progp	49379	*L*	23520	15405	13442	10959	10959	10959
		F	22313	15488	13511	11112	11112	11112
		S	23610	15426	13481	10965	10965	10965
		G	23013	15233	13316	10914	10914	10914
		C	15496	14257	14192	14137	14137	14137
trans	93695	*L*	50388	34242	28802	21194	18245	18245
		F	49164	34706	29347	21681	18643	18643
		S	50439	34353	28937	21302	18377	18377
		G	49657	34094	28740	21211	18255	18255
		C	25252	23215	23068	23125	23089	23089
Binary								
geo	102400	*L*	81793	71240	68545	63991	62578	62342
		F	79057	69793	67502	63710	62519	62266
		S	81943	71187	68679	64424	62992	62776
		G	82043	71484	68737	64111	62645	62403
		C	85048	75257	73403	71219	70896	70802
obj1	21504	*L*	13471	10938	10366	10516	10516	10516
		F	13774	11355	10771	10867	10867	10867
		S	13656	11025	10480	10588	10588	10588
		G	13539	10952	10386	10549	10549	10549
		C	14078	12862	12698	12686	12686	12686
obj2	246814	*L*	143312	100862	92374	83723	82503	81340
		F	145145	104063	95635	86702	85387	84230
		S	146074	102413	93763	84883	83581	82334
		G	144961	101347	92748	83941	82672	81493
		C	113503	104951	103560	103119	103809	104279
pic	513216	*L*	62177	53650	52710	52045	51970	51848
		F	59495	53549	52575	52205	52188	52092
		S	63397	54227	52867	52115	52046	51932
		G	63645	54176	53006	52214	52110	51958
		C	67468	64350	63948	63632	63694	63652

L Lexical ordering
F Empirical frequency ordering
S Pre-set frequency ordering based on natural language style
G Ordering refer a new symbol list that form by the symbol's properties
C High order symbol remapping

Table 4.-1 Compression Result of baseline Content Prediction Varies on Context Length

File	Size	Context Order				
		8	12	17	30	64
Text						
bib	111261	32443	32396	32402	32402	32402
book1	768771	287589	287562	287581	287582	287587
book2	610856	194017	193909	193908	193883	193880
news	377109	141485	141287	141261	141230	141285
paper1	53161	18053	18016	18012	18002	18004
paper2	82199	28123	28076	28068	28066	28069
progc	39611	13454	13435	13432	13440	13439
progl	71646	17245	17151	17115	17064	17038
progp	49379	11997	11742	11646	11548	11524
trans	93695	20158	19881	19790	19603	19562
Binary						
geo	102400	68747	68683	68683	68658	68657
obj1	21504	11518	11511	11480	11481	11481
obj2	246814	86424	86106	85958	85836	85832
pic	513216	54166	54249	54281	54285	54284

• The Block Size available for the testing is 64K.

Table 4.-2 Compression Result of baseline Content Prediction Varies on Block Size

File	Size	Block Size				
		4K	8K	16K	32K	64K
Text						
bib	111261	43174	39553	36308	33781	32402
book1	768771	341310	325308	310862	298403	287581
book2	610856	240858	225501	212562	202106	193908
news	377109	166317	159203	153044	147533	141261
paper1	53161	20730	19496	18643	18217	18012
paper2	82199	33032	31269	29899	28567	28068
progc	39611	15056	14079	13571	13428	13432
progl	71646	18843	17917	17529	17179	17115
progp	49379	12851	12181	12009	11821	11646
trans	93695	28430	24182	22187	20216	19790
Binary						
geo	102400	71809	70116	69484	69008	68683
obj1	21504	11493	11421	11474	11480	11480
obj2	246814	91823	89394	87740	86091	85958
pic	513216	54016	54087	54173	54240	54281

• The context Order sets to 17 for the tests.

Table 5.-1 Performance of Bounded-Length Block Sorting

File	Size		Context Order 1		2		3		4	
			left	right	left	right	left	right	left	right
Text										
bib	111261	*1*	57475	57792	45900	46031	37225	37331	33925	34103
		2	53599	53624	41244	41571	33437	33390	30568	30519
		3	56950	57441	45241	45401	36557	36754	33649	33827
		4	53065	53203	40545	40822	32750	32761	30320	30191
book1	768771	*1*	393857	392716	329739	330461	285516	285783	265694	265861
		2	392795	390769	319448	317380	271809	269820	250462	248800
		3	392373	391464	325337	326034	279497	280077	259435	259988
		4	390839	389169	314046	311723	264684	262629	243653	241705
book2	610856	*1*	306744	307131	243884	244349	201246	201035	184510	184092
		2	304674	304278	234565	234129	189477	188788	172109	170984
		3	305888	306287	240920	241212	197640	197374	181277	180932
		4	303698	303399	231183	230535	185799	184757	169086	167753
news	377109	*1*	205763	207159	166525	167476	142972	142943	134849	134446
		2	203538	204126	160995	161063	136456	135815	128051	127029
		3	205206	206601	165455	166357	142220	142180	134473	134156
		4	202851	203436	159746	159722	135432	134827	127384	126406
paper1	53161	*1*	27545	27624	22195	22201	19311	19306	18447	18402
		2	26735	26731	21113	21026	18339	18231	17545	17376
		3	27464	27538	21957	21964	19180	19173	18441	18399
		4	26643	26631	20871	20776	18214	18087	17564	17402
paper2	82199	*1*	42185	42185	34442	34487	29697	29788	28204	28301
		2	41212	41289	32851	32756	28066	27890	26661	26513
		3	42067	42068	33983	33999	29259	29399	27941	28067
		4	41027	41121	32317	32119	27619	27445	26417	26250
paper3	46526	*1*	24519	24465	20714	20640	18213	18150	17549	17425
		2	23898	23910	19795	19652	17323	17154	16692	16444
		3	24470	24414	20524	20437	18035	17984	17488	17379
		4	23820	23823	19569	19388	17126	16960	16636	16374
paper4	13286	*1*	7220	7182	6111	6084	5614	5595	5528	5495
		2	6980	7000	5839	5834	5348	5359	5258	5266
		3	7204	7164	6065	6041	5603	5584	5552	5520
		4	6962	6969	5777	5775	5333	5346	5278	5285
paper5	11954	*1*	6563	6560	5564	5551	5203	5184	5116	5096
		2	6345	6362	5345	5356	5000	4970	4914	4884
		3	6546	6540	5524	5508	5200	5200	5143	5132
		4	6324	6327	5304	5317	4992	4987	4936	4923
paper6	38105	*1*	19899	19751	16058	15945	14029	14000	13556	13494
		2	19087	19033	15272	15181	13347	13303	12912	12822
		3	19830	19685	15889	15776	13955	13968	13609	13562
		4	19001	18987	15090	15033	13277	13263	12976	12890
progc	39611	*1*	20149	20026	15704	15697	14302	14253	13869	13755
		2	19296	19360	14993	14984	13550	13453	13176	13127
		3	19986	19881	15543	15532	14206	14202	13937	13841
		4	19127	19235	14853	14899	13559	13547	13258	13236
progl	71646	*1*	31376	31369	22778	22771	19505	19497	18694	18657
		2	29323	29540	20819	20926	17680	17787	17028	17108
		3	31176	30987	22474	22407	19496	19471	18889	18871
		4	29236	29211	20499	20594	17794	17919	17299	17389

progp	49379	*1*	21644	21423	15559	15476	13318	13283	12885	12870
		2	20719	20579	14587	14384	12376	12212	11957	11790
		3	21395	21206	15295	15224	13345	13291	12991	12961
		4	20459	20341	14256	14184	12347	12273	12030	11966
trans	93695	*1*	43360	43658	31310	31375	24525	24400	22226	22043
		2	40157	40185	28174	28171	22354	22402	20430	20425
		3	42848	43163	30925	30960	24601	24505	22837	22601
		4	39860	39926	27989	27956	22528	22541	21032	20981
Binary										
geo	102400	*1*	64849	70073	64895	67423	64992	66638	64813	66289
		2	61676	67534	60125	63249	60063	61945	59888	61552
		3	64058	69742	64786	67201	64904	66512	64750	66221
		4	60420	66852	59976	62869	59958	61727	59772	61394
obj1	21504	*1*	13110	12831	11878	11596	11567	11463	11556	11512
		2	12546	12281	11508	11330	11251	11237	11237	11270
		3	12964	12700	11820	11551	11566	11476	11554	11520
		4	12471	12228	11454	11277	11229	11229	11221	11267
obj2	246814	*1*	114246	116067	95012	97023	89082	89855	87582	86706
		2	111293	112036	90959	92499	85324	86029	84116	82985
		3	113202	115385	94674	96681	89154	90233	88071	87719
		4	110345	111299	90711	92174	85481	86419	84704	84033
pic	513216	*1*	69971	68764	66703	65918	66010	65362	65581	64840
		2	65070	64603	63032	62574	62530	62086	61994	61502
		3	62047	60661	58866	57960	57988	57173	58933	58061
		4	57391	56852	55382	54845	54706	54176	55447	54921

File	Size		Context Order							
			5		6		7		8	
			left	right	left	right	left	right	left	right
Text										
bib	111261	*1*	32744	32827	32434	32507	32339	32347	32295	32251
		2	29470	29436	29265	29186	29210	29047	29177	28976
		3	32759	32791	32568	32588	32539	32488	32519	32423
		4	29510	29336	29407	29200	29414	29113	29406	29066
book1	768771	*1*	258619	258652	255797	255821	254709	254708	254270	254304
		2	243629	242036	241244	239564	240314	238588	240023	238281
		3	252854	252858	250282	250232	249440	249343	249180	248994
		4	237548	235364	235445	233181	234720	232407	234571	232165
book2	610856	*1*	178869	178565	176891	176775	176112	176022	175972	175876
		2	166766	165693	164993	164126	164383	163513	164294	163366
		3	176082	175755	174543	174300	174028	173615	174043	173518
		4	164244	162819	162879	161519	162430	160946	162436	160873
news	377109	*1*	132407	131949	131463	131064	131048	130717	130828	130493
		2	125678	124592	124757	123783	124331	123432	124119	123232
		3	132337	132109	131690	131414	131388	131190	131218	131025
		4	125365	124401	124736	123794	124444	123563	124287	123434
paper1	53161	*1*	18262	18184	18149	18103	18118	18055	18098	18025
		2	17370	17178	17277	17125	17245	17067	17224	17038
		3	18322	18245	18235	18211	18209	18191	18200	18173
		4	17454	17274	17394	17255	17366	17227	17355	17216
paper2	82199	*1*	27794	27787	27710	27661	27697	27645	27690	27614
		2	26307	26090	26233	26006	26219	25983	26234	25956
		3	27574	27643	27539	27569	27524	27565	27531	27540
		4	26121	25909	26095	25863	26083	25861	26103	25844

paper3	46526	*1*	17353	17268	17348	17274	17339	17269	17341	17251
		2	16502	16315	16510	16317	16502	16319	16502	16316
		3	17339	17276	17340	17280	17353	17282	17349	17274
		4	16503	16301	16507	16307	16522	16316	16510	16317
paper4	13286	*1*	5526	5491	5525	5497	5524	5502	5521	5503
		2	5251	5263	5257	5259	5257	5261	5255	5259
		3	5556	5527	5555	5535	5557	5538	5559	5541
		4	5277	5292	5280	5291	5281	5295	5281	5292
paper5	11954	*1*	5108	5067	5098	5058	5097	5064	5095	5065
		2	4907	4859	4895	4853	4891	4858	4891	4859
		3	5137	5121	5135	5117	5134	5116	5132	5118
		4	4933	4915	4931	4912	4930	4916	4930	4919
paper6	38105	*1*	13407	13364	13351	13300	13338	13275	13314	13260
		2	12791	12724	12748	12681	12730	12667	12711	12660
		3	13497	13476	13456	13438	13457	13416	13439	13406
		4	12888	12827	12865	12810	12863	12794	12856	12792
progc	39611	*1*	13756	13655	13728	13592	13687	13559	13677	13553
		2	13105	13066	13096	13009	13056	12985	13055	12979
		3	13869	13794	13851	13747	13833	13716	13829	13717
		4	13230	13216	13220	13179	13205	13165	13209	13158
progl	71646	*1*	18231	18225	17954	17947	17659	17649	17483	17456
		2	16645	16781	16440	16578	16265	16412	16132	16266
		3	18576	18527	18414	18385	18179	18186	18059	18050
		4	17036	17140	16900	17019	16740	16903	16658	16795
progp	49379	*1*	12679	12698	12444	12387	12359	12364	12222	12232
		2	11765	11610	11561	11441	11486	11403	11382	11298
		3	12861	12854	12757	12689	12679	12648	12604	12561
		4	11932	11881	11829	11749	11778	11731	11727	11658
trans	93695	*1*	21304	21083	20840	20661	20517	20338	20264	20182
		2	19659	19548	19243	19149	18959	18888	18821	18780
		3	22187	21985	21843	21625	21649	21415	21495	21363
		4	20543	20420	20256	20094	20102	19927	19991	19884
Binary										
geo	102400	*1*	64744	66239	64745	66221	64704	66203	64689	66182
		2	59818	61493	59805	61469	59764	61421	59723	61368
		3	64684	66178	64694	66194	64642	66162	64634	66163
		4	59706	61348	59736	61369	59675	61300	59653	61289
obj1	21504	*1*	11547	11529	11542	11542	11544	11547	11547	11553
		2	11230	11280	11218	11287	11221	11293	11220	11295
		3	11549	11529	11557	11546	11554	11543	11557	11545
		4	11221	11268	11223	11285	11219	11278	11219	11277
obj2	246814	*1*	85945	85377	85748	84946	85734	84770	85509	84439
		2	83005	82415	82917	82107	82869	82041	82709	81734
		3	86929	86712	86845	86388	86937	86296	86807	86052
		4	83903	83549	83904	83347	83942	83330	83879	83082
pic	513216	*1*	64639	63920	64405	63687	63920	63169	63284	62602
		2	60918	60474	60752	60360	60351	59930	59631	59294
		3	57529	56638	57289	56369	57841	56898	56960	56007
		4	53920	53420	53749	53234	54307	53800	53379	52856

1 Move-To-Front Coding and Arithmetic Coding
2 Move-To-Front Coding and Arithmetic Coding with 3-ary model
3 Move-To-Front Coding with adaptive frequency ordering & Arithmetic Coding
4 Move-To-Front Coding with adaptive frequency ordering & Arithmetic Coding with 3-ary model

Table 5.-2 Performance of Bounded-Length Block Sorting with using cache model

File	Size	Context Order 1	2	3	4	5	6	7	8
Text									
bib	111261	51483	39115	31852	29004	27863	27524	27354	27257
book1	768771	370748	299429	254006	235159	228693	226291	225449	225167
book2	610856	289027	221381	177931	161450	156450	154925	154271	154083
news	377109	195013	153093	129163	121017	118448	117540	117101	116903
paper1	53161	25407	20182	17707	16859	16649	16555	16497	16481
paper2	82199	39025	30992	26827	25419	24997	24873	24849	24835
paper3	46526	22650	18748	16657	15917	15730	15725	15729	15714
paper4	13286	6609	5656	5179	5072	5066	5068	5081	5077
paper5	11954	6021	5209	4873	4757	4723	4709	4713	4718
paper6	38105	18043	14611	12960	12450	12319	12277	12251	12233
progc	39611	18189	14382	12989	12580	12514	12461	12431	12427
progl	71646	27892	20202	17159	16380	16072	15826	15644	15512
progp	49379	19354	13900	11900	11428	11209	11054	11009	10933
trans	93695	38023	27118	21862	19719	18852	18442	18207	18085
Binary									
geo	102400	60527	57766	56921	56724	56647	56618	56595	56554
obj1	21504	11335	10361	10244	10277	10293	10304	10308	10315
obj2	246814	104114	85829	79534	76584	75745	75473	75361	74985
pic	513216	53902	53237	52912	52088	52050	52009	51438	51181

Table 5.-3 Comparison among the best lossless compression algorithms

File	Size	BS-BCL	GZIP	PPM*	PPMD+	PPMZ	BW95	BS-SM
Text								
bib	111261	1.96	2.51	1.91	1.86	1.771	2.02	1.95
book1	768771	2.34	3.25	2.40	2.30	2.235	2.48	2.39
book2	610856	2.02	2.70	2.02	1.96	1.887	2.10	2.04
news	377109	2.48	3.06	2.42	2.35	2.280	2.56	2.50
paper1	53161	2.48	2.79	2.37	2.33	2.263	2.52	2.46
paper2	82199	2.42	2.89	2.36	2.32	2.245	2.50	2.41
paper3	46526	2.70	3.11					
paper4	13286	3.05	3.33					
paper5	11954	3.15	3.34					
paper6	38105	2.57	2.77					
progc	39611	2.51	2.68	2.40	2.36	2.293	2.54	2.49
progl	71646	1.73	1.80	1.67	1.68	1.505	1.75	1.72
progp	49379	1.77	1.81	1.62	1.70	1.549	1.74	1.70
trans	93695	1.54	1.61	1.45	1.47	1.273	1.52	1.50
Binary								
geo	102400	4.42	5.34	4.83	4.73	4.476	4.73	4.50
obj1	21504	3.81	3.84	4.00	3.73	3.712	3.88	3.87
obj2	246814	2.43	2.63	2.43	2.38	2.287	2.53	2.46
pic	513216	0.80	0.80	0.85	0.80	0.770	0.79	0.77

BS-BCL Block Sorting with a Bounded Context Length in this thesis
PPM* a recently published unbounded context version of PPM
PPMD+ a further-improved version of PPM
PPMZ a mixture of several PPM variance by Bloom C.
BW95 Block Sorting + run-length coding + historical context from recent codings
BS-SM Block Sorting with structured coding model by Fenwick P.

Appendix - B

Image Compression Standards

In the past years, a number of compression standards have emerged and a number is now being developed. Although it would be convenience to use only one general image compression standard, a growing number of standards are developed because of enhanced processing power, dedicated hardware, new compression techniques, and networks with different bandwidths.

Each compression standard supports a specific image application. It is difficult to choose the correct compression standard for a specific application. As is true of compression in general that there does not exist one best compression algorithm, the same is true of image compression: there is no best standard. Some applications require fast real-time encoding, at the cost of the compression factor, while other applications want maximum compression at encoding that need not be done real-time, as long as decoding is real-time.

Compression algorithm can be categorized in two groups: lossless and lossy compression. Lossless algorithms generate exactly the same bit pattern of an object after decompression as before the object was compressed. These compression algorithms are used for text and computer binary files. Lossy compression algorithms, however, may lose some information during compression. In a good lossy compression algorithm, the lost information is not visible in case of a picture. Most lossy compression algorithms have the ability to specify a quality-setting that determines how much quality (information) may be lost for a higher compression ratio.

Lossy compression algorithms are useful for compression of samples data. This data is analog data from a microphone or a camera that is converted to a digital approximation. Therefore, lossy compression algorithms that change the data slightly are not catastrophic. Lossy compression followed by decompression, however, cause quality loss that can better be avoided by reducing the number of compression-decompression operations for a picture. If a picture must be manipulated (in the image space), it can best be store as raw data between the image operations.

DCT transformation

A transformation that is useful in image compression is the DCT. This transformation converts a $n \times n$ block of elements into another block of $n \times n$ coefficients. These $n \times n$

coefficients represent two-dimensional unique spatial frequencies. The DCT function is reversible by using an IDCT function.

The first coefficients, which has a zero horizontal and vertical frequency, is called the DC-coefficient ad is equal to the average value of the original elements. The other coefficients are called AC-coefficients and represent the dimensional spatial frequencies.

The DCT and IDCT are lossless if the DCT encoded data are stored with perfect accuracy. In practice, however, the coefficients are stored as integers which can introduce small differences with the original data after the IDCT decoding.

If the DCT transformation is applied to blocks of pixels, higher spatial frequency coefficients become (near) zero because most pixels next to each other differ little in value. If relative more bits are used to encode the lower frequency coefficients than the higher frequency coefficients, a (lossy) compression method is created.

Fractal compression

Fractal compression is one of the latest techniques in lossy image compression. Fractals are images that recursively contain themselves. They are defined by a number of translations that include re-scales, rotations and dimensional flips. If you zoom into a fractal image, it appears that the image has an infinite resolution, but it is actually a part of the same image that reappears in itself. The idea behind fractal compression is to automatically find a fractal that resembles the image that must be compressed. A mayor advantage of fractal compression is the ability to decompress the image to any given resolution. The first implementation of such an algorithm was implemented by Arnaud Jacquin and was capable of compression from 8:1 to 50:1 while remaining reasonable quality. This implementation searches a combination of transformations that represent the image the best. Unfortunately, the search to find this transformation is very computationally intensive.

Wavelet compression

A relative new and promising development in the area of lossy compression is the use of wavelet transformation. An important characteristic of this transformation is that if it is applied on a time-domain signal, it results in a representation that is localized in time domain as well as in frequency domain. Compared to the Fast Fourier Transform (FFT) that is of an order of $N \times \log_2(N)$ for N elements, a fast wavelet transform has an order of N for the same number of elements.

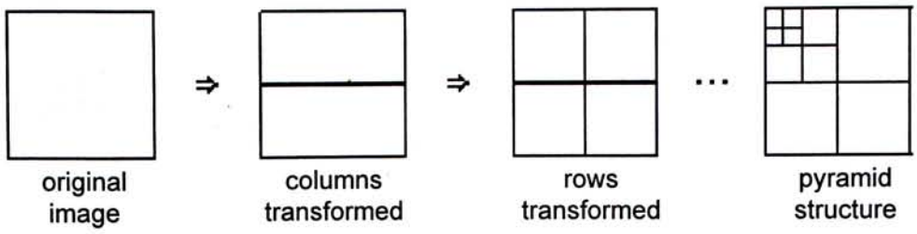


Figure B.-1 Construction of image multi-resolution pyramid from one dimensional transformation

The wavelet transformation converts a sample of 2^j value into 2^{j-1} approximation wavelet transform coefficients and 2^{j-1} detail wavelet transform coefficients. This transformation can be repeated over the generated approximation wavelet transform coefficients a number of times, until the minimum number of 2 approximation transform coefficients and 2^{j-2} detail transform coefficients remain. A flow of wavelet transformation is shown in Figure B.-1. The number of transformation is called the number of levels of the wavelet transformation. The wavelet transformation is reverse, so applying the reverse wavelet transformation a number of times (equal to the number of levels) on the generated wavelet coefficients, the original sample is recomposed.

Wavelet compression is obtained by only storing those coefficients of the wavelet transformation that have an amplitude above a certain threshold together with the place of those coefficients in the transformed domain. Because the coefficients are also time-domain, high contrast edges are maintained at the cost of low contrast areas. By using quantization and entropy encoding in combination with wavelet transform the number of bits needed to store the wavelet coefficients are further reduced.

Appendix - C

Human Visual System Characteristics

Contrast

The HVS is less sensitive to small changes in intensity at high average intensities, than at low.

Contrast sensitivity function

The CSF provides a relationship between contrast sensitivity and spatial frequency. It shows that the HVS is relatively insensitive to low power stimulus at low frequencies. The HVS is most sensitive at midrange frequencies (about 4 cycles/degree).

Spatial masking

Spatial masking is the reduced visibility of a stimulus which occurs when the stimulus is in the close vicinity of a large change in background luminance.

Texture masking

Texture masking is the reduced visibility of a stimulus which occurs when the stimulus is in a textured region -> the HVS is less sensitive to variations in the true image signal in high activity regions, than in low activity (predominantly homogeneous) regions.

In addition to these characteristics, the HVS also gains a lot of visual information from strong edges. Thus, enhancing the edges in images can often make the image a lot easier to identify. Conversely, image compression techniques which blur or corrupt edges make the image harder to recognize.

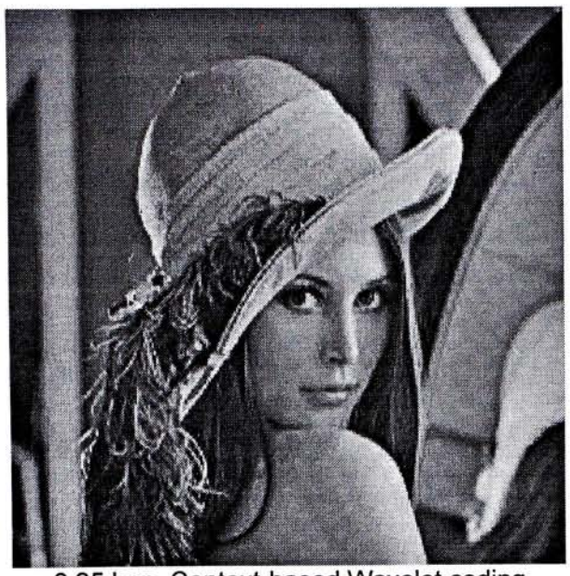
Appendix - D

Lossy Compression Results

Table 6.-7 Coding results of some common testing files

Image Files	Context-based Wavelet coding Bit Rate			RD-OPT-based JPEG coding Bit Rate			SPIHT Bit Rate		
	0.25	0.5	1	0.25	0.5	1	0.25	0.5	1
baboon	23.2133	25.1489	28.5712	22.7731	25.0552	28.5025	23.27	25.65	29.18
barbara	27.6113	31.7366	36.8681	27.2621	30.9965	36.1806	28.13	32.11	37.45
boat	31.0326	34.6070	39.2015	29.6493	33.3515	37.9509	30.97	34.45	39.12
bridge	24.8725	27.3441	30.3274	24.5500	26.8181	30.1307	24.96	27.27	30.7
couple	29.2384	32.1580	36.2334	28.3905	31.6366	35.7161	29.25	32.45	36.58
crowd	30.1120	33.6093	38.7034	28.7494	32.5156	37.3249	30.15	33.89	38.86
flower	38.4927	42.5803	46.0657	35.4212	40.5217	45.3400	38.55	42.90	46.32
girl	33.8983	37.3965	41.3543	32.5401	36.5000	40.9206	34.05	37.58	41.6
goldhill	30.4417	33.1257	36.2769	29.6544	32.3899	35.9370	30.56	33.13	36.55
hustler	37.7418	41.1494	44.7978	35.4605	39.7069	44.3021	37.75	41.21	44.87
lake	28.6411	31.3192	34.3927	27.5493	30.6384	34.0602	28.68	31.57	34.82
lena	33.8059	36.8749	39.9889	31.9067	35.5014	39.2184	34.15	37.25	40.46
man	29.9670	32.7883	36.8225	28.8248	31.8240	35.8354	30.01	33.08	37.34
marie	38.3774	42.0622	45.3402	35.4549	40.2445	44.7026	38.38	42.29	45.54
peppers	33.3738	35.6458	38.1847	31.4205	34.4931	37.5288	33.53	35.84	38.39
plane	32.5410	36.4464	40.7925	30.6890	34.7888	39.5474	32.64	36.65	41.23
tiffany	31.0022	33.8792	37.0284	29.3946	32.3195	35.9604	31.11	33.98	37.41
woman	40.0107	42.2931	45.3713	37.6965	41.1323	44.6656	40.20	42.54	45.75
zelda	37.1661	39.4859	42.1429	35.1151	38.3670	41.5707	37.43	39.59	42.08

Context-based Wavelet coding	The context-based image compression algorithm that proposed in this thesis.
RD-OPT-based JPEG compression	Optimizing the DCT quantization tables in an image-specific manner, rather than uniform scalar quantization in normal JPEG. It is the best result for DCT-based compression algorithm [RL95].
SPIHT	Wavelet image Compression with Set Partitioning in Hierarchical Trees. It is the state-of-art now.



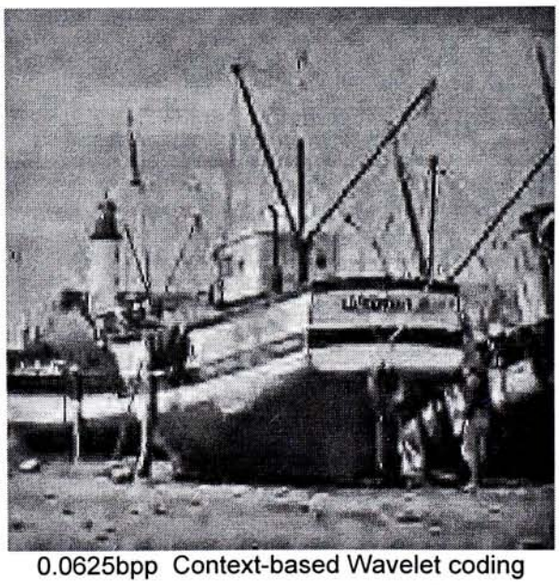
0.25 bpp Context-based Wavelet coding

0.25 bpp Context-based Wavelet coding

0.125 bpp Context-based Wavelet coding

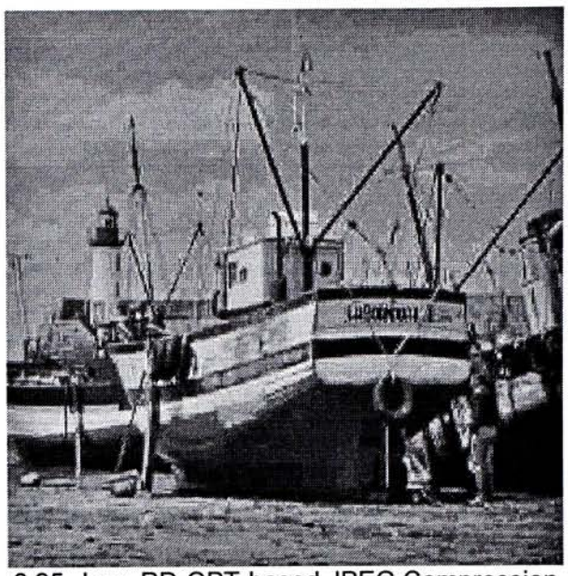
0.125bpp Context-based Wavelet coding

0.0625 bpp Context-based Wavelet coding



0.0625bpp Context-based Wavelet coding

0.25 bpp RD-OPT-based JPEG Compression



0.25 bpp RD-OPT-based JPEG Compression

0.125 bpp RD-OPT-based JPEG Compression

0.125 bpp RD-OPT-based JPEG Compression

0.0625 bpp RD-OPT-based JPEG Compression

0.625 bpp RD-OPT-based JPEG Compression

0.25 bpp SPIHT Wavelet Compression

0.25 bpp SPIHT Wavelet Compression

0.125 bpp SPIHT Wavelet Compression

0.125 bpp SPIHT Wavelet Compression

0.0625 bpp SPIHT Wavelet Compression

0.0625 bpp SPIHT Wavelet Compression

References

[ABMD92] M. ANTONINI, M. BARLAUD, P. MATHIEU, I. DAUBECHIE, "Image coding using wavelet transform", *IEEE Trans. on Image Proc., vol. 1, pp. 205-220, 1992.*

[AKMK92] V. R. ALGAZI, Y. KATO, M. MIYAHARA, K. KOTANI, "Comparison of image coding techniques with a Picture Quality Scale", *Proc. of SPIE, Applications of Digital Image Processing XV, vol. 1771, pp. 396-405, 1992.*

[AM97] Z. ARNAVUT, S. S. MAGLIVERAS, "Block Sorting and Compression", *Data Compression Conference, DCC97.*

[AND77] J. H. ANDREAE, "Thinking with teachable machine", *London: Academic Press (1977).*

[BCW90] T. C. BELL, J. G. CLEARY, I. H. WITTEN, "Text Compression", *Prentice Hall, New Jersey, 1990.*

[BLO96] C. BLOOM, "LZP : a new data compression algorithm", *Data Compression Conference, 1996.*

[BM89] T. BELL, A. M. MOFFAT, "A Note on the DMC Data Compression Scheme", *Computer Journal, vol. 32 (1989), pp. 16-20.*

[BMH92] J. N. BRADLEY, C. M. BRISLAWN, T. HOPPER, "The FBI wavelet / scalar quantization standard for grayscale fingerprint image compression", *Proc. of SPIE, Visual Info. Proc. II, vol. 1961, pp. 293-304, 1992.*

[BSTW86] J. L. BENTLEY, D. D. SLEATOR, R. E. TARJAN, V. K. WEI, "A locally adaptive data compression algorithm", *Communications of the ACM, vol. 29, no. 4, Apr. 1986, pp. 320-330.*

[BW94] M. BURROWS, D. J. WHEELER, "A Block-sorting Lossless Data Compression Algorithm", *Digital Systems Research Center Research Report 124, May 1994.*

[CTW95] J. G. CLEARY, W. J. TEAHAN, I. H. WITTEN, "Unbounded Length Contexts for PPM", *Data Compression Conference, DCC95.*

[CW84] J. G. CLEARY, I. H. WITTEN, "Data Compression Using Adaptive Coding and Partial String Matching", *IEEE Trans. Comm., COM-32 (Apr. 1984), pp. 396-402.*

[dQR92] R. L. de Queiroz, K. R. Rao, "Human visual sensitivity-weighted progressive image transmission using the lapped orthogonal transform", *Journal of Electronic Imaging, 1 (1992), pp. 328-338.*

[ELI75] P. ELIAS, "Universal Codewords Sets and Representations of Integers", *IEEE Transactions on Information Theory, 21 (1975), pp. 194-203.*

[FEN94] P. FENWICK, "A new data structure for cumulative probability tables", *Software - Practice and Experience, 24 (3) (1994), pp. 327-336.*

[FEN95A] P. FENWICK, "Experiments with a Block-Sorting Text Compression Algorithm", *University of Auckland, Department of Computer Science, Technical Report 111, March 1995.*

[FEN95B] P. FENWICK, "Improvements to the Block-Sorting Text Compression Algorithm", *University of Auckland, Department of Computer Science, Technical Report 120, July 1995.*

[FEN95C] P. FENWICK, "Block Sorting Text Compression — Final Report", University of Auckland, Department of Computer Science, Technical Report 130, April 1996.

[FEN96] P. FENWICK, "Block Sorting Text Compression", *Australasian Computer Science Conference, ACSC96, Melbourne, Australia.*

[GOL66] S. W. GOLOMB, "Run-Length Encodings", *IEEE Trans. Inform. Theory, IT-12 (July 1966), pp. 399-401.*

[HUF52] D. A. HUFFMAN, "A Method for the Construction of Minimum Redundancy Codes", *Proceedings of the Institute of Radio Engineers, 40 (1952), pp. 1098-1101.*

[HV93] P. G. HOWARD, J. S. VITTER, "Design and Analysis of Fast Text Compression Based on Quasi-Arithmetic Coding", *Data Compression Conference, pp. 98-107, DCC93.*

[JCF95] R. L. JOSHI, V. J. CRUMP, T. R. FISHER, "Image Subband Coding Using Arithmetic Coded Trellis Coded Quantization", *IEEE Trans. Circuits and Systems for Video Technology, 5(6) (1995), pp. 515-523.*

[KM95] L. KE, M. W. MARCELLIN, "Near-lossless image compression : minimum-entropy, constrained-error DPCM", *IEEE International Conference on Image Processing, Washington DC, Oct. 1995.*

[LH91] D. A. LELEWER, D. S. HIRSCHBERG, "Streamlining Context Models for Data Compression", *Data Compression Conference, pp.313-322, DCC91.*

[LRO97] S. M. LOPRESTO, K. RAMCHANDRAN, M. T. ORCHARD, "Image coding based on mixture modeling of wavelet coefficients and a fast estimation-quantization framework", *Data Compression Conference 97, DCC97, pp. 221-230.*

[LZ95] W. LI, Y. Q. ZHANG, "Vector-based signal processing and quantization for image and video compression", *Proc. of IEEE, vol. 83, no. 2, pp. 317-335, Feb. 1995.*

[MAC92] B. MACQ, "Weighted optimum bit allocations to orthogonal transforms for picture coding", *IEEE Journal on Selected Areas in Communications, 10 (1992), pp. 875-883.*

[MNW95] A. MOFFAT, R. NEAL, I. H. WITTEN, "Arithmetic Coding Revisited", *Data Compression Conference, DCC95.*

[MOF90] A. M. MOFFAT, "Implementing the PPM Data Compression Scheme", *IEEE Trans. Comm., COM-38 (Nov. 1990), pp. 1917-1921.*

[NEL96] M. NELSON, "Data Compression with the Burrows-Wheeler Transform", *Dr. Dobb's Journal, Sep. 1996.*

[NEV96] C. G. NEVILL-MANNING, "Inferring Sequential Structure", *Doctoral thesis, University of Waikato, Hamilton, New Zealand.*

[PMLA88] W. B. PENNEBAKER, J. L. MITCHELL, G. G. LANDON, JR. AND R. B. ARPS, "An overview of the basic principles of the Q-coder adaptive binary arithmetic coder", *IBM J. Research and Development, vol. 32, pp. 717-726, Nov. 1988.*

[PM93] W. B. PENNEBAKER, J. L. MITCHELL, "JPEG Still Image Data Compression Standard", *Van Nostrand Reinhold, New York, 1993.*

[RIS84] J. RISSANEN, "Universal coding, information, prediction, and estimation", *IEEE Trans. Inform. Theory, vol. 30 (July 1984), pp. 629-636.*

[RL81] J. RISSANEN, G. G. LANGDON, "Universal Modeling and Coding", *IEEE Trans. Inform. Theory, IT-27 (Jan. 1981), pp. 12-23.*

[RL95] V. RATNAKAR, M. LIVNY, "RD-OPT : An efficient algorithm for optimizing DCT quantization tables", *Data Compression Conference, 1995.*

[SHA48] C. E. SHANNON, "A Mathematical Theory of Communication", *Bell System Technical Journal, vol.27 (July 1948), pp. 398-403.*

[SHA51] C. E. SHANNON, "Prediction and Entropy of Printed English", *Bell System Technical Journal, vol. 30 (Jan 1951), pp. 50-64.*

[SHA93] J. M. SHAPIRO, "Embedded image coding using zerotrees of wavelet coefficients", *IEEE Trans. Signal Processing, vol. 41, no. 12 (Dec. 1993), pp. 3445-3462.*

[SP96] A. SAID, W. A. PEARLMAN, "A new fast and efficient image codec based on set partitioning in hierarchical trees", *IEEE Trans. Circuits and Systems for Video Tech., vol.6, no. 3 (June 1996), pp. 243-249.*

[TEA95] W. J. TEAHAN, "Probability estimation for PPM", *New Zealand Computer Science Research Students Conference Proceedings, University of Waikato, Hamilton, NZ, pp. 267-274, 1995.*

[TUN68] B. P. TUNSTALL, "Synthesis of Noiseless Compression Codes", *Ph.D. thesis, Georgia Inst. Technol., Atlanta, 1968.*

[TVC96] M. J. TSAI, J. VILLASENOR, F. CHEN, "Stack-run image coding", *IEEE Transactions on Circuits and Systems for Video technology, vol. 6 (Oct. 1996), pp. 519-521.*

[USE96] B. USEVITCH, "Optimal Bit Allocation for Biorthogonal Wavelet Coding", *Data Compression Conference, pp. 387-395, DCC96.*

[VBL95] J. VILLASENOR, B. BELZER, J. LIAO, "Wavelet Filter Evaluation for Image Compression", *IEEE Trans. Image Processing, 2 (1995), pp.1053-1060.*

[VOL96] P. A. J. VOLF, "Text Compression Methods Based on Context Weighting", *Ph. D. thesis, Eindhoven Univ. of Technology (1996).*

[WB] I. H. WITTEN, T. BELL, "The Calgary/Canterbury text compression corpus", *Anonymous ftp from ftp.cpsc.ucalgary.ca : /pub/text.compression/corpus/text.compression.corpus.tar.Z.*

[WB91] I. H. WITTEN, T. C. BELL, "The Zero Frequency Problem : Estimating the Probabilities of Novel Events in Adaptive Text Compression", *IEEE Trans. Inform. Theory, IT-37 (July 1991), pp. 1085-1094.*

[WEL84] T. A. WELCH, "A technique for high-performance data compression", *IEEE Computer, vol. 17, no. 6 (June 1984), pp. 8--19.*

[WMB94] I. H. WITTEN, A. MOFFAT, T. C. BELL, "Managing Gigabyte : compressing and indexing documents and images", *New York : Van Nostrand Reinhold.*

[WMS95] X. WU, N. MEMON, K. SAYOOD, "A context-based, adaptive, lossless / nearly-lossless coding scheme for continuous-tone images", *A proposal submitted in response to the Call for Contributions for ISO/IEC JTC 1.29.12, 1995.*

[WNC87] I. H. WITTEN, R. NEAL, J. G. CLEARY, "Arithmetic Coding for Data Compression ", *Comm., ACM-30, 6 (Jun. 1987), pp. 520-540.*

[WRA96] M. J. WEINBERGER, J. J. RISSANEN, R. B. ARPS, "Applications of universal context modeling to lossless compression of grayscale images", *IEEE Trans. Image Processing, 5(4) (1996), pp. 1053-1060.*

[WST95] FRANS M.J. WILLEMS, YURI M. SHTARKOV, TJALLING J. TJALKENS, "The Context Tree Weighting Method: Basic Properties," *IEEE Trans. Inform. Theory, May 1995.*

[WST96] FRANS M.J. WILLEMS, YURI M. SHTARKOV, AND TJALLING J. TJALKENS, "Context Weighting for General Finite-Context Sources," *IEEE Trans. Inform. Theory, Sept. 1996.*

[WWY97] L. WONG, VICTOR K. WEI, R. W. YEUNG, "Two Results in Text Compression Algorithms", *Int'l Symposium on Information Theory 1997.*

[XGO96] Z. XIONG, O. GULERYUZ, M. T. ORCHARD, "A DCT-based Embedded Image Coder", *IEEE Signal Processing Letters, vol. 3 (Nov. 1996), pp. 289-290.*

[XRO93] Z. XIONG, K. RAMCHANDRAN, M. T. ORCHARD, "Joint optimization of scalar and tree-structured quantization of wavelet image decompositions", *Proc. 27th Annual Asilomar Conf. on Sig., Syst., and Comp., Pacific Grove, CA, Nov. 1993.*

[XRO96] Z. XIONG, K. RAMCHANDRAN, M. T. ORCHARD, "Wavelet packets image coding using space-frequency quantization", *submitted to IEEE Trans. Image Processing, January 1996.*

[XRO97] Z. XIONG, K. RAMCHANDRAN, M. T. ORCHARD, "Space-frequency quantization for wavelet image coding", *to appear in IEEE Trans. Image Processing, 1997.*

[YOK96] H. YOKOO, "An Adaptive Data Compression Method Based on Context Sorting", *Data Compression Conference, DCC96.*

[ZASB95] A. ZANDI, J. ALLEN, E. SCHWARTZ, M.BOLIEK, "CREW : Compression with reversible embedded wavelets", *Data Compression Conference, pp.212-221, DCC95.*

[ZL77] J. ZIV, A. LEMPEL, "A Universal Algorithm for Sequential Data Compression", *IEEE Trans. Inform. Theory, IT-23 (May 1977), pp. 337-343.*

[ZL78] J. ZIV, A. LEMPEL, "Compression of Individual Sequences via Variable Rate Coding", *IEEE Trans. Inform. Theory, IT-24 (Sept. 1978), pp. 530-536.*