

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

Group Testing for Image Compression

Edwin S. Hong

**A dissertation submitted in partial fulfillment
of the requirements for the degree of**

Doctor of Philosophy

University of Washington

2001

Program Authorized to Offer Degree: Computer Science and Engineering

UMI Number: 3036484

UMI[®]

UMI Microform 3036484

Copyright 2002 by ProQuest Information and Learning Company.

**All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.**

**ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346**

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Bell and Howell Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature Shw. Ag

Date 9/28/01

University of Washington

Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

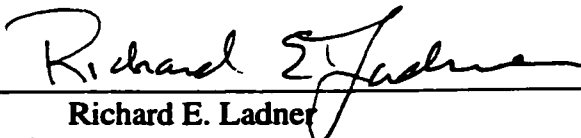
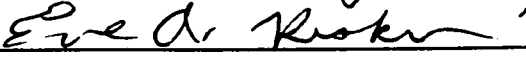
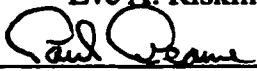
Edwin S. Hong

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of Supervisory Committee:


Richard E. Ladner

Reading Committee:


Richard E. Ladner

Eve A. Riskin

Paul Beame

Date:

9/28/01

University of Washington

Abstract

Group Testing for Image Compression

by Edwin S. Hong

Chair of Supervisory Committee:

**Professor Richard E. Ladner
Department of Computer Science and Engineering**

This thesis studies the application of group testing to image compression. Group testing is a technique used for identifying a few significant items out of a large set. Image compression studies techniques for making image data take up less storage space. We first explain the many interesting and deep connections between image compression and group testing, and then demonstrate the effectiveness of group testing techniques for image compression, from both practical and theoretical points of view.

In particular, we show that group testing is a generalization of the zerotree coding method widely used in wavelet-based image compression. We also show the equivalence of elementary Golomb codes and the binary splitting procedure used in Hwang's generalized group testing method. Next, we present new image coding techniques based on transform coding which apply group testing to the output of different transforms. We present one new image coder for each type of transform we study, namely: the wavelet transform; the wavelet packet transform; and block transforms such as the discrete cosine transform and lapped transforms. Group testing's flexibility and usefulness is shown in its applicability to many different transforms. In terms of compression performance, these new algorithms are competitive with many recent state-of-the-art image coders that use the same transforms on

a wide variety of images.

We also present a study on the theoretical performance of group testing on correlated Markov sources. We show how images can be modeled by these Markov sources, and relate these theoretical performance results to the performance of group testing on image compression.

TABLE OF CONTENTS

List of Figures	vii
List of Tables	ix
Chapter 1: Introduction	1
Chapter 2: Image Compression Background	4
2.1 Image Compression Problem	4
2.1.1 Digital Images and Quantization	4
2.2 Main Image Compression Goal	5
2.2.1 Measures of Image Quality	5
2.2.2 Rate-Distortion Curves	6
2.3 Other Image Compression Goals	9
2.3.1 Resource Usage	9
2.3.2 Training	9
2.3.3 Embeddedness	9
2.3.4 Resolution Scalability	10
2.3.5 Spatial Accessibility	11
2.3.6 Identifiable Bitstream	11
2.3.7 Entropy and Image Compression	12
2.4 Image Coding Techniques	12
2.4.1 Transform Coding	13
2.4.2 Wavelet Transform	15

2.4.3	Bit-plane Encoding	21
2.4.4	Significance and Refinement Passes	23
2.4.5	Fractional Bit-planes	23
2.4.6	Zerotrees	24
2.4.7	Entropy Coding	25
2.5	Statistical Characteristics of Images	26
2.6	Recent Wavelet-based Image Compression Algorithms	28
2.6.1	EZW	28
2.6.2	SPIHT	29
2.6.3	ECECOW	30
2.6.4	Ordentlich <i>et al.</i> 's Algorithm	31
2.6.5	PWC	32
2.6.6	EBCOT and JPEG-2000	33
Chapter 3:	Entropy Coding and Information Theory	35
3.1	Introduction to Entropy Codes	35
3.2	Introduction to Information Theory	36
3.2.1	Entropy of Sources	37
3.3	Examples of Entropy Codes	38
3.3.1	Simple Example	38
3.3.2	Block Codes	39
3.3.3	Arithmetic Coding	40
3.3.4	Golomb Codes	41
Chapter 4:	Group Testing	45
4.1	Group Testing Problem	45
4.1.1	Problem Definition	45

4.1.2	Issues	46
4.1.3	Applicability	47
4.2	Group Testing Algorithms	47
4.2.1	Hwang's Generalized Binary Splitting Algorithm	47
4.3	Other Group Testing Algorithms	53
4.3.1	Doubling	53
4.3.2	Jumping	54
Chapter 5:	Group Testing and Compression	55
5.1	Group Testing and Entropy Coding	55
5.1.1	Relation to Golomb Codes	56
5.1.2	Effectiveness of Group Testing	56
5.1.3	Exact Partitioning Method	57
5.2	Group Testing and Zerotree Coding	61
5.3	Group Testing and Image Compression	62
5.3.1	Previous Work	62
5.3.2	Mismatch Between Source Models	62
5.3.3	Making the Sources Match	63
5.3.4	Defining New Group Testing Algorithms	64
Chapter 6:	Group Testing for Wavelets	68
6.1	The GTW Algorithm	69
6.1.1	Algorithm Overview	69
6.1.2	GTW Classes	69
6.1.3	Class Ordering	72
6.1.4	GTW Details	75
6.2	Results	79

6.3	Alternative Approaches	80
6.3.1	Alternative Significant Neighbor Metric	80
6.3.2	Alternative Pattern Types	83
6.3.3	Varying Ordering of Testing Groups	84
6.3.4	GTW with Arithmetic Coding	86
6.3.5	Varying Group Testing Strategies	87
6.4	Conclusions	88
6.4.1	Comparison with Wavelet Image Coders	88
6.4.2	Summary of Results	89
Chapter 7:	Group Testing for Wavelet Packets	90
7.1	Introduction	90
7.2	Wavelet Packets Background	91
7.3	Group Testing for Wavelet Packets	93
7.3.1	Best Basis	93
7.3.2	GTWP Classes	95
7.4	Results	98
7.5	Conclusion	101
Chapter 8:	Group Testing for Block Transforms	103
8.1	Block Transform Background	104
8.1.1	Discrete Cosine Transform	104
8.1.2	Lapped Transforms	106
8.1.3	Organization into Subbands	106
8.1.4	Relation to the Wavelet Transform	108
8.1.5	Previous Algorithms	109
8.2	Group Testing for Block Transforms	111

8.2.1	GTBT Classes	112
8.3	Results	114
8.4	Conclusion	118
Chapter 9:	Theoretical Performance of Group Testing	119
9.1	Modeling Image Transform Coefficients	120
9.1.1	Bit Characteristics of Transform Coefficients	121
9.2	Independence Model	122
9.2.1	Encoding Independence Model	122
9.2.2	Group Testing Performance on Independence Model	123
9.2.3	Relation to GTW	123
9.2.4	Group Testing Bit-rate for i.i.d. Sources	124
9.3	Modeling Dependencies	126
9.3.1	Choosing Dependencies to Model	126
9.3.2	Dependence Model	127
9.3.3	Relation to Transform Coefficient Bits	128
9.4	Two-state Markov Chain Model	131
9.4.1	Two-state Markov Chain Source	132
9.4.2	Entropy of Markov Source	133
9.4.3	Simple Sequential Coding	133
9.4.4	Bit-rate of Simple Sequential Coding	134
9.4.5	Two-state Sequential Coding	137
9.4.6	Bit-rate of Two-state Sequential Coding	138
9.4.7	Interleaved Coding	140
9.4.8	Bit-rate of Interleaved Coding	141
9.4.9	Comparison of Methods	143
9.4.10	Alternative Method for Calculating Bit-rates	149

9.5 Conclusion	151
Chapter 10: Conclusions	153
Bibliography	155

LIST OF FIGURES

2.1	Barbara at different fidelities	7
2.2	Rate-distortion curve	8
2.3	Transform coding diagram	13
2.4	The Haar wavelet and Daubechies' biorthogonal wavelet	16
2.5	1D DWT	19
2.6	One level of the 2D DWT	20
2.7	2D DWT	21
2.8	Parent-child relationships in the DWT	24
4.1	Group testing example	52
5.1	Comparing group testing bit-rate with the entropy on an i.i.d. source	57
5.2	Group testing code as a binary tree	60
5.3	Diagram of Markov source	65
6.1	GTW subband levels	70
6.2	GTW neighbors	71
6.3	GTW pattern type	72
6.4	Bootstrapping	73
6.5	Group iteration size scatter plot	74
6.6	Comparing GTW and SPIHT	79
6.7	Significance results of logit metric value	82
6.8	Significance results of <i>ad hoc</i> significant neighbor count	82

6.9	Information propagation in patterns	83
6.10	Performance of different class orderings	85
6.11	Performance of generalized coefficient orderings	87
7.1	Subbands with wavelet packets	92
7.2	Best basis for Barbara	95
7.3	GTWP-S subband levels	96
7.4	GTWP-J orientation type	97
7.5	Comparing GTW, GTWP, and JPEG-2000	99
8.1	Block transform coefficient reorganization	107
8.2	GTBT subband levels	113
8.3	Comparing GT-GLBT with others	117
9.1	Comparing various group testing methods on $S_{p,0.5}$	144
9.2	Comparing group testing to entropy on $S_{p,0.5}$	145
9.3	Efficacy of group testing on $S_{p,0.5}$	146
9.4	Comparing various group testing methods on $S_{p,0.4}$	147
9.5	Comparing group testing to entropy on $S_{p,0.4}$	148
9.6	Efficacy of group testing on $S_{p,0.4}$	148
9.7	Diagram of encoder for SC(k, j)	150

LIST OF TABLES

2.1	Filter coefficients for the Haar wavelet and Daubechies' biorthogonal wavelet	17
3.1	Elementary Golomb code EG_8	42
4.1	Size 8 group iteration	52
6.1	Comparing GTW and SPIHT-AC	81
7.1	GTWP-J neighborhood significance label	98
7.2	Comparing GTWP with other coders	100
8.1	Comparing non-hierarchical DCT transform coders	115
8.2	Comparing GTBT with other coders	116
9.1	Edge probabilities for the two-state Markov encoder	150

LIST OF PROCEDURES

4.1	Binary Splitting	48
4.2	Hwang's Generalized Binary Splitting	50
5.1	Optimal Binary Splitting	59
6.1	GTW Significance Pass	77
6.2	Group Iteration	78
6.3	Adaptive Group Testing	78

ACKNOWLEDGMENTS

There are many people whom I would like to thank for helping me on the path to writing this thesis and obtaining my degree. First of all, I would like to thank my advisor Richard Ladner, who patiently guided me in this research, helped me organize the many ideas I had into a coherent whole, as well as offered many suggestions that helped improve the quality of this written thesis. I would also like to thank the other reading committee members, Paul Beame and Eve Riskin, for their careful proof reading and for their suggestions on where more detailed explanations would make the thesis more understandable. I would also like to thank my former advisor Anna Karlin for helping me work on my writing skills.

Besides my advisor, there are also many other fellow students that I would like to thank who critiqued my presentations and helped me explain the overview ideas in this thesis in a simple and coherent manner. These people include Dave Hsu, Agnieszka Miguel, Alex Mohr, Steve Wolfman, Ken Yasuhara. Apologies to the many other students that helped that I may have neglected to mention.

Portions of this thesis have been previously published in other conferences; this thesis expands upon these papers and adds some new material. Portions of chapter 5 and chapter 6 can be found in [20]. Chapter 7 can be found in [21]. Chapter 8 will be published in [22].

I would also like to thank the people who graciously provided code for image compression. This includes: G. Davis *et al.* for the Baseline Wavelet Transform Coder Construction Kit [11], H. S. Malvar for code that computes the DCT, LOT, and LBT transforms [36], and T. Tran for code that computes the GLBT transform [54].

Finally, I would also like to thank the people that encouraged and supported me during the time I worked on this thesis. This includes both of my parents Mark Hong and Grace

Hong, as well as some of my friends from my church.

Chapter 1

INTRODUCTION

In the past decade, the transmission, storage, and processing of digital images have become widespread. Although many factors have contributed to the increasing usage of digital images, one principal factor is the availability of inexpensive hardware that is capable of displaying, transmitting, and otherwise using digital images. One important factor making inexpensive hardware capable of using digital images is the use of image compression algorithms. These algorithms have reduced the size of digital images, thus reducing the storage and transmission cost. Since digital images take up a lot of storage space (for example, about 10 million bytes for a standard 4×6 inch photograph), image compression has been instrumental in making the usage of digital images practical and widespread.

The first international digital image compression standard for continuous-tone images was JPEG [57] (for Joint Photographic Experts Group). This standard was instrumental in proliferating the use of digital images because manufacturers made sure that their equipment could understand images in the JPEG format. At the time it was defined, the JPEG standard was competitive with the state of the art image compression methods in terms of compression performance.

Since the advent of JPEG, much research has significantly improved the amount of compression attainable on digital images. Shapiro's embedded zerotree wavelet (EZW) coder [49] pioneered the zerotree technique; this technique was later refined by Said and Pearlman's set partitioning in hierarchical trees (SPIHT) [45] coder. It has proved to be a computationally simple and effective method for coding wavelet-transformed coefficients

of an image. Many other recent image compression methods have also been inspired by zerotree coding. This is particularly true of the more recent JPEG-2000 standard [37], which is a vast improvement upon the original JPEG.

In this thesis, we explain how group testing is intimately related to some methods used in image compression. Group testing is a technique for identifying a few significant items out a large pool. First studied by Dorfman [13], group testing has been applied to many diverse fields due to its generality. The purpose of this thesis is to further explore the ways in which group testing can be applied to obtain image compression algorithms that have good performance.

In chapter 2, we first introduce the field of image compression. We explain some goals that are desired for image coders (such as good compression performance and embeddedness), explain some basic techniques used for recent image coders (such as wavelet transform coding, bit-plane coding, and zerotree coding) , review recent image coding algorithms [49, 45, 60, 40, 34, 52], and provide some of the reasons behind the effectiveness of these image coders.

In chapter 3, we introduce some basic information theory concepts and some basic methods for losslessly encoding any information. Since many image compression techniques rely on these methods of lossless encoding, understanding these concepts is crucial to understanding the efficacy of different image compression techniques.

After this background on image compression, we then introduce the group testing problem in chapter 4, along with some well-known group testing algorithms. Chapter 5 shows the many ways in which group testing can be applied to image compression. In fact, we show that group testing is a generalization of zerotree coding. We also show that group testing can be used as a standard entropy coder for a binary alphabet. This is particularly interesting because the binary splitting procedure used in Hwang's generalized group testing method [14] turns out to be equivalent to elementary Golomb codes. Furthermore, these codes have been used in many recent image compression algorithms.

In chapter 6, we introduce our Group Testing for Wavelets algorithm (GTW), which

explores one particular method in which group testing can be applied to image compression. GTW basically divides the wavelet-transformed coefficients into different classes, and uses a different group tester to code each different class. Our results for this algorithm show that the resulting image quality is comparable to SPIHT when the compression ratio is the same. This shows that GTW is an effective image compression technique, and verifies that group testing is useful for image compression.

In chapters 7 and 8, we introduce several other image coders that rely on the group testing technique, and compare the results with other existing coders found in the literature. We apply group testing to coefficients generated from many different transforms, including the wavelet transform, wavelet packets, the discrete cosine transform, as well as various versions of lapped transforms. We also discuss experimental results from many different variations of our technique. When measuring compression performance, our results show that our image coders are competitive with other recent image techniques when using the same transform; on some images, our image coding results surpass the results of previous image coders. We believe the discussion of our methods and results will provide insight on the benefits and disadvantages of many different image compression techniques.

In chapter 9, we explore the theoretical performance of group testing on images. We first define some statistical processes that generate data; this data can be used as a model for an image. One model we explore is the correlated Markov source. We then analytically determine the performance of different group testing methods on the model data. Because these methods are related to our image coding algorithms, their performance results provide insight into the effectiveness of our image coders.

We conclude in chapter 10 by reviewing the topics covered by this thesis, and suggesting future work.

Chapter 2

IMAGE COMPRESSION BACKGROUND

2.1 *Image Compression Problem*

In the image compression problem, we are given a digital image, and our goal is to produce an encoded description that takes up less storage space than the original image, and also can be decoded to produce the original image. To more formally describe this problem, we first define images, and then explain the various types of image compression and the goals desired for image compression algorithms.

2.1.1 *Digital Images and Quantization*

We can think of an image as a two-dimensional function f such that $f(x, y)$ for any real numbers x, y is a real number representing the color seen at position (x, y) in the image. We define a *digital image* as a discrete approximation (in both position and color value) to the underlying image. Thus, a digital image is a finite array of integers of some fixed size. Each integer represents a discrete approximation to the color seen at that position. Any one element of the array is known as a *pixel* of a digital image.

In this paper, we will only use 8-bit monochrome images, meaning black and white images whose pixels are 8-bit integers ranging from 0 to 255. In this case, 0 represents black, 255 represents white, and the numbers in between represent the many different shades of gray available. Note that compression techniques on 8-bit monochrome images, in general, can also be applied to color images with an arbitrary number of bits used per pixel; studying compression of only 8-bit monochrome images is not a huge restriction.

The process of approximating data such as $f(x, y)$ with a fixed-size integer is known

as *quantization*. Let us assume the color values for $f(x, y)$ ranged continuously from 0 to 1. Then in *uniform scalar quantization* on $f(x, y)$, we pick a fixed *quantization step size* s (such as $\frac{1}{256}$) and approximate $f(x, y)$ by rounding it to the nearest multiple of s . In effect, we are defining *quantization bins*; these are intervals of length s such that any value in that bin will be represented by the midpoint of that interval. We can then number the quantization bins sequentially and represent $f(x, y)$ by the number of the quantization bin in which it is contained. In *non-uniform scalar quantization*, the size of the quantization bins are not uniform. Quantization is a basic compression technique that can be used any time data need to be approximated. As another example, we could uniformly quantize a pixel of an 8-bit monochrome image with step size 8. This means representing the pixel with a 5-bit integer n ranging from 0 to 31, where $8n$ represents the approximation to the value of the original pixel.

2.2 Main Image Compression Goal

An image compression algorithm is defined by a method of encoding the input image into a compressed description, and a method of decoding the compressed description to recover the original image. We work in the field of *lossy image compression*, where the reconstructed image from the decoding process does not have to be identical to the original image; we allow some of the original data to be lost in the compression process. In this framework, encoders can obtain arbitrarily high compression at the expense of lower reconstruction fidelity. The primary goal is to maximize the fidelity (or minimize the distortion) of the reconstructed image when given a fixed number of bits with which to encode an image.

2.2.1 Measures of Image Quality

The two most common objective measures of image quality are mean squared error (MSE) and peak signal to noise ratio (PSNR). The MSE is simply the mean of the squared error

at each pixel location. Let $X = (x_1, x_2, \dots, x_n)$ represent the values of the n pixels of an image, and $X' = (x'_1, x'_2, \dots, x'_n)$ represent the reconstructed image. Then the MSE (denoted σ^2) is defined as

$$\sigma^2 = \sum_{i=1}^n (x_i - x'_i)^2 / n.$$

Note that in general, higher values for σ^2 correspond to lower image quality. The PSNR is defined as

$$10 \log_{10} \left(\frac{M^2}{\sigma^2} \right),$$

where M denotes the maximum possible value for any pixel. For 8-bit monochrome images, $M = 255$. The PSNR is measured in units of decibels (dB). Note that higher values for the PSNR usually correspond to better image quality. Most images with a PSNR of 40.0 dB are of very high quality and are visually identical to the original image; those at 30.0 dB are of medium quality images and have some noticeable degradation; those at 25.0 dB are of low-quality and have significant degradation. Figure 2.1 shows a cropped version of the frequently-used Barbara image at different quality settings, generated using the GTW coder presented in chapter 6.

2.2.2 Rate-Distortion Curves

Rate-distortion curves are used to describe the performance of an image compression algorithm. As shown in figure 2.2, a rate-distortion curve is a plot of the quality of the reconstructed image that the algorithm achieves at all possible sizes for the compressed image description. The size of the compressed image is usually given as a *bit-rate*, which is simply number of bits taken up by the compressed image divided by the total number of pixels in the image. Bit-rates are expressed in units of bits per pixel (bpp).

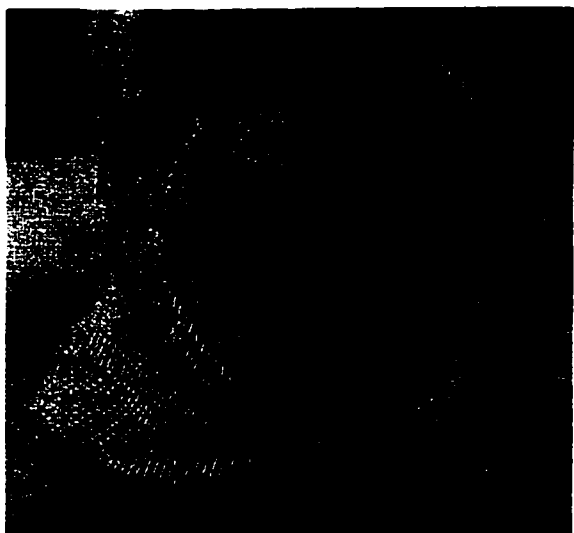
When a rate-distortion curve uses the PSNR as the metric for image quality, we also call it a *PSNR curve*. For PSNR curves, the ideal plot is as far up and to the left as possible. We subjectively say that an algorithm has good *rate-distortion performance* or good *PSNR performance* when its PSNR curves are far enough up and to the left.



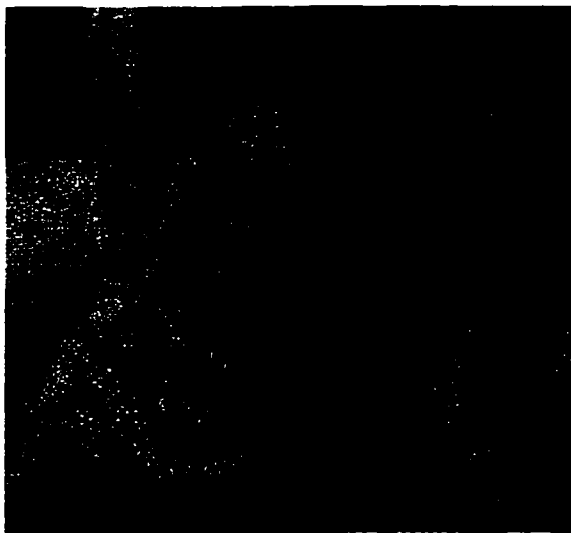
Original



PSNR=40.0 dB



PSNR=30.0 dB



PSNR=25.0 dB

Figure 2.1: A crop of the standard Barbara image and reconstructions at different fidelities.

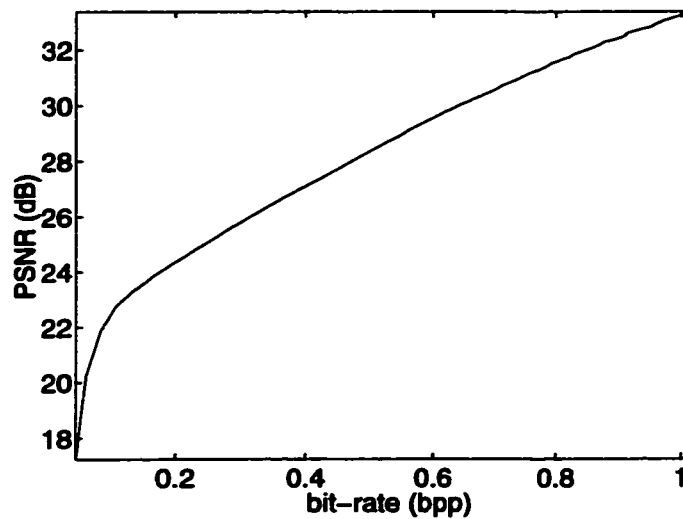


Figure 2.2: A typical rate-distortion curve.

As can be seen in figure 2.2, the slope of the PSNR curve increases steeply at bit-rates near zero; this shows how each bit of the compressed image description for small files is very important to the reconstruction quality. Intuitively, most of the bit budget for small files is allocated to describe long, high-contrast edges in the images and overall intensity levels of large regions in the image; not describing these characteristics would make the reconstructed image fidelity very low. As the bit-rate gets higher, the PSNR curve starts leveling out. Intuitively, more of the bit budget is being used to describe small details of the image, for example, that certain small regions of the image should be slightly lighter or darker than others. As the rate increases, more of the bit budget is used to describe small details, and improvements to the image quality become less noticeable. Thus, we expect the curve to level out because describing one small detail takes roughly the same amount of information as describing overall characteristics of a large region, but improves the image quality much less.

2.3 Other Image Compression Goals

2.3.1 Resource Usage

There are many other goals desired for image compression algorithms. For some applications, the speed of encoding and decoding an image is critical. In other applications, the amount of memory used by the compression algorithm needs to be minimized to minimize the cost of the entire system. Clearly, it is desirable for an image compression algorithm to be both as fast as possible, and to have a memory footprint that is as small as possible.

2.3.2 Training

One issue related to an algorithm's practicality is whether or not it needs to be trained on sample image data. Training algorithms on the specific characteristics of certain types of images should lead to better rate-distortion curves, but training costs more in terms of the computation required for training and the ease of implementation.

2.3.3 Embeddedness

Embeddedness is another characteristic often desired for image compression algorithms. We say that an image compression algorithm is *embedded* when the data that are more important for reconstructing the image are stored before less important data in the compressed image description. If we view the description of the compressed image as a bitstream, then this means that any prefix of the bitstream represents an efficiently compressed encoding of the original image. In fact, if we take a long enough prefix of the bitstream we should be able to get a completely lossless representation of the original image.

An embedded algorithm is useful whenever images are needed at several different quality settings. Given a request for an image at some particular quality setting, an efficiently compressed version of the image at that setting can be obtained by simply truncating the bitstream at the appropriate location. One task that typically uses an embedded image

coder is progressive image transmission. In this task, an image is transmitted over a slow network link, and the receiver continually updates the quality of the image on the screen as it receives more image data.

Another natural use for embedded algorithms is in the context of unequal loss protection. In this framework, the idea is to vary the amount of error correction applied to each part of the data; the more important data are protected more strongly than the less important data. The protected data can then be sent over a network with the knowledge that a reasonable description of the image will get through to the other side even in the presence of lots of unanticipated noise in the network.

Note that it is possible to have an image coder that is not embedded but which still has good rate-distortion performance. This coder would produce a compressed output bitstream based on the original image and the target bit-rate. Furthermore, data produced at different target bit-rates would be completely different from each other, and presumably optimized for the given target bit-rate. It is reasonable to expect non-embedded image coders to have better rate-distortion performance than embedded image coders because they have more freedom in organizing their output bits.

2.3.4 *Resolution Scalability*

The *resolution* of an image measures the number of vertical or horizontal pixels contained in an inch of a displayed digital image. The goal of *resolution scalability* is the ability to easily display the image at different resolutions. For an algorithm to be resolution scalable, the beginning of the compressed image bitstream should contain data for reconstructing a small, low-resolution version of the image. Each successive part of the bitstream, along with the previous bits, should contain the data for reconstructing a larger, higher-resolution version of the image. This capability would be very useful in any system that allows multi-resolution editing of images, or in any system that requires different resolutions of an image for output to different devices (such as a computer monitor or a laser printer).

2.3.5 *Spatial Accessibility*

Spatial accessibility is the ability to easily access an arbitrary cropped portion of the image without having to decode the entire image. This feature, when combined with resolution scalability, would be very useful in a multi-resolution image editing scheme. After viewing a low-resolution overview of the image, a user may want to zoom in on a particular region. For interactive response times on large images, it should not be necessary to decode the entire high-resolution image in order to show the high-resolution crop of the user-specified region.

2.3.6 *Identifiable Bitstream*

Unfortunately, embeddedness and resolution scalability conflict in that they both require the bitstream of the compressed image description to be ordered in conflicting ways. Furthermore, spatial accessibility would suggest that the image should be divided into independently coded blocks, which also appears to conflict with embeddedness and resolution scalability. If embeddedness, resolution scalability, and spatial accessibility are all desired, then one way that makes this partially possible is to make the compression algorithm have an *identifiable bitstream*. This means that the bitstream is divided into easily identifiable chunks; each chunk would contain data representing a small improvement in PSNR for a small block of the image. By using a simple post-processing step on the compressed image, the chunks could be arbitrarily reordered to form an embedded bitstream, a resolution scalable bitstream, or a spatially accessible bitstream. Furthermore, if the image were stored in an embedded format, but only a low-resolution version of the image were required, then the decoder could read through the image description in order and decode only the necessary chunks for producing the low-resolution version; it could easily identify and skip all the unnecessary chunks. Thus, the entire image bitstream need not be decoded in order to produce the desired parts of the image.

2.3.7 Entropy and Image Compression

A more theoretical way to study image compression is in terms of *entropy*. First mathematically defined by Claude Shannon [48], entropy is a measure of the amount of information contained in data. Shannon has shown that the entropy of any piece of data is a lower bound on the number of bits that are required to losslessly encode that data. Since entropy is only defined for data generated from some probability distribution, studying the entropy of images requires modeling real images with probabilistic models. The entropy of these models approximates the amount of information contained in real images. Since the entropy is a lower bound on the amount of compression attainable, the best any image compression algorithm can do is to compress images generated from a probabilistic model at a bit-rate equivalent to the entropy of the model. Studying the entropy of probabilistic models for images is one way of understanding the limits of the performance of image compression algorithms. We discuss this concept in more detail in chapter 3.

2.4 Image Coding Techniques

The best image compression algorithms currently all use some form of transform coding. In this framework, there are three steps in the image encoding process; these three steps are applied in reverse order to decode an image. This is illustrated in figure 2.3.

The first step is to apply a mathematical transform operation to the image pixels to reduce the correlation between individual pixels; the data output from the transform are called *transform coefficients*. Transforms are used because transform coefficients are more amenable to compression than image pixels. In order for the original image pixels to be recoverable, the transform step must be invertible.

Next, the transform coefficients are encoded via a coefficient coding scheme. In lossy image compression, this transform coefficient coding step is where some data are lost. This step uses symbols to represent the transformed image as accurately as possible given a limited budget for the number of symbols. Note that loss occurs because the transform

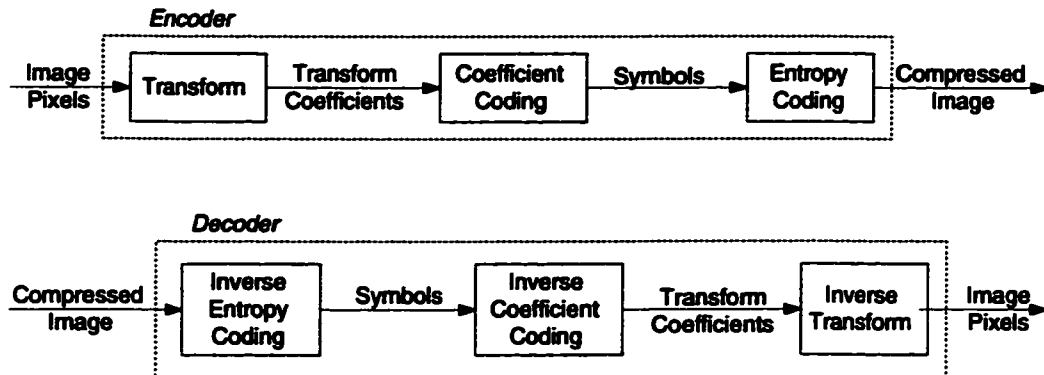


Figure 2.3: Diagram of encoder and decoder for an image compression algorithm using transform coding.

coefficients are quantized (as defined in section 2.1.1).

In the last step, the symbols output from the coefficient coder are losslessly compressed using a standard *entropy code*. Note that an entropy code is simply any method capable of losslessly compressing a sequence of symbols.

We now proceed to describe the typical methods used for the different steps of image coding. We start with giving an overview of different kinds of transforms, followed by one example of a transform, the discrete wavelet transform (DWT). Next, we describe two methods for coefficient coding: bit-plane encoding and zerotree coding. This will be followed by a short discussion of entropy codes; a much more detailed introduction to entropy codes can be found in chapter 3. Throughout this section, we will discuss why these techniques are useful for image compression, and we end this section by summarizing the statistical properties of images that can help achieve good PSNR performance.

2.4.1 Transform Coding

The only transforms frequently studied are linear transforms. For an image consisting of $m \times n$ pixels, the transform can be represented by an $mn \times mn$ matrix, which when multiplied by the mn pixels, results in mn transform coefficients. Typically, the matrix is

sparse and repetitive, so that there are much more efficient ways to compute the transform of an entire image than via matrix multiplication.

Adjacent pixels in an image are almost always correlated, meaning that they have the same color, or a similar color. The purpose of transforming the data is to decorrelate the image pixels. There are typically many transform coefficients whose values are zero or close to zero. This makes the transform coefficients easier to compress because the compression algorithm can assume they are zero unless they are otherwise described. Thus, many coefficients often need not be described.

The efficacy of a transform depends on the amount of energy compaction the transform achieves; higher energy compaction corresponds to fewer coefficients with large magnitude and more coefficients close to zero. There are many different measures of transform efficacy, but one widely used measure is the *coding gain* [46], which is a ratio of the arithmetic mean of the variances of the transform coefficients to their geometric means. It can be shown that the coding gain of a transform is an objective measure of the improvement in compression performance obtained when quantizing the transform coefficients instead of directly quantizing the input data [26]. Although much research has gone into designing transforms for different types of data (including images) and evaluating the transforms with different efficacy measures, this work is beyond the scope of this paper.

So far, we have been describing *non-adaptive transforms*, where the transform is fixed for all input images. In contrast, *adaptive transforms* can vary depending on the statistical characteristics of the input image. Varying the transform is equivalent to varying the matrix that represents the linear transform. Adaptive transforms have a fixed amount of variation that they allow, and try to choose the best variation that will help image compression the most. For the decoder to be able to decode the image, a description of exactly which variation of the transform was used must be contained in the compressed image description.

The transforms described in this thesis include the discrete wavelet transform (section 2.4.2), the wavelet packet transform (section 7.2), the discrete cosine transform (section 8.1.1), and lapped orthogonal and biorthogonal transforms (section 8.1.2). These trans-

forms have all previously been used for image compression.

2.4.2 Wavelet Transform

Wavelet Transform Overview

The wavelet transform represents its input in terms of functions that are localized in both time and frequency. Mathematically, the wavelet transform approximates a function by representing it as a linear combination of two sets of functions: Φ and Ψ . The set Φ is constructed from a function $\phi(t)$ known as the *scaling function*, while Ψ is constructed from a function $\psi(t)$ known as the *mother wavelet*. Note that both $\phi(t)$ and $\psi(t)$ are *time-limited*, meaning that the values $\phi(t)$ and $\psi(t)$ are non-zero only when t is within a small interval. Every function $f \in \Phi$ is a *translation* of $\phi(t)$, meaning $f(t) = \phi(t - c)$ for some real value c . Every function $f \in \Psi$ is a translation and/or *dilation* of $\psi(t)$, meaning $f(t) = \psi(c_1 t - c_0)$ for some real values c_0 and c_1 . Once Φ is specified, then the mother wavelet $\psi(t)$ as well as the set Ψ is completely determined; Ψ must be defined so that the original function can be reconstructed from the transform coefficients.

The coefficients of the functions in Φ represent a low-resolution overview of the input function; the Ψ coefficients represent differences between the low-resolution overview and the actual input necessary to obtain a better approximation. For a more detailed introduction on wavelets, see [10]. The scaling function and mother wavelet for the simple Haar wavelet and Daubechies biorthogonal wavelet [3] are shown in figure 2.4. Note that the Daubechies biorthogonal wavelet is the wavelet typically used for wavelet-based image compression.

Discrete Wavelet Transform as Filters

The discrete wavelet transform (DWT) is applied to discretely sampled data and is based on two filters: a low-pass filter that outputs Φ coefficients representing low-frequency data, and a high-pass filter that outputs Ψ coefficients representing high-frequency data. Transforming a one-dimensional input signal of length n produces $n/2$ low-frequency coefficients

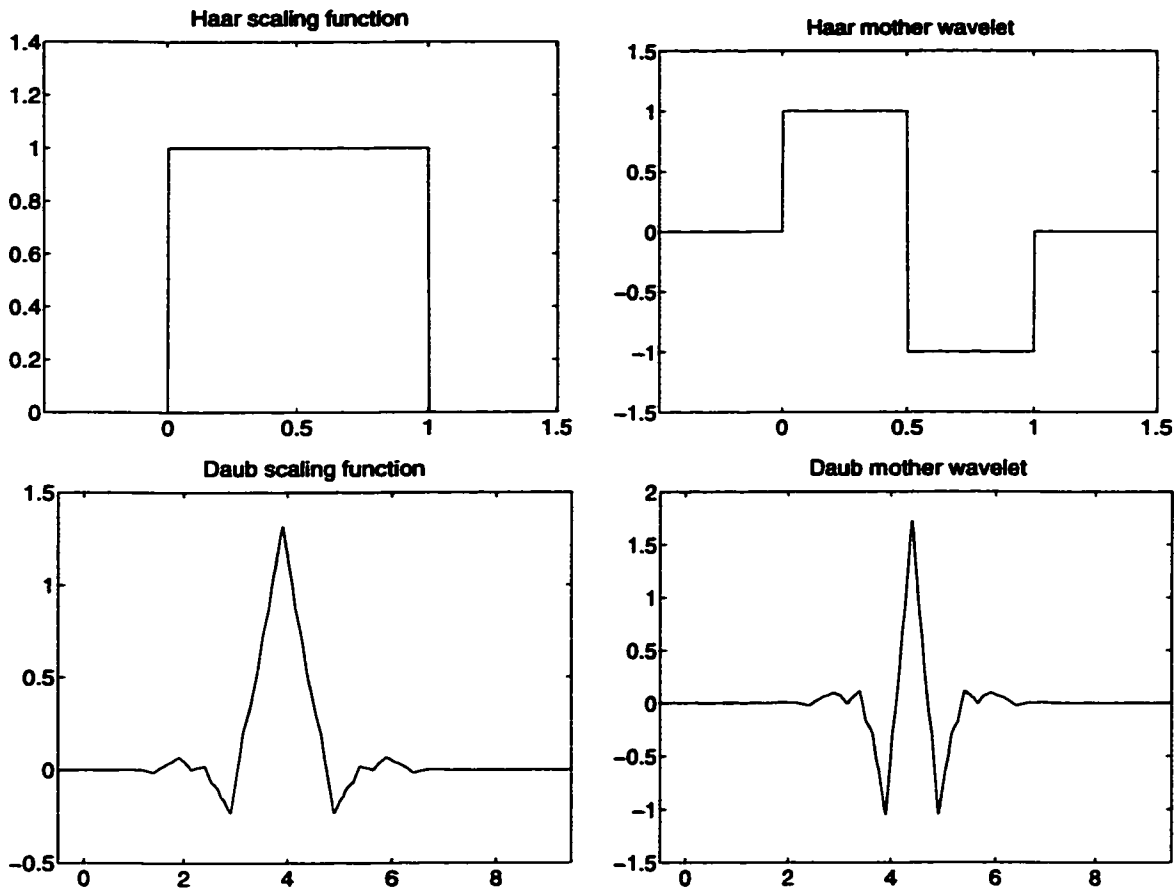


Figure 2.4: The Haar Wavelet (top) and Daubechies' Biorthogonal Wavelet (bottom).

from the low-pass filter and $n/2$ high-frequency coefficients from the high-pass filter. Thus, the number of input data points in the signal is the same as the total number of output coefficients. Note that wavelets are defined so that the output coefficients can be inverted to exactly reconstruct the original input data.

A *filter* is defined by a finite set of filter coefficients. If a filter has k filter coefficients, then it is known as a *k-tap filter*. The time required to compute the DWT is directly proportional to k . Although higher tap filters are more expensive to compute, they can also be more efficient at decorrelating the input data. The filter coefficients for the Haar wavelet and the Biorthogonal Daubechies wavelet are shown in table 2.1. As can be seen, the Haar

wavelet consists of two very simple 2-tap filters, while the Daubechies biorthogonal wavelet consists of a 7-tap filter and a 9-tap filter.

Table 2.1: Filter coefficients for the Haar wavelet and Daubechies' biorthogonal wavelet.

Wavelet	Low-pass filter	High-pass filter
Haar	$\sqrt{2}/2$	$\sqrt{2}/2$
	$-\sqrt{2}/2$	$\sqrt{2}/2$
Biorthogonal	-0.06453888	0.03782846
	-0.04068942	-0.02384947
	0.41809227	-0.11062440
	0.78848562	0.37740286
	0.41809227	0.85269868
	-0.04068942	0.37740286
	-0.06453888	-0.11062440
		-0.02384947
		0.03782846

The filters for the Haar wavelet are particularly simple to compute. Given an input sequence (x_1, x_2, x_3, \dots) , the low-pass filter essentially averages every pair of consecutive numbers, producing the sequence $\frac{\sqrt{2}}{2}(x_1 + x_2, x_3 + x_4, x_5 + x_6, \dots)$. The high-pass filter essentially computes the difference between every pair of consecutive numbers, producing $\frac{\sqrt{2}}{2}(x_2 - x_1, x_4 - x_3, x_6 - x_5, \dots)$. Thus, the example sequence $(3, 8, 12, 2, 7, 1)$ would be transformed into the low-pass sequence $\sqrt{2}(5.5, 7, 4)$ and the high-pass sequence $\sqrt{2}(2.5, -5, -3)$.

More generally, the output of a filter can be computed by a *convolution* of the filter coefficients with the input data followed by *subsampling*. A convolution of filter coefficients c_1, \dots, c_m with input data x_1, \dots, x_n produces output y_1, \dots, y_n defined as follows:

$$y_i = \sum_{j=1}^m c_j x_{i+j-\lceil m/2 \rceil}.$$

To make this sum well-defined, we reflect the value of the coefficients at the border: for

$i \leq 0$, $x_i = x_{-i+1}$, and for $i > n$, $x_i = x_{2n-i+1}$. The subsampling halves the number of coefficients output by the convolution by keeping only odd-numbered coefficients. Note that the DWT need not be computed this way; since subsampling removes half the coefficients, the half of the coefficients output from the convolution that would be removed do not need to be computed. In fact, in terms of number of arithmetic operations, a very computationally efficient method of computing the DWT, called the lifting method, was presented in [56].

The Haar wavelet, although very simple to compute and understand, is not an extremely effective transform for images. Intuitively, representing the input as coefficients from the Haar scaling function results in an approximation that consists of many stair steps. We know that adjacent input pixels of images more often vary continuously in intensity, and less often in discrete jumps. Thus, we might expect a wavelet with a scaling function that has a smoother slope to be able to represent our input pixels better. Although methods for designing wavelets are beyond the scope of this paper, [3] shows that the previously presented 7/9-tap biorthogonal filter is particularly suitable for transforming images.

Subbands and DWT Levels

A *subband* is the set of transform coefficients output when applying one filter to the input data points. Thus, after the DWT is applied once to the input data points, there are two subbands, one representing the low-pass filter output (L_1), and the other representing the high-pass filter output (H_1). The DWT is typically applied in a series of levels, where the 2nd level of the transform recursively applies the DWT to the L_1 , creating two new subbands: a low-pass level 2 subband (L_2) and a high-pass level 2 subband (H_2). In general, L_i (H_i) is the low-pass (high-pass) subband created by applying the DWT on L_{i-1} . This process is illustrated in figure 2.5. Note that we refer to low-pass subband and low-frequency subband synonymously; the same is true for high-pass subband and high-frequency subband. Note also that under our definition, L_{i+1} represents lower-frequency content than L_i ; similarly, H_i represents higher-frequency content than H_{i+1} .

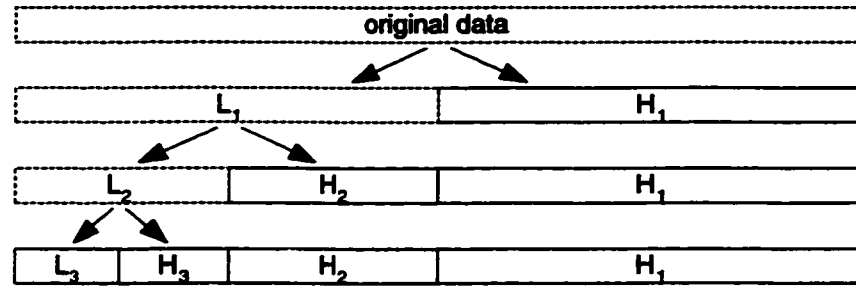


Figure 2.5: Illustration of a one-dimensional DWT transform when 3 levels are performed. Dotted boxes represent data to which the DWT is applied.

Since the low-pass subband data represents a rough overview of the entire data, we expect the coefficients in this subband to still have significant correlation with their neighbors. Thus, it is useful to reapply the DWT to the low-frequency subband to further reduce the correlation. Note that the number of coefficients in the low-pass subband is halved with every level. The low-pass subband is typically decomposed until the number of coefficients left in the low-pass subband is small. With only a few coefficients left, applying an additional level of DWT is typically unable to decorrelate the coefficients further. The well-known technique of applying filters to input data, and encoding the resultant subbands is known as *subband coding*. However, Mallat [32] was the first to apply many levels of the DWT; this showed that wavelets could be used for subband coding.

DWT on Images

When applied to two-dimensional data such as images, the DWT is applied horizontally to each row of the image, followed by vertically on each column of the image. One level of the transform represents applying the DWT once horizontally, then once vertically. Thus, the first level of the transform consists of four subbands: a horizontally and vertically low-pass subband (LL), a horizontally low-pass and vertically high-pass subband (LH), a horizontally high-pass and vertically low-pass subband (HL), and a horizontally and vertically high-pass subband (HH). These four labels define the *orientation* of a subband, which describes the

filters that were applied in each direction to generate the subband. The four subbands of different orientation from one level of the transform are shown in figure 2.6.

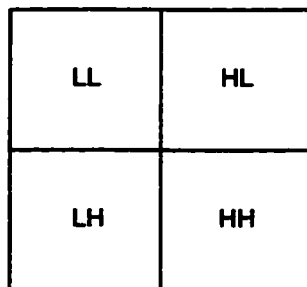


Figure 2.6: Subbands of different orientation after applying one level of DWT.

The LL subband represents a low-resolution overview of the image, while the other subbands represent a high-frequency detail information necessary to reconstruct the image. In the LH, HL, and HH subbands, most coefficients are close to zero; those that are not close to zero represent edges in the image. Note that the number of coefficients in each subband is a factor of 4 less than the number of input pixels.

Each successive level of the transform operates only on the LL subband data produced by the previous level, which still contains highly-correlated pixels. Figure 2.7 shows three levels of wavelet transform applied to an image. We use LH_i to represent the subband with orientation LH created by the i th level of the wavelet transform. HL_i and HH_i are defined similarly. In an n -level wavelet transform, there will be an LL_n subband representing a small overview of the entire image (this is the lowest-frequency subband), as well as n LH subbands, n HL subbands, and n HH subbands. Note that any subband in level i has a quarter of the number of coefficients of any subband in level $i - 1$. This particular method of recursively applying the DWT in a series of levels to the LL subband is known as the *dyadic wavelet decomposition*, the *pyramidal wavelet decomposition*, as well as the *octave-subband wavelet decomposition* of an image. It has also been called by the simpler (but semantically overloaded) term “wavelet transform of an image”.

The dyadic wavelet decomposition is useful for image compression because it hierarchi-

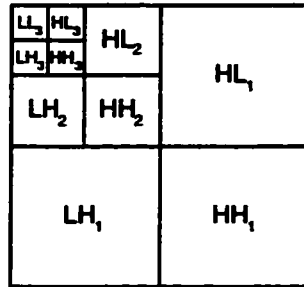


Figure 2.7: Subbands formed with 3-level wavelet transform.

cally decomposes the data into many subbands. This structure efficiently represents edges in the high-frequency subbands, as well as smooth regions in the low frequency subbands. Note that edges in an image typically correspond to larger magnitude coefficients in the high-frequency subbands. Images with lots of edges tend to have more larger magnitude coefficients in their high-frequency subbands. As an example, the Barbara image (figure 2.1) has lots of edges due in part to the many stripes in Barbara's clothes; transform coefficients for this image tend to have larger magnitude coefficients in the high-frequency subbands than those for the typical image without many edges. Recent image coders have shown that the dyadic wavelet composition is an efficient representation that can lead to good rate-distortion performance.

2.4.3 Bit-plane Encoding

The *bit-plane encoding* approach was first applied to data generated by a space probe in [47]. Instead of sending the values of each data point as it is generated, this scheme first buffers a large block of k data points and sends the most significant bit of each data point in the block. This is followed by the next most significant bit of each data point, and so on, down to the least significant bit of each data point of the block.

Bit-plane encoding has also been applied to progressive image transmission (see [41]). In this scheme, an image is sent in a series of *bit-planes*, where bit-plane i represents the i th

most significant bit of all the pixels in the image. Thus, receiving the first bit-plane results in a black and white overview of the entire image, receiving the second bit-planes refines the picture with two additional shades of gray, and receiving further bit-planes increases the number of shades of gray represented, improving the accuracy of the received image. To obtain compression, standard compression techniques are applied to the bits within each bit-plane before they are sent.

Bit-plane encoding has also more recently been applied to wavelet-transformed image coefficients. If we normalize the transform coefficients so that their magnitudes are less than 1, then bit-plane i consists of the i th most significant bit of the magnitude of all coefficients written in binary. The bit-planes are coded in order, so that, in general, no bit from bit-plane i is coded before all bits from bit-plane $i - 1$ have been coded.

The bit-plane ordering represents successive refinement of a simple scalar quantization (see section 2.1.1) of the coefficients. With each additional bit-plane, the quantization bins get smaller and the known precision of each coefficient increases. We say a coefficient is *significant* when enough bit-planes have been coded so that its value is known to be nonzero. A coefficient is *insignificant* or a *zero coefficient* when it is still indistinguishable from zero. Coefficients large in magnitude will become significant after a few bit-planes have been coded; smaller coefficients will become significant later.

Since the wavelet coefficients can be positive or negative, the sign of each coefficient must also be encoded. The sign is typically encoded when a coefficient is discovered to be significant. This way, no sign bit needs to be encoded for coefficients that are zero.

This bit-plane approach has been proven to be an effective method for generating an embedded bitstream. Said and Pearlman [45] show that greater magnitude coefficients in the wavelet transform domain affect the quality of the image more than the coefficients of lesser magnitude. This means that to get the highest quality image quickly, first code the information from the largest coefficients. This suggests that the successive refinement strategy of bit-plane encoding is the best way to generate an embedded bitstream. Also note that the sign of a coefficient will convey no information while the coefficient is insignif-

icant, but that the sign is very important once the coefficient becomes significant. Thus, sending the sign bit immediately after a coefficient becomes significant helps produce a good embedded bitstream.

Although many recent image coders encode the wavelet-transformed coefficients in a bit-plane by bit-plane fashion, they all differ in the method of compressing a bit-plane. Some of the methods used in compressing a bit-plane will be discussed in the following subsection.

2.4.4 Significance and Refinement Passes

Wavelet image coding methods typically code each bit-plane with a *refinement pass* and a *significance pass*. The *refinement pass* of the algorithm sends one more bit for each already significant coefficient, refining it further. The *significance pass* codes the other coefficients by identifying the coefficients that will become significant at this bit-plane. The two passes represent a division of the coefficients into two groups: one containing coefficients that became significant in a previous bit-plane, and one containing coefficients not yet known to be significant.

This division is important because the probability that a refinement bit is a one is much different from the probability that a significance pass bit is one. Most significance pass bits are expected to be zero because many wavelet coefficients are near zero. In contrast, the refinement bits should have roughly an equal chance of being one or zero. This division leads to better rate-distortion performance.

2.4.5 Fractional Bit-planes

The *fractional bit-plane method* is a generalization of the significance and refinement passes; it was introduced in [40]. In this method, a bit-plane is coded with many passes (typically four passes) instead of just a significance pass and a refinement pass. The first pass codes the bits that are most likely to become significant according to some statistical

criteria. The next pass codes bits that are slightly less likely to become significant. This proceeds, with each successive pass coding bits that are even less likely to become significant. The last pass is typically the refinement pass. The idea behind this method is that coding bits with a higher probability to be significant is more likely to be beneficial to the overall image quality than coding other bits.

2.4.6 Zerotrees

Shapiro [49] pioneered the use of *zerotrees* in its significance pass to predict the insignificance of coefficients. A zerotree is a set of zero coefficients from many different subbands of similar orientation that represent the same spatial location in the image. This set is organized in a tree, where each coefficient not in the lowest-frequency subband has four children coefficients in the next higher-frequency subband of similar orientation (assuming it exists). These children always represent the same spatial location, as illustrated in figure 2.8. Coefficients in the lowest frequency subband have children in the subbands of different orientations at the same transform level. Although Shapiro's EZW and Said and Pearlman's SPIHT [45] both used these trees, the exact assignment of children to the lowest frequency subband was slightly different in the two algorithms.

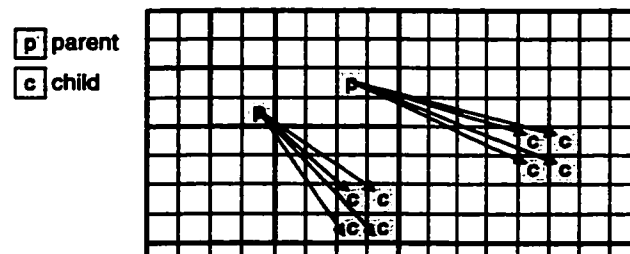


Figure 2.8: Illustration of two parents connected to their children in a 3-level wavelet transform.

The zerotree method places all the coefficients into large tree structures, and tests whether the trees are *significant*. A tree is significant if any coefficient in that tree is sig-

nificant; a tree is insignificant if all its coefficients are insignificant (i.e. if it is a zerotree). Each tree is coded with a single symbol. Only the significant trees must be subdivided into smaller subtrees which are then tested for significance and coded; the values of all coefficients in an insignificant tree are known after coding it as a zerotree.

This method was based on the following zerotree property, which Shapiro discovered was generally true: if a coefficient is found to be insignificant in a given bit-plane, then all its descendants (its children, children's children, etc.) are also likely to be insignificant. The significance pass uses this property by representing an entire zerotree with a single symbol. This takes advantage of the zerotree property by using one symbol to represent the value of many insignificant coefficients.

Another key idea in this technique is that only the results of the tests for significance need be encoded; it is not necessary to use symbols to encode which trees of coefficients were tested. This is because the encoder and decoder can agree on a deterministic algorithm that chooses which trees to test based only on the test results seen so far.

The advantages of this zerotree technique are that it is simple, easy to implement, relatively fast, and that it requires no training. It also performs well in terms of rate-distortion performance.

2.4.7 Entropy Coding

Recall that *entropy coding* is the term used to describe any method used to losslessly compress an input sequence of symbols. *Non-adaptive* entropy coding assumes knowledge of some statistical characteristics of the input, e.g. knowing the expected frequency distribution of the input symbols. *Adaptive* entropy coding assumes the statistical characteristics of the input sequence will change over time, and tries to adapt to the changing characteristics. Adaptive entropy coding techniques typically try to estimate the frequency distribution of the input as more symbols are coded. These methods change their method of encoding whenever a symbol appears more often than previously statistical estimates would suggest. Adaptivity works well with small alphabets because the changing probabilities of the source

can be quickly estimated.

Entropy coding can compress efficiently when a lot of correlation between source symbols remains in the source. As mentioned before, we know that the transform coefficients of an image are often near zero. Thus, we expect the leading bit-planes of the significance pass coefficients to be mostly zero. With this distribution skewed towards zero, we would expect any entropy coding technique to be able to obtain a high amount of compression on these bits.

One important idea in entropy coding is the notion of *contexts*. When encoding a sequence of symbols, often the symbols are correlated in that the conditional statistical distribution of the next symbol given different previous values are vastly different. For example, in English text, we know that 'q' is always followed by 'u'; If 'q' is the previous symbol, then the next symbol should be encoded with a different method than if 'x' were the previous symbol. A *context*, then, is simply the set of values for previous symbols with which you choose a particular method of encoding the next symbol. Thus, the method used or the statistical characteristics assumed about the next symbol depends on the current context.

Two entropy codes used in image compression algorithms are arithmetic coding and elementary Golomb codes. We will describe these two entropy codes and give a more detailed introduction to entropy coding in chapter 3.

2.5 Statistical Characteristics of Images

Compression works by finding how the input data are correlated, and then by removing this correlation so that the data can be represented in fewer bits. To explain why a particular wavelet image coder performs well, we need to understand some of the statistical characteristics of natural images so that we can see how the image coder exploits them. Since we study compression algorithms that use the DWT, this section describes the statistical characteristics of wavelet-transformed coefficients.

Two statistical characteristics of wavelet-transformed coefficients have been previously

mentioned: Section 2.4.2 mentions that the values of the transform coefficients are typically zero or close to zero. This means that few coefficients have a large magnitude. In addition, the coefficients typically obey Shapiro's zerotree property (section 2.4.6); this implies that zerotrees occur frequently.

Another property of wavelet coefficients is that the magnitude of a child coefficient is usually less than that of its parent. This implies that coefficients in a lower-frequency subband are more likely to become significant before coefficients in a higher-frequency subband. This knowledge was used in deciding the testing order for the trees and the method of subdividing the trees in the zerotree methods. As an example, Shapiro's EZW algorithm always subdivides a significant tree into several parts, where the first part is the former root coefficient of the significant tree. This root coefficient is always checked for significance before the other parts of the significant tree, because it is more likely to be significant than the other parts.

The key observation about wavelet coefficients in Said and Pearlman's SPIHT algorithm [45] was the following: if a coefficient becomes significant in bit-plane k , then it is likely that its *siblings* (those other coefficients having the same parent) will become significant in bit-plane k or became significant in a previous bit-plane. By exploiting this fact in the way it subdivided its trees, SPIHT improved upon EZW in terms of rate-distortion performance.

More recent work ([40, 60, 52, 7]) has shown that significant coefficients are likely to be found in clusters that are close together. This is similar to the property that SPIHT exploited, but more general because it implies an adjacent neighbor of a significant coefficient in any subband is likely to be significant, regardless of whether the two coefficients share the same parent. This can be exploited by altering the strategy of coding a coefficient based on the values of its neighboring coefficients.

In general, the more correlation between coefficients that an algorithm can exploit, the better its rate-distortion performance. We can see the basic trend in obtaining better compression is to find some property of wavelet coefficients that was not previously accounted for, and then exploit it to obtain better compression.

2.6 Recent Wavelet-based Image Compression Algorithms

The image compression algorithms we present all use the wavelet transform and bit-plane encoding. The main way in which they differ is how they code each bit-plane. We now explain some recent algorithms and point out some of their key ideas and contributions.

2.6.1 EZW

Wavelet-based image compression has received much attention ever since Shapiro proposed his EZW [49] algorithm. This algorithm was significant because it was embedded and it had excellent rate-distortion performance. Its rate-distortion performance was better than or competitive with virtually all previously known image compression techniques, including those that were not embedded. Furthermore, the algorithm was fast at encoding and decoding. In fact, it was much faster than some previous techniques that had worse PSNR performance. The EZW algorithm codes a bit-plane with significance and refinement passes, where the significance pass uses zerotrees. Both passes use an adaptive arithmetic coder for entropy coding.

One of the most important ideas that came out of the EZW paper was that the method of coding the significance pass is crucial to the effectiveness of the algorithm. Using a simplified model for the wavelet-transform coefficients, Shapiro used an entropy argument to show the following bounds for an ideal transform coding system on a simple image model using non-uniform scalar quantization: When encoding an image at 0.5 bpp, the probability that a coefficient will be quantized to zero is at least 95%, and the cost of encoding the significance pass is 54% of the bit budget. Furthermore, as the target bit-rate decreases, the probability of zero increases, and the percentage of the bit budget allocated to the significance pass increases as well.

Because almost all of the coefficients are insignificant in the first few bit planes, zerotrees effectively encode these initial bit planes; this is one reason EZW produces very good images at low bit-rates. As the bit-rate gets higher, more and more coefficients be-

come significant, and the zerotree property starts to hold less often. Comparing EZW to other work, it seems that the zerotree idea was important not in and of itself, but because it showed that you could relate the coefficients in one subband to coefficients in other related subbands, and use this relation to predict coefficient values.

2.6.2 *SPIHT*

Said and Pearlman's SPIHT [45] coder improved upon EZW by using a different wavelet, a different way of organizing coefficients into zerotrees, and a different arithmetic coder with context. Otherwise, it was roughly the same as EZW. One important characteristic of SPIHT is its fast run-time performance. Some of the algorithmic differences between EZW and SPIHT serve to make SPIHT somewhat faster than EZW. SPIHT codes its trees in such a way that whenever a single coefficient is checked for significance, all of its not-yet-coded siblings are also checked for significance immediately. This helps improve performance since siblings are highly correlated.

Said and Pearlman analyzed the PSNR performance of SPIHT with arithmetic coding (SPIHT-AC), as well as SPIHT without arithmetic coding (SPIHT). Interestingly enough, the PSNR performance improvement in adding arithmetic coding was only about 0.3 dB at a wide range of bit-rates. This shows that if execution speed is valued over PSNR performance, the arithmetic coding step could be omitted without paying too much in PSNR performance. SPIHT-AC uses an adaptive arithmetic coder with several contexts; the context for a coefficient consists of the number of its siblings already found to be significant.

SPIHT-AC obtained PSNR performance that was about 1.0 dB better than EZW on average. As previously mentioned in section 2.5, the key observation that helped SPIHT obtain this level of performance was in realizing that adjacent coefficients are correlated.

2.6.3 ECECOW

Wu's embedded conditional entropy coding of wavelet coefficients (ECECOW) [60] pushed the envelope of rate-distortion performance. Instead of zerotrees, ECECOW uses the more traditional context modeling approach to code a bit-plane. The context defined for a given coefficient consists of all the bits known so far for its neighbors. Every coefficient has about 8 neighbors, including some of the adjacent coefficients in the same subband, the parent coefficient, coefficients adjacent to the parent coefficient, and coefficients representing the same spatial location but in the neighboring subband of different orientation. The exact number of neighbors as well as the particular neighbors used depended on the orientation of the subband containing the coefficient. In this context modeling approach, coefficients in different contexts are coded with a different entropy coder. This is because coefficients with the same context are likely to have the same probability of being significant. ECECOW uses an adaptive arithmetic coder to code each context.

Since this algorithm has many contexts, it suffers from the *context dilution* problem: many coefficients will only have a few other coefficients that share the same context and the adaptive coder cannot effectively estimate the probabilities required to adapt to the characteristics of the context. To overcome this, ECECOW uses training to determine which contexts have similar characteristics, and merges them together.

ECECOW is based on the idea that in coding any coefficient, one should ideally use all the information that has already been coded to predict its value. With a perfect model containing information about the dependencies for every coefficient, we would be able to encode the image perfectly, at its entropy. Following this approach, ECECOW uses as many bits as are computationally feasible in the context. Instead of using just whether or not the neighbors of a coefficients are significant yet, ECECOW's context includes the complete value for neighbors. This method also shows that a coefficient exhibits dependencies with its surrounding neighbors, and not just with its siblings.

Another important idea in ECECOW is that the sign-bit of the wavelet coefficients

exhibits some correlation with surrounding coefficients. Unlike EZW or SPIHT, ECECOW also defines contexts for encoding the sign bit; this results in improved PSNR performance. In fact, Deever and Hemami [12] show experimentally that properly encoding the sign bits yields an average of 0.3 dB increase in PSNR performance over a wide range of bit-rates and images.

ECECOW and its variants were shown to have the best rate-distortion curves so far; improvements of 0.4 to 1.3 dB over SPIHT were obtained on standard test images at various bit-rates. This is at the cost of high complexity, slow run-time performance, and training required.

2.6.4 *Ordentlich et al.'s Algorithm*

Ordentlich *et al.* [40] defined an algorithm (which we call OWS) that aimed more for simplicity than for best rate-distortion performance. Like ECECOW, this method uses context modeling; however, its contexts are much simpler. There are only four contexts used: one for the refinement pass coefficients; one for coefficients with a significant neighbor; one for coefficients with a significant parent and no significant neighbors; and one for coefficients with no significant parent or neighbors. The neighbors are the eight spatially adjacent coefficients to the given coefficient. This algorithm uses fractional bit-planes to code the coefficients. Four passes are made through the bit-plane, one pass for each of the four contexts. The passes are ordered so that passes with coefficients more likely to be significant are done earlier than those with coefficients less likely to be significant. Each pass is coded with a different adaptive elementary Golomb coder.

This paper brings up an interesting issue. There are two goals in deciding the order in which to code the bits of a bit-plane: code the bits that reduce the distortion most (i.e., those most likely to be significant), or code the bits that will give you the most information, enabling more accurate predictions of the not yet coded bits. It is interesting that OWS gives preference to the bits that reduce distortion. OWS achieves rate-distortion performance comparable to SPIHT, although it is a little worse (on average approximately

0.05 dB worse). However, it is fairly simple. The elementary Golomb coder is part of its simplicity.

2.6.5 PWC

Malvar's progressive wavelet coder (PWC) [34] algorithm was different in that it was designed for resolution scalability, good rate-distortion performance, as well as speed.

Note that the wavelet transform technique naturally produces a multi-resolution image; the reconstructed LL subbands from each level of the transform represent the original image at different resolutions. Thus, to be completely resolution scalable, it is necessary to first encode all the bits of the coefficients for reconstructing the lowest frequency LL_n subband followed by all the bits for reconstructing the LL_{n-1} subband, and so on. Following this ordering would mean that the bit-plane by bit-plane ordering is not strictly followed; this results in a noticeable loss in embedded performance, at the gain of resolution scalability.

The PWC algorithm codes transform coefficients block by block for resolution scalability and spatial accessibility. Within each block, coefficients are coded bit-plane by bit-plane with a significance and a refinement pass. After reordering coefficients of a block in a complicated but fixed way, PWC makes a significance pass over the reordered coefficients, using Langdon's adaptive elementary Golomb coder (discussed in section 3.3.4) to code the bits. The refinement pass bits are not entropy coded. The reordering was defined to make the zero bits more likely to be closer together; this improves Golomb coding performance. The rate-distortion performance of PWC is comparable to SPIHT at high (around 1.0 bpp) bit rates. Although rate-distortion performance of PWC at low bit-rates was not published, it would almost surely be somewhat worse than SPIHT because embeddedness was traded for resolution scalability.

This algorithm does not use the dependencies between neighbors or between a parent and its children at all, unlike the previous algorithms. That makes this method the simplest so far. The lack of dependencies also allows for the blocks of wavelet coefficients to be independently coded.

Note that ECECOW and OWS could be easily modified for resolution scalability because it is easy to apply these algorithms to code a block of wavelet-transformed coefficients. This is not possible with the zerotree techniques because a block of wavelet-transformed coefficients may not contain any trees.

2.6.6 EBCOT and JPEG-2000

Taubman's embedded block coding with optimized truncation (EBCOT) coder [52] was significant because it introduced the idea of identifiable bitstreams (see section 2.3.6) and demonstrated an image coder with that capability. EBCOT also showed that identifiable bitstreams are possible without too much complexity or loss in overall rate-distortion performance. His coder featured the capabilities of embeddedness, resolution scalability, and spatial accessibility through the use of identifiable bitstream.

EBCOT divides each subband into relatively small code-blocks (32×32 or 64×64). Code-blocks are then coded independently, in bit-plane order. To construct an identifiable bitstream, code-blocks are tagged with information so that they can be split into k chunks each with some specified quality metric. An embedded bitstream can be produced by ordering the bit stream so that the first chunk from each code-block comes first (forming layer 1), followed by the second chunk from each code-block (forming layer 2), and so on down to the last chunk from each code-block. Since the code-blocks are relatively small, and the multi-resolution wavelet transform method is used, this coder is also spatially accessible and resolution scalable.

Code-blocks are encoded in bit-plane order, with a zerotree-like method. The code-blocks are subdivided into sub-blocks, typically of size 16×16 . A quad-tree structure is placed over the sub-blocks; we can view the encoding method as a typical zerotree coding method except that when a sub-block is found to be significant, then the bit-plane of that sub-block is encoded with a specialized sub-block coding method. The coefficients of a significant sub-block are coded by an arithmetic coder with 18 different contexts. A coefficient's context depends on its significance status (i.e. if it is a significance or refinement

bit) and on the significance status of its eight adjacent neighbors. Furthermore, there are also five additional contexts for encoding the sign bits of coefficients that just became significant. Like the OWS algorithm, the bit-plane of a sub-block is coded sequentially in a series of four passes using the fractional bit-plane method.

EBCOT can be anywhere from 0.0 dB to 1.1 dB better than SPIHT depending on the actual image and encoding bit-rate. Over a wide variety of images and bit-rates, PSNR performance improvement over SPIHT averages out to be about 0.4 dB.

The EBCOT algorithm is the basis for the JPEG-2000 standard; the main differences of JPEG-2000 help improve the execution speed at the expense of an average loss of 0.2 dB in PSNR performance when compared to EBCOT. These changes include using a low complexity, less-effective arithmetic coder, reducing the size of the sub-blocks to 4×4 , and reducing the number of fractional bit-plane passes to three. With these changes, the execution speed of a heavily optimized version of JPEG-2000 coder is comparable to SPIHT for small images, and significantly faster for large images. In large images, SPIHT suffers from inherently non-local memory accesses that slow it down.

Chapter 3

ENTROPY CODING AND INFORMATION THEORY

Entropy coding losslessly compresses an input sequence of symbols by representing it with a more compact sequence of output symbols. Although the alphabet for the output symbols could be arbitrarily large, for simplicity, we assume the output symbols are bits, (i.e., they come from the binary alphabet).

To effectively compress the input sequence, we should represent input symbols that are more likely to occur with fewer bits than symbols that are less likely to occur. This illustrates the necessity of having a statistical model for the occurrence of the input symbols. The effectiveness of an entropy code depends not only on the method of encoding the input sequence, but also on the probability that any particular input symbol is likely to occur.

We first proceed to give a mathematical description of entropy codes. To measure their effectiveness, we define the bit-rate of an entropy code that represents the amount of bits per source symbol that a particular code uses. We then give a brief introduction to information theory, defining the concept of entropy. Since the entropy of a source is a lower bound on the bit-rate of any entropy code on the source, the entropy is used to judge the effectiveness of entropy codes. Lastly, we discuss some specific entropy codes used in image compression, in particular, arithmetic coding and Golomb codes, as well as their the effectiveness in terms of entropy.

3.1 Introduction to Entropy Codes

When studying the theoretical performance of entropy codes, we are concerned with encoding symbols from an m -symbol input alphabet $\Sigma = \{a_1, a_2, \dots, a_m\}$ that are gener-

ated from some *source* S . We think of S as generating an infinite sequence of symbols x_1, x_2, x_3, \dots according to some statistical process; the x_i 's are not necessarily generated independently. For any string $\mathbf{s} = s_1 s_2 \dots s_n$ from alphabet Σ , we define

$$\mathbf{Pr}[\mathbf{s}] = \mathbf{Pr}[x_1 = s_1, x_2 = s_2, x_3 = s_3, \dots, x_n = s_n].$$

An entropy code encodes a source S by representing the input symbols with output bits from the binary alphabet $\Sigma_2 = \{0, 1\}$. We will use Σ^n to denote the set of strings of length n from alphabet Σ , and Σ^* to denote the set of all strings from Σ . Thus, Σ_2^* denotes the set of all binary strings.

An entropy code is basically a map from input strings in Σ^* to output strings in Σ_2^* . This mapping is represented by the *encoding function* $f : \Sigma^* \mapsto \Sigma_2^*$. For the code to be uniquely decodable, f must be injective.

The main goal in defining entropy codes is to minimize their *bit-rate*, the average number of bits required to encode one source symbol. The bit-rate $R(f, S)$ of a code with encoding function f on source S is defined as follows:

$$R(f, S) = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{\mathbf{s} \in \Sigma^n} \mathbf{Pr}[\mathbf{s}] |f(\mathbf{s})|. \quad (3.1)$$

The summation represents the expected length of encoding an input string of length n by f . The division by n makes this represent the expected length of the encoding of one source symbol.

3.2 Introduction to Information Theory

C. E. Shannon [48], founder of the field of information theory, introduced an important quantitative measure of information. He defined the *self-information* of an event as follows: Suppose we have a sample space S , and suppose A is an event in S that can occur with probability $\mathbf{Pr}[A]$. Then the self-information associated with event A occurring is

$$\log_x \frac{1}{\mathbf{Pr}[A]} = -\log_x \mathbf{Pr}[A].$$

The logarithm is typically taken base 2, making the units of information *bits*.

The *entropy* of a system is simply the expected amount of self-information in an experiment. That is, suppose (A_i) are independent events of sample space S such that

$$\bigcup_i A_i = S.$$

Then the entropy of S , denoted $H(S)$, is given by

$$H(S) = - \sum_i \Pr[A_i] \log_2 \Pr[A_i].$$

If $\Pr[A_i] = 0$ in the above definition, then we omit it from the calculation, in effect defining that $0 \cdot \log_2 0 = 0$. This will also be true of the entropy definitions below.

3.2.1 Entropy of Sources

In the context of entropy coding, we are concerned with coding sources that generate an infinite number of symbols from some input alphabet Σ . The entropy of source S is defined as follows:

$$H(S) = \lim_{n \rightarrow \infty} \frac{1}{n} \left(- \sum_{s \in \Sigma^n} \Pr[s] \log_2 \Pr[s] \right) \quad (3.2)$$

The *first-order entropy* of a source S that generates strings $s_1 s_2 s_3 \dots$ is given by

$$H(S) = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n \left(- \sum_{x \in \Sigma} \Pr[x = s_i] \log_2 \Pr[x = s_i] \right). \quad (3.3)$$

The first-order entropy of a source (equation 3.3) is simply the average of the entropy of generating each source symbol independently; it represents a simple approximation to the entropy of the source (equation 3.2). When the symbols of a source are independent, the first-order entropy is the same as the entropy. In general, the first-order entropy is an upper bound on the entropy of a source. Intuitively, this is because independently generated symbols have more self-information than symbols that have dependencies. With dependent symbols, learning the values of some symbols makes the values of other symbols easier to predict, meaning there is less overall information content.

The typical model for a source is one in which the symbols generated are all independently and identically distributed (i.i.d.), these sources are known as *i.i.d. sources*. Note that for i.i.d. sources, the first-order entropy is equivalent to the entropy. For an i.i.d. source S , the entropy can be calculated as:

$$H(S) = - \sum_{a \in \Sigma} \Pr[a] \log_2 \Pr[a]. \quad (3.4)$$

One of the important contributions made by Shannon was showing that the entropy of a source is a measure of the average number of bits needed to encode one source symbol. In fact, Shannon proved that for any encoding scheme f on source S , the bit-rate $R(f, S)$ satisfies $R(f, S) \geq H(S)$.

3.3 Examples of Entropy Codes

3.3.1 Simple Example

As an example, suppose we want to encode an i.i.d. source S that generates three symbols, a , b , and c , where each symbol is generated independently with probability $\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{4}$ of occurring, respectively. Since a occurs more frequently, we encode it with the string 0. We encode occurrences of b and c with 10 and 11, respectively. We use f to denote our simple encoding scheme.

Bit-rate of Example

We use equation 3.1 to calculate the bit-rate. Starting with $n = 1$, we get our first approximation to the bit-rate as

$$\sum_{s \in \Sigma} \Pr[s] |f(s)| = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot 2 = 1.5.$$

Since the symbols occur independently, we see that for $n > 1$,

$$\begin{aligned}
 \sum_{\mathbf{s} \in \Sigma^n} \Pr[\mathbf{s}] |f(\mathbf{s})| &= \sum_{\mathbf{s} \in \Sigma^{n-1}} \sum_{s_n \in \Sigma} \Pr[\mathbf{s}s_n] |f(\mathbf{s}s_n)| \\
 &= \sum_{\mathbf{s} \in \Sigma^{n-1}} \sum_{s_n \in \Sigma} \Pr[\mathbf{s}] \Pr[s_n] (|f(\mathbf{s})| + |f(s_n)|) \\
 &= \sum_{\mathbf{s} \in \Sigma^{n-1}} \sum_{s_n \in \Sigma} \Pr[\mathbf{s}] \Pr[s_n] |f(\mathbf{s})| + \sum_{\mathbf{s} \in \Sigma^{n-1}} \sum_{s_n \in \Sigma} \Pr[\mathbf{s}] \Pr[s_n] |f(s_n)| \\
 &= \sum_{\mathbf{s} \in \Sigma^{n-1}} \Pr[\mathbf{s}] |f(\mathbf{s})| \sum_{s_n \in \Sigma} \Pr[s_n] + \sum_{s_n \in \Sigma} \Pr[s_n] |f(s_n)| \sum_{\mathbf{s} \in \Sigma^{n-1}} \Pr[\mathbf{s}] \\
 &= \sum_{\mathbf{s} \in \Sigma^{n-1}} \Pr[\mathbf{s}] |f(\mathbf{s})| + \sum_{s_n \in \Sigma} \Pr[s_n] |f(s_n)|.
 \end{aligned}$$

By induction, this implies that

$$\sum_{\mathbf{s} \in \Sigma^n} \Pr[\mathbf{s}] |f(\mathbf{s})| = n \sum_{s \in \Sigma} \Pr[s] |f(s)|.$$

This shows that the bit-rate of our system by equation 3.1 is simply

$$R(f, S) = \lim_{n \rightarrow \infty} \frac{n}{n} \sum_{s \in \Sigma} \Pr[s] |f(s)| = 1.5.$$

Comparison to Entropy

The entropy of the source S can be calculated by equation 3.4. We get

$$H(s) = -\left(\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{4} \log_2 \frac{1}{4} + \frac{1}{4} \log_2 \frac{1}{4}\right) = -\left(-\frac{1}{2} - \frac{1}{2} - \frac{1}{2}\right) = 1.5.$$

By Shannon's theorem, our encoding function f happens to be optimal for the source since the bit-rate of f equals the entropy of the source.

3.3.2 Block Codes

One important class of codes is defined by a one-to-one mapping f' from a finite set I of input strings to a finite set O of binary strings. Strings in set I are called *input messages*, and string in set O are called *output codewords*. To encode a string $\mathbf{s} \in \Sigma^*$, \mathbf{s} is parsed into

a sequence of input messages, and each input message is mapped via f' to form a sequence of output codewords. The encoding is the concatenation of the output codewords. Note that care must be taken when defining I and O so that it is possible to correctly encode any string, and so that any encoding can be correctly decoded.

If all the input messages are of the same length, and all output codewords are of the same length, then the code is known as a *block-to-block* (B-B) code. If all input messages are of the same length but output codeword lengths vary, then the code is known as a *block-to-variable-length* (B-V) code. In contrast, *variable-length-to-block* (V-B) codes and *variable-length-to-variable-length* (V-V) codes are those where the input messages vary in length. V-B codewords all have the same length, while V-V codewords can vary in length.

Tunstall [55] proves that the bit-rate $R(f', S)$ of block code f' on an i.i.d. source S can be simply calculated as follows:

$$R(f', S) = \frac{\sum_{\mathbf{o} \in O} \Pr[f'^{-1}(\mathbf{o})] |\mathbf{o}|}{\sum_{\mathbf{i} \in I} \Pr[\mathbf{i}] |\mathbf{i}|}. \quad (3.5)$$

This is basically the expected codeword length divided by the expected input message length. This method of calculation was used in [16]. Note that the simple example of section 3.3.1 was a simple B-V code.

3.3.3 Arithmetic Coding

Arithmetic codes work by allocating fewer output bits to symbols that occur more frequently. The basic idea is to represent every possible length m sequence of symbols by a fixed interval $[a, b)$ where $0 \leq a < b \leq 1$. The size of the interval $[a, b)$ is proportional to the probability that the corresponding sequence of symbols can occur. Encoding is achieved by choosing a number x in $[a, b)$ and writing down its sufficiently long prefix of x (in binary) so that x is known to be in $[a, b)$. Decoding is achieved by using x to identify $[a, b)$. For more details on practical implementations, see [46]. See [59] for an example of an adaptive arithmetic code. It can be shown that the bit-rate $R(f, S)$ for any arithmetic code f on an i.i.d. source S satisfies $R(f, S) = H(S)$.

3.3.4 Golomb Codes

Golomb codes were originally defined in [18] to compress sources that generate non-negative integers from a geometric probability distribution. Let S'_p be one of these sources, where integer i is generated independently with probability $(1 - p)p^i$. This source is conceptually equivalent to the binary i.i.d. source S_p where each bit b is generated to be 0 with probability p , in the following manner: Given S_p , form a sequence of non-negative integers (z_0, z_1, \dots) where z_0 is the number of zero bits between the beginning of S_p and the first one bit, and z_i is the number of zero bits between the i th one bit and the $(i + 1)$ th one bit in S_p . We can see that this generated sequence of non-negative integers has the same probability distribution as S'_p . Basically, run-length encoding a binary source S_p produces a geometrically distributed source S'_p that has infinitely many symbols. Thus, source S_p can be efficiently compressed via run-length encoding followed by Golomb codes; codes that use this process are called elementary Golomb codes.

The elementary Golomb code of order k (EG_k) is a simple V-V code which maps variable-length input messages into variable-length binary output codewords as follows: If k is a power of 2, then the input message 0^k is mapped to codeword 0, and the string 0^i1 for $0 \leq i < k$ is mapped to a codeword of length $1 + \log_2 k$. When k is not a power of 2, the mapping is only slightly more complicated: 0^k is still mapped to codeword 0. The string 0^i1 for $0 \leq i < k'$ is mapped to a codeword of length $1 + \lfloor \log_2 k \rfloor$, and the string 0^i1 for $k' \leq i < k$ is mapped to a codeword of length $1 + \lceil \log_2 k \rceil$, where $k' = 2^{\lfloor \log_2 k \rfloor + 1} - k$. As an example, table 3.1 shows the code EG_8 . Rice [44] independently discovered elementary Golomb codes of order 2^i for integral i ; thus these codes are also known as Golomb-Rice codes.

By definition, the concatenation of input messages of a block code can generate any infinitely long string; this ensures that any source generating infinitely many symbols can be parsed into input messages. However, when block codes are used to encode strings of finite length, the string may not be representable by a concatenation of input messages; for

Table 3.1: Elementary Golomb code of order 8 (EG_8).

<i>input message</i>	<i>output codeword</i>
00000000	0
00000001	1000
0000001	1001
000001	1010
00001	1011
0001	1100
001	1101
01	1110
1	1111

example, the string 100 cannot be represented by a concatenation of input messages from EG_8 . However any finite string can be represented in the form $s_1s_2 \cdots s_n$, where each s_i for $i < n$ is an input message (s_n may not be an input message). If the string to be encoded is of this form where s_n is not an input message, then some alternate method must be used to code s_n . Since the bit-rate is defined in terms of the limiting behavior for infinitely long strings, the exact method of encoding s_n does not affect the bit-rate of the block code.

For the elementary Golomb codes, a simple method to encode any finite-length string would be to add a 1 to the end of any string to be encoded, and then use the elementary Golomb code to encode this slightly modified string. The ending 1 ensures that the input string is a concatenation of input messages. After decoding a string, the trailing 1 can be removed. Under this method, the input string 000 encoded using EG_8 would be coded with 1100, and the input string 00000000 would be coded with a 01111.

Note that Golomb codes are extremely simple computationally; they can be implemented by using simple table lookup operations. Also note that unlike previously defined codes, elementary Golomb codes can only be used when the input alphabet is a binary

alphabet.

Effectiveness of Golomb Codes

Gallager and Van Voorhis [17] studied the effectiveness of Golomb codes in terms of entropy, for geometrically distributed sources. If we recast their results to apply to elementary Golomb codes on binary i.i.d. sources, then they can be stated as follows: To minimize the bit-rate of the elementary Golomb code, the order k of the code should be the unique integer satisfying the following:

$$p^k + p^{k+1} \leq 1 < p^k + p^{k-1}. \quad (3.6)$$

We let $k^*(p)$ denote the unique integer k satisfying inequalities 3.6. If we pick the optimal order $k^*(p)$ for an i.i.d. source S_p , then the bit-rate $R(\text{EG}_{k^*(p)}, S_p)$ of the resulting elementary Golomb code satisfies $H(S_p) \leq R(\text{EG}_{k^*(p)}, S_p) < H(S_p) + 0.04$. It can also be shown that as $p \rightarrow 1$, $R(\text{EG}_{k^*(p)}, S_p) \rightarrow H(S_p)$.

Comparison with Arithmetic Coding

One advantage of Golomb coding over arithmetic coding is that it is computationally simpler. Although both Golomb codes and arithmetic codes are linear time algorithms in the size of the input, arithmetic codes are slower in practice. For each input bit is processed, arithmetic coding requires computing the new interval $[a, b)$ that will represent the previous input string extended with the new bit; this requires some arithmetic operations. In Golomb coding, only string comparisons are needed.

Since the bit-rate for an arithmetic coding system on an i.i.d. source can achieve the entropy of the source, it would appear to be more efficient than elementary Golomb codes. In practice, however, symbols from sources are rarely independent. Furthermore, typically adaptive codes are used in practice; these theoretical results apply only to the non-adaptive versions of the codes. Thus, one should keep in mind that the effectiveness of codes in practical applications such as image compression may not match up exactly with the theoretical

results.

Even if elementary Golomb codes perform worse than arithmetic codes in practice, elementary Golomb codes still have the advantage of being simpler and easier to implement. In addition, we would expect the amount of compression achievable by Golomb codes to be fairly close to that of arithmetic codes.

Adaptive Elementary Golomb Coding

Adaptive methods for Golomb codes must choose an appropriate value of k , the order of the code, when given the input sequence so far. A simple adaptive Golomb coding scheme for choosing k was defined by Langdon [29]. It starts out with $k = 1$. From then on, k adapts according to whether 0^k was found in the input. If so, double k ; otherwise, halve k (although k never drops below 1). This simple strategy was meant to adapt extremely quickly to the input source, as well as to be very inexpensive computationally. Note that the decoder can detect when the value of k changes based on the values of the bits already decoded.

A simple alternate adaptive strategy for choosing k is initially start out with $k = 1$, and to double k as long as no 1 bits are found. Once a 1 bit is found, use the ratio of the number of 1 bits seen so far to the total number of bits seen as an approximation for p , and choose the order to be $k^*(p)$. In this strategy, the decoder can still compute the proper value of k based on the values of the bits already decoded.

Chapter 4

GROUP TESTING

4.1 Group Testing Problem

Group testing identifies a few significant items out of a large pool of items. This field was first studied by Dorfman [13], who was trying to find an efficient method of identifying Army recruits who were infected with syphilis [14]. In this scenario, a laboratory blood test could detect the presence or absence of syphilitic antigen. Instead of testing the recruits individually, the blood samples of several men could be pooled together and tested. A lack of the antigen would imply none of these men has syphilis; its presence would imply at least one of these men has syphilis. If the percentage of infected recruits is small, then pooling the bloods samples can greatly reduce the required number of laboratory tests.

4.1.1 Problem Definition

The general group testing problem can thus be defined as follows: Given n items, s of which are significant, what is the best way to identify the s significant items? It is assumed that items can be identified as significant or insignificant only through *group tests*. A group test is the process of picking a subset of the n items and determining whether there is a significant item in that set. There are two possible outcomes of a group test on set K : either K is *insignificant* (meaning all items in K are insignificant), or K is *significant* (meaning there is at least one significant item in K).

There are many different fields of group testing, each defined by a different formulation of the group testing problem. In *combinatorial group testing*, it is assumed that s , the number of significant items, is known at the beginning. Furthermore, the goal is to minimize the

total number of group tests that can occur in a worst-case scenario. In *competitive group testing*, s is not assumed to be known. Furthermore, the goal is to minimize the *competitive ratio*. This is the worst-case ratio of the number of group tests required by a given algorithm compared to the number of tests needed by the best group testing algorithm that knows s in advance. Algorithms for combinatorial group testing and competitive group testing can be found in Du and Hwang's book [14]. In *probabilistic group testing* literature (see [51, 25]), a probability distribution is put over the set of items specifying each item's probability of significance, and the objective is to minimize the expected number of group tests.

4.1.2 Issues

The basic decision a group testing algorithm has to make is the number of items that should be tested together. On one hand, if a large set is found to be insignificant, then a lot of information is learned because all items in that set must be insignificant. However, if the large set is significant, then not that much information is learned: none of the items in the set is yet fully known, and the significant item(s) could be located anywhere in the set. In fact, when a small set is found to be significant, more information is learned than when a large set is found to be significant. This is because there is more uncertainty about the exact location of the significant item in the larger set.

To maximize the amount of information learned, one would like to perform group tests on large sets in case the set is insignificant. However, one would also like the sets tested to be small in case the set is significant. The actual test size must be chosen to balance these conflicting goals; intuitively, the size is chosen so that the amount of information one would learn from finding an insignificant set is about the same as the amount learned from finding a significant set.

Group testing will be much better than individual testing when the ratio between significant items and insignificant items is small. With a small ratio, it is possible to test larger sets and mark many items as insignificant with just one test.

4.1.3 Applicability

Group testing has been found to be widely applicable to problems in many different areas. Besides screening for disease, applications have been found in areas as diverse as industry (for identifying defective Christmas lights), graph theory, and fault tolerant computing. In fact, group testing can also be used for data compression, if we view compression in the following manner:

Given a binary bitstream as input, think of each input bit as an item, where 1's represent significant items, and 0's represent insignificant items. Now let an encoder use a group testing algorithm to identify the significant items (the 1's in the bitstream). As output, it would send binary bits representing the outcomes of the group tests performed: 0 for an insignificant set, and 1 for a significant set. As long as the method of choosing the input bits for each group test is deterministic, a decoder could infer the input bitstream based solely on the output.

Now that we have an encoding/decoding scheme, the obvious next step is to determine the compression performance of this scheme. Since group testing algorithms try to minimize the number of group tests, in our scenario, that is equivalent to minimizing the number of bits in the encoder's output. Since this is exactly what a good compression scheme should do, we expect good group testing algorithms to also be good at compressing data.

4.2 Group Testing Algorithms

4.2.1 Hwang's Generalized Binary Splitting Algorithm

One of the first group testing algorithms was proposed by Hwang in 1972 [24]. It solved the combinatorial group testing problem, where, as input, the algorithm is given n items and the knowledge that s of these n items are significant. The algorithm identifies the s significant items by repeatedly performing the *binary splitting* procedure, a well-known technique for identifying one significant item out of a set. We first describe binary splitting,

followed by Hwang's algorithm.

Binary Splitting

The binary splitting procedure (procedure 4.1) works on an input set K , with K assumed to be significant (so that at least one item in K is significant). It uses a recursive, binary-search-like process to identify one significant item in K using $\lceil \log_2 |K| \rceil$ tests. In addition to finding one significant item x , the binary splitting process will also identify a set I of insignificant items, where $I \subset K$. Thus, the set $K - (I \cup \{x\})$ contains those items whose significance status remains unknown after the procedure.

Procedure 4.1 Binary Splitting

Initially, K is the significant set given as input, and $I = \emptyset$ represents the items identified as insignificant.

- Base Case: $|K| = 1$. One significant item has been found. Return it along with I .
 - Inductive step: $|K| > 1$.
 1. Partition K into two sets K_1 and K_2 , where there are no more than $2^{\lceil \log_2 |K| \rceil - 1}$ items in either set.
 2. Perform a group test on K_1 .
 - If K_1 is significant, then recursively look for a significant item in K_1 .
 - If K_1 is insignificant, then we know K_2 must be significant. Set $I \leftarrow I \cup K_1$ and recursively look for a significant item in K_2 .
-

In step 1, the binary splitting algorithm divides K into two smaller sets K_1 and K_2 . Note that this step has been intentionally incompletely specified in that exact sizes for sets K_1 and K_2 are not given. In the worst case, it will take $\lceil \log_2 \max(|K_1|, |K_2|) \rceil$ tests to find a significant item. Thus, as long as K is partitioned into sets of size at most $2^{\lceil \log_2 k \rceil} / 2$, binary splitting will still have the same worst-case behavior. Thus, from the perspective of solving the combinatorial group testing problem, the exact sizes, as long as they are within

the constraints given in step 1, are not relevant. The special case of the binary splitting method where K is split into two sets of size $\lfloor |K|/2 \rfloor$ and $\lceil |K|/2 \rceil$ is known as *halving*.

Hwang's Generalized Binary Splitting

It is easy to see that s applications of binary splitting on the input set of size n will find all s significant items, and will cost at most $s \lceil \log_2 n \rceil$ group tests. Clearly this method is not optimal because we are requiring each significant item to take roughly $\log_2 n$ group tests to find. If we performed binary splitting on groups of size $k < n$, and we expected there to be at least one significant item in each group of size k , then we could get away with finding all significant items with about $s \log_2 k$ total group tests. This could be significantly smaller than $s \lceil \log_2 n \rceil$. Roughly speaking, this is the main idea behind Hwang's generalized binary splitting algorithm (procedure 4.2). In this algorithm, Hwang chose k to be roughly n/s ; on average, there exists one significant item in every n/s items. Note that k is chosen adaptively as the significance status of more items are determined.

We say a *group iteration* of size k is the process of performing a group test on a set K of size k , and then, if necessary, performing binary splitting to identify a significant item in K . This is basically one application of step 3 in procedure 4.2. Generalized binary splitting simply performs group iterations repeatedly until all significant items are found.

The choice of the group iteration size k and the associated set K of size k can drastically affect the effectiveness of the group tester. As mentioned previously in section 4.1.2, choosing k involves trading off using small k to quickly identify significant items and using large k to quickly identify insignificant items. In generalized binary splitting, k is chosen to be a power of 2 to simplify the binary splitting. Note that when roughly half the unidentified items are significant, $k = 1$ and individual testing is performed. With so many significant items, testing groups is not worthwhile. In the other case when less than half the unidentified items are significant, k is chosen to be roughly $|U|/\ell$ rounded to a power of 2, where U is the set of remaining unidentified items, and ℓ is the number of remaining significant items. This number is approximately the size for which you would expect to

Procedure 4.2 Hwang's Generalized Binary Splitting

We are given n items, s of which are known to be significant.

Let U , S , and I , represent the current set of unidentified items, significant items, and insignificant items, respectively.

Initially, U is the initial set of n items, $S = \emptyset$, and $I = \emptyset$.

While $|S| \neq s$ repeat the following:

1. Let $\ell = s - |S|$ be the number of significant items remaining left to be found.
2. Let $k = \begin{cases} 1, & \text{if } |U| \leq 2\ell - 2, \\ 2^\alpha, \text{ where } \alpha = \lfloor \log_2 \frac{|U| - \ell + 1}{\ell} \rfloor, & \text{otherwise.} \end{cases}$
3. Pick a set K of k items from U , and test K for significance.
 - If K is insignificant, then set $U \leftarrow U - K$, and $I \leftarrow I \cup K$.
 - If K is significant, then perform binary splitting on K to identify one significant item x , as well as a set I' of insignificant items. Set $U \leftarrow U - (\{x\} \cup I')$, and $I \leftarrow I \cup I'$.

$I \leftarrow I \cup U$. (The remaining items in U are insignificant because after exiting the while loop, all significant items have been found).

find one significant item.

Hwang proved the following performance bound on his generalized binary splitting algorithm [14]:

Theorem 1 *Let $M(s, n)$ be the worst-case number of group tests required by the generalized binary splitting algorithm when given n total items, s of which are significant. Then*

$$M(s, n) = \begin{cases} n, & \text{for } n \leq 2s - 2, \\ (\alpha + 2)s + p - 1, & \text{for } n > 2s - 2, \end{cases}$$

where $\alpha = \lfloor \log_2(\frac{n-s+1}{s}) \rfloor$, and $p < s$ is a non-negative integer uniquely defined in the equation $n - s + 1 = 2^\alpha s 2^\alpha p + \theta$, for $0 \leq \theta < 2^\alpha$.

Group Testing Example

Consider a binary source with 1000 bits, 80 of which are known to have value 1 (i.e., 80 items are significant). If we apply procedure 4.2, then the initial group iteration size would be $k = 8$ via the formula in step 2. For this example, we assume that when k items are chosen, they are the first k bits in the source that are not yet coded. We also assume that whenever a set K is partitioned into two sets, the first set tested (K_1) will contain the bits of the first half of K , in the order that the bits were in the input source.

We will first examine two cases, one where the first 8 input bits are 00000000 and the other where they are 00000110. We will use ? to denote bits whose value are not yet coded. Figure 4.1 shows the sequence of group tests performed, and the resulting output bits for the two example cases.

We can calculate the output produced by a group iteration of size 8 by examining all possible cases; the result is shown in table 4.1. If the 8 input bits are all 0, then one group test (and thus one output bit) is used to describe all 8 bits. In all other cases, the first output bit signifies that the set of input bits is significant, and the remaining 3 output bits indicate the position of the first significant input bit. The value of the input bits represented with a

State of Input	Output Bit
Case I: <div>????????</div> 00000000	0
Case II: <div>????????</div> <div>????</div> ???? 0000 <div>??</div> ?? 0000 <div>?</div> ??? 000001??	1 0 1 0

Figure 4.1: Example showing group tests performed. Each box represents a group test of the items inside the box.

Table 4.1: Code associated with a group iteration of size 8.

<i>Input bits</i>	<i>Output bits</i>
00000000	0
00000001	1000
0000001?	1001
000001??	1010
00001???	1011
0001????	1100
001?????	1101
01??????	1110
1???????	1111

'?' will not affect the output bits sent. Note that the value of these bits will remain unknown after one group iteration; their value must be determined in a subsequent group iteration. In contrast, the input bits represented with '0' will be coded (known to be insignificant) after one group iteration.

Generalizing from this example, we conclude that for a group iteration of size k , if the chosen set K consists of k 0's, then one 0 is output. When K contains a 1, then a 1 is output, followed by a $\log_2 k$ length binary integer representing the location of the first 1 in the set K .

4.3 Other Group Testing Algorithms

4.3.1 Doubling

The doubling algorithm of Bar-Noy *et al.* [4] solves the competitive group testing problem, where the number of significant items s is unknown. This algorithm is the same as Hwang's algorithm, except that the method of choosing the group iteration size k is different. In the doubling strategy, k starts at 1, and doubles after every group test. It continues doubling as long as no significant items are found, and resets to 1 once a significant item is found. To quickly handle the case where there are no significant items, one additional group test of the entire set of n items is performed at the beginning.

Recall that the goal of competitive group testing does not directly minimize the number of group tests, but instead tries to minimize the competitive ratio. Although it is not trivial to prove, this algorithm was shown in [14] to have a competitive ratio of 2. This means that the number of group tests that this algorithm performs is no more than a factor of 2 times greater than the number of group tests that the optimal algorithm that knew the exact number of significant items would have performed.

4.3.2 *Jumping*

The Jumping algorithm [15] is an extension of the doubling algorithm, and was shown to have a competitive ratio of 1.65. It works by performing group iterations of size $1 + 2, 4 + 8, \dots, 2^i + 2^{i+1}$ until a significant item is found. After a significant item is found, all remaining unidentified items are tested together in one group test to handle the case when no more significant items remain. If some significant items remain, then the algorithm resets, and follows its initial procedure on the set of remaining unidentified items. This means it performs a group iteration of size $1 + 2 = 3$, followed by one of size $4 + 8 = 12$, and so on, until another significant item is found. This process repeats until all items are identified.

Chapter 5

GROUP TESTING AND COMPRESSION

In this chapter, we describe the deep connections between group testing and data compression. In particular, we show how Hwang's generalized binary splitting algorithm generates an elementary Golomb code. By making this relation, we can state some of the known results about the effectiveness of Golomb codes and apply them to group testing.

We also show how zerotree coding can be thought of as a special case of group testing. This realization led us to investigate the use of group testing as a basis for image compression.

We end this chapter with a discussion about how group testing can be used in the field of image compression. This chapter provides structure to the rest of this thesis by overviewing the content of the rest of the chapters in this thesis.

5.1 Group Testing and Entropy Coding

As described in section 4.1.3, any group testing algorithm can be used as a method for compressing a stream of bits. Thus, group testing can be viewed as a standard entropy code that works on a binary alphabet.

One question that arises is whether combinatorial, competitive, or probabilistic group testing algorithms would be more suitable for data compression. In the combinatorial group testing problem, it is assumed that the number of significant items in the initial set is known before the algorithm starts. Although this is usually not true in the context of data compression, it is possible for the encoder to send the number of significant items (as side information) to the decoder at the beginning of the bit stream. With this method, any combinatorial

group testing algorithm can be used as an entropy code.

In competitive group testing, the number of significant items is not known at the beginning; this more accurately models the data compression scenario. This scenario is similar to the adaptive entropy coding methods in that the algorithm tries to adaptively estimate the number of remaining significant items as bits are encoded. However, the competitive ratio metric used (see section 4.1.1) in this problem does not directly correspond to minimizing the number of group tests.

The probabilistic group testing framework seems most applicable to data compression since its goal is exactly the same as the goal of an entropy coder: namely, to minimize the expected number of tests required. Of course, if the probability distribution of the data to be compressed is known only to the encoder, then this information must also be sent as side information to the decoder.

5.1.1 Relation to Golomb Codes

If we look more closely at Hwang's generalized binary splitting method (section 4.2.1), we can see a deeper connection between group testing and Golomb codes. In fact, a group iteration of size k is essentially EG_k , the elementary Golomb code of order k (presented in section 3.3.4). If we ignore the '?'s in table 4.1, we see that this table showing the code corresponding to the group iteration is exactly the same as the EG_8 code presented previously in table 3.1.

5.1.2 Effectiveness of Group Testing

As explained in chapter 3, the effectiveness of an entropy code can be characterized by comparing its bit-rate with the entropy of the source. Since group testing is an entropy code, we can calculate the bit-rate for group testing on simple sources and compare it with the entropy of the source. Because of the close relation between group testing and elementary Golomb codes, our task is vastly simplified. Let us first consider the binary i.i.d. source

model where each source bit has value 0 with probability p , and 1 with probability $1 - p$. By relating group testing to Golomb codes, we learn that the optimal group iteration size is $k^*(p)$ (see section 3.3.4). This means it is the unique integer k satisfying the inequalities 3.6.

We will use $GTR(p)$ to refer to the group testing bit-rate on source S_p using the optimal group iteration size $k^*(p)$. To measure the efficacy of group testing, we can compare this bit-rate to the entropy of the S_p . The entropy of S_p can simply be calculated using equation 3.3; $H(S_p) = -p \log_2 p - (1 - p) \log_2(1 - p)$. A method for calculating the group testing bit-rate $GTR(p)$ will be presented in chapter 9. Figure 5.1 compares the group testing bit-rate $GTR(p)$ with the entropy of the source $H(S_p)$ for various values of p . As the graph shows, the group testing bit-rate is always within 5% of entropy, and usually much lower.

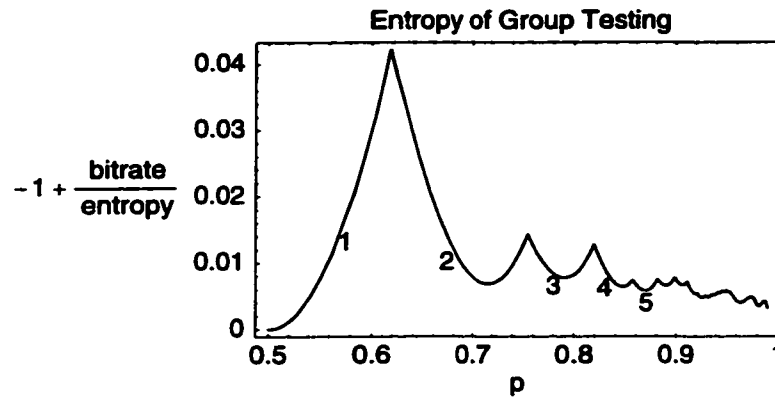


Figure 5.1: Compares group testing bit-rate to entropy, on an i.i.d. source where p is the probability that a source bit is 0. Numbers on the graph indicate the optimal group iteration size $k^*(p)$ for that section of the curve.

5.1.3 Exact Partitioning Method

Note that when the group iteration size k is not a power of 2, then it is not immediately obvious how binary splitting (procedure 4.1) should partition the initial set of K of size k

into two sets K_1 and K_2 to minimize the expected number of bits required to code the group iteration. Although the exact partitioning method is irrelevant in the combinatorial group testing problem (see section 4.2.1), it becomes relevant when our metric is to minimize the expected number of group tests.

We now proceed to define the optimal binary splitting procedure. This procedure is exactly like the binary splitting procedure, except that the exact partitioning method is defined to minimize the expected number of group tests. The simple and elegant halving method (see section 4.2.1) turns out not to be the optimal binary splitting method. The optimal method is to partition K into two sets of size i and $k - i$ where i is a power of two and $k - i$ is as close to i as possible. Furthermore, the set of smaller size should be tested first. Making this choice in the binary splitting is the only way to ensure that group testing with group iteration size k is the same as an elementary Golomb code of order k for all values of k . The reason for this choice is explained below.

The example in section 4.2.1 shows that group testing with group iteration size k is a V-V code (defined in section 3.3.2) where the input sequence 0^k is coded with 0, and for $0 \leq j < k$, $0^j 1$ is coded with a codeword that starts with a 1. Define $l = \lfloor \log_2 k \rfloor$. If k is a power of 2, then $l + 1$ is the length of the codeword for the input sequence $0^j 1$. When k is not a power of 2, then of the k input sequences of the form $0^j 1$, $k' = 2^{l+1} - k$ of them will be coded with a codeword of length $l + 1$, while $k - k'$ of them will be coded with a codeword of length $l + 2$. Choosing the sizes of K_1 and K_2 corresponds to deciding which of the input sequences $0^j 1$ will have longer codewords than the others. Clearly, we want the less probable input sequences to have longer codewords, and more probable input sequences to have shorter codewords. If p is the probability that a source bit is zero, then the probability of input sequence $0^j 1$ occurring is $p^j(1 - p)$. This implies that if $j < k'$, we should code $0^j 1$ with a codeword of length $l + 1$, and that the other input sequences should be coded with codewords of length $l + 2$.

We can use a binary tree to represent the sequence of group tests performed by a group iteration of size k . The interior nodes of this tree represent group tests, with left

Procedure 5.1 Optimal Binary Splitting

Initially, K is a significant set input, and $I = \emptyset$ represents the items identified as insignificant.

- Base Case: $|K| = 1$. One significant item has been found. Return it along with I .
 - Inductive step: $|K| > 1$.
 1. Let $\ell \leftarrow 2^{\lfloor \log_2(|K|/2) \rfloor}$, $h \leftarrow 2^{\lceil \log_2(|K|/2) \rceil}$. (ℓ and h are the powers of two just smaller and just larger than $|K|/2$, respectively)
 2. If $|K|/2 - \ell \leq h - |K|/2$ then $k_0 \leftarrow \ell$ else $k_0 \leftarrow h$. (Find the closest power of 2 to $|K|/2$.)
 3. Let $k_1 = \min(k_0, |K| - k_0)$. Let $k_2 = \max(k_0, |K| - k_0)$. (Make k_1 the smaller one.)
 4. Partition K into two sets K_1 and K_2 , where $|K_1| = k_1$ and $|K_2| = k_2$.
 5. Perform a group test on K_1 .
 - If K_1 is significant, then recursively look for a significant item in K_1 .
 - If K_1 is insignificant, then we know K_2 must be significant. Set $I \leftarrow I \cup K_1$ and recursively look for a significant item in K_2 .
-

and right branches corresponding to significant and insignificant test results, respectively. Left branches are labeled with a 1 to signify the output bit sent due to a significant result; similarly, right branches are labeled with a 0. The group testing algorithm starts at the root node and follows the branches downward according to the group testing results. Eventually it will reach a leaf, signifying that the group iteration is finished, and that that part of the input bits have been identified. The leaves are labeled with the bits from the input that have been identified. This tree is illustrated in figure 5.2.

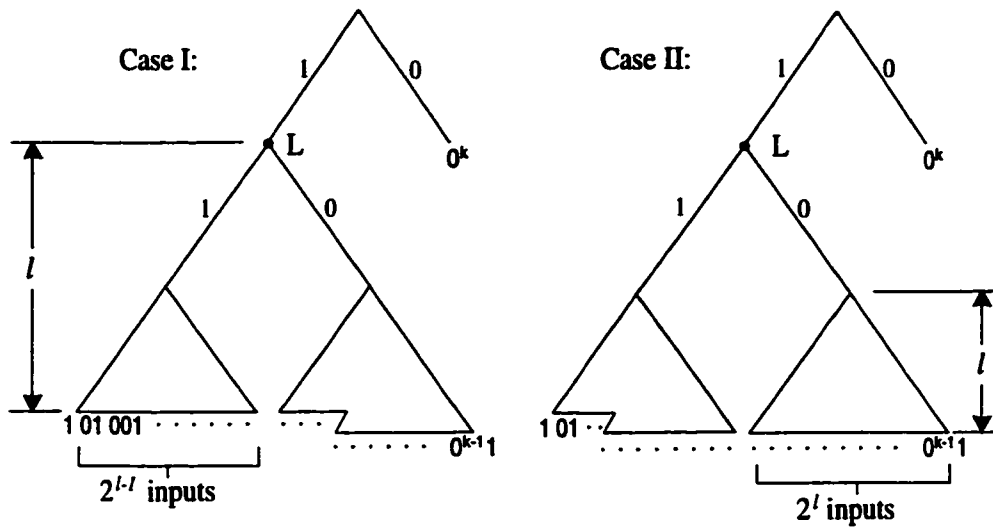


Figure 5.2: Binary tree representing the V-V code produced by one group iteration. Edge labels correspond to output bits; leaves correspond to the input sequence that was coded on that branch.

Note that the root node represents the initial group test of size k . Thus, if the first group test result is insignificant, then the k input bits have been identified to be 0^k , and this is the label at the leaf node located at the right branch of the root node. Also note that the size of each group test is equal to the number of leaves in the subtree rooted at that node.

Let node L be the left child of the root node, and consider the subtree rooted at this node. In this subtree, the left-most leaf represents a series of group tests where the group tested was always significant. This corresponds to where the input sequence was 1. The

right-most leaf of this subtree corresponds to a series of group tests where each group tested was insignificant. This corresponds to the input sequence $0^{k-1}1$. In fact, the leaves at the bottom of the tree represent the input sequence $1, 01, 001, \dots, 0^{k-1}1$ when read from left to right. Furthermore, the depth of a leaf represents the length of the associated output codeword. To make the more probable input sequences have shorter codewords, we need to construct the tree where the first k' leaves are at depth $l + 1$, and where the rest of the leaves are at depth $l + 2$.

Recall that in the binary splitting algorithm, we split set K of size k into two sets K_1 and K_2 , respectively of size k_1 and k_2 , and test set K_1 first. If K_1 is significant, then we will have to identify a significant item from a set of k_1 elements; if K_1 is insignificant, we will have k_2 elements from which to identify a significant item. This implies that the left child of node L has k_1 leaves, and that the right child has k_2 leaves. To optimize the depths of the leaves for coding performance, either the left child of node L should be a complete binary tree (case I of figure 5.2), or the right child of node L should be a complete binary tree (case II). This implies one of K_1 or K_2 must have a power of 2 number of elements. Furthermore, we also need to test the smaller set first so that number of elements on the left branch of node L will be less than the number of elements on the right branch. This is the reason behind the method of partitioning in the optimal binary splitting procedure.

5.2 Group Testing and Zerotree Coding

Another way to characterize group testing is by viewing it as a generalization of zerotree coding. Recall that zerotree coding tests trees of coefficients for significance, where trees are significant only when some coefficient in that tree is significant. Testing a tree for significance is exactly the same as performing a group test on the coefficients in that tree. The significance pass of any zerotree coding algorithm can be thought of as a group tester that identifies the significant coefficients of a given bit-plane. Group testing is more general than zerotree coding because the groups that it tests together are not restricted to be trees,

but can be arbitrary collections of coefficients.

5.3 Group Testing and Image Compression

Up to this point, we have been explaining what has been previously done in image compression, and how group testing relates to previous image compression techniques. Now we proceed to discuss the best methods of applying group testing to image compression. We start by describing how previous image compression algorithms can be thought of as using group testing. We then discuss some of the issues that arise when applying group testing to image compression, and then talk about some of the general techniques that can be used to develop a group testing-based image coder.

5.3.1 Previous Work

Based on section 2.6, it is clear that a form of group testing has already been used in image compression. For instance, Ordentlich, *et al.* [40] and Malvar [34] both use elementary Golomb codes to effectively code wavelet coefficients, which can be thought of as a group testing technique. Furthermore, the many zerotree coding algorithms that have been used for image compression can also be thought of as using group testing. Thus, we can say that a significant number of wavelet coding techniques essentially rely on group testing. Note that these previous techniques use group testing to code the bits in the significance pass. Coding these significance pass bits appears to be where group testing is most useful. In fact, these bits can be directly coded with any group testing algorithm.

5.3.2 Mismatch Between Source Models

An obvious way of using group testing in the significance pass is to simply take all the transform coefficients in order and code them with a group testing coder. Unfortunately, directly applying group testing in this manner may result in a coder with poor PSNR performance. This is because the group testing algorithms that have been studied do not take

into account the typical statistical characteristics of the transform coefficients of images.

The combinatorial group testing problem assumes that nothing is known about the initial items. This is similar to assuming that all coefficients are equally probable to be significant, and that all coefficients are independent of each other. As explained in section 2.5, neither of these facts is strictly true in our application for wavelet coefficients.

For example, we know that one reason zerotree coding compresses efficiently is that the coefficients can be organized into trees that usually obey the zerotree property. Since the group testing algorithms defined in the literature do not know about the zerotree property, the algorithm cannot be the most effective coder for its input. The same reasoning can be applied for other statistical characteristics of wavelet-transformed coefficients, such as the dependence upon neighbors.

Unfortunately, the probabilistic group testing problems that have been studied so far also suffer from this mismatch. To the best of our knowledge, all probabilistic group testing problems deal only with the case where the probability of significance of each input item is independent from all other items. This is not true for the transform coefficients.

5.3.3 Making the Sources Match

One way to surmount this difficulty is to make the characteristics of our input to group testing match with what the group testing algorithm expects. To do this, we can organize all the wavelet coefficients into *classes* which tend to have characteristics that match what the group testing algorithm expects. Each class could then be coded with a different group testing coder. Since different classes have different characteristics, we should use different group testers for each class.

To illustrate this approach, we can use Hwang's generalized binary splitting algorithm as our basic group testing coder. We can think of this algorithm as expecting its input items to be probabilistically independent of each other, as well as all having approximately the same probability of significance. To make this coder to work well, we would want to define the classes so that coefficients in a class satisfy these two characteristics as much

as possible. That is, we will try to make the coefficients in each class as independent as possible of each other, as well as approximately having the same probability of significance.

This is the basic approach we will explore in detail in chapters 6–8. In these chapters, we divide the transform coefficients into different classes, and then code each class with group iterations. Since the probability of finding a significant item will be different for different classes, each class will be coded with a different group tester based on group iterations. Each group tester will adaptively choose its own group iteration size based on the estimated probability of significance for its coefficients. Since the classes will have different probabilities of significance, they also will use different group iteration sizes.

We will apply this group testing method to a wide variety of different transforms, including the wavelet transform, wavelet packets, discrete cosine transform, and lapped transforms. Each of the different transforms will require a different definition of the classes in order for the coder to work well.

5.3.4 Defining New Group Testing Algorithms

Another completely different way to tackle the mismatch problem is to define new variants of the group testing problem that know about the statistical properties between the input items. Then we could proceed to define new algorithms for solving this problem; these new algorithms could then be directly applied to compressing the bits from the significance pass of an image coder. We now proceed to discuss some possible variants of group testing that could be useful for image compression.

Two-state Markov Source

In this problem, the input comes from a two-state Markov chain shown in figure 5.3. This chain has two probability parameters p and q , representing the probability of moving from state 0 to state 1 and from state 1 to state 0, respectively. Each step of the Markov chain defines the next bit of the source as follows: if the chain moves to state i , then the next bit

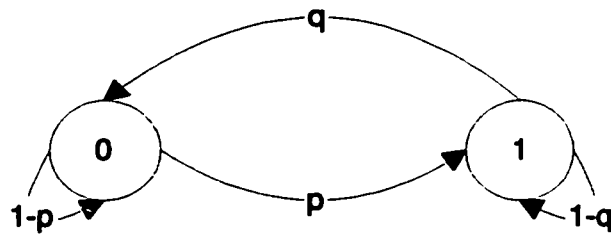


Figure 5.3: Markov-chain diagram of source.

is i . The goal of a group testing algorithm would be to minimize the expected number of group tests to find the significant items, when it is given n items generated by this Markov source. Although we are using this source as a simplified model of the significance of transform coefficients coded in the significance pass, this Markov chain has previously been used to model bit errors in a channel (the Gilbert model) as well as pixels of black and white images (the Capon model [6]).

Note that the bits generated by this source are not independent. If $p < \frac{1}{2}$ and $q < \frac{1}{2}$, then after seeing a 0, it is more likely that another 0 will follow; similarly, after seeing a 1 it is more likely that another 1 will follow. Modeling a dependence between source bits appears to be general problem that is potentially applicable in many realistic scenarios besides image compression. For instance, consider the use of group testing for identifying defective Christmas lights or beakers that leak. If we assume the items came out of a factory line, it may be plausible that the probability that an item is defective is not independent, but that if the machinery produced a defective item at time t , then it is more likely to also produce another defective item at time $t + 1$.

To measure the effectiveness of algorithms that solve this problem, we can compare the bit-rate of a group testing algorithm on this source with the entropy of the source. The entropy of this source and the bit-rates of several group testing algorithms for encoding this source are calculated in chapter 9.

More General Source Models

Of course, there are many more ways to model the statistical characteristics of wavelet coefficients. Unfortunately, the model in the previous section was a one-dimensional model in that items only depend on the neighbors that came before or after it in time. It would be nice to model statistical dependences in two-dimensions, since wavelet-transformed coefficients of images are inherently two-dimensional. One possible way of modeling the input is to use a Markov random field, an extension of Markov chains to handle multiple dimensions. Another possible source model for the wavelet-transformed coefficients is a generalized Gaussian distribution field, having slowly spatially varying variances; this is the model used in [30]. In these statistical models, minimizing the expected number of group tests seems to be the goal that most naturally applies when using group testing for image compression.

Adaptivity and Unknown Model Parameters

To accurately represent transform coefficients, a source model of an image usually has some model parameters that take on different values for different kinds of images. For example, if an image contains lots of smooth regions with the same value, we expect inter-pixel correlation to be fairly high. This may cause the coefficients in the transform domain to be more probabilistically dependent on their neighbors. In the other case, when there are not many smooth regions in an image (such as a picture of a crowd of people), we may expect the coefficients in the transform domain to be less dependent on their neighbors. In addition to being different on different images, these model parameters may also differ on different parts of a single image.

Unfortunately, the values of these model parameters for a particular image are not known until the image is analyzed. For an encoder to be able to take advantage of the model parameters and obtain more efficient compression, it must tell the decoder the model parameters. This can be done by encoding the parameters at the beginning of the compressed

image description. This will work well whenever the model parameters over the entire image can be succinctly described. Alternatively, an adaptive approach could be used to handle unknown model parameters. In this method, the model parameters are estimated, and as more of the image is encoded, the parameters estimates are updated. The algorithm would adapt according to the current estimate of the parameters.

Model Parameter Inquiry Problem

As a more structured way of tackling the problem of unknown parameters, we could expand our definition of the group testing problem to also include costs of inquiring about the model parameters. Given a model with some unknown parameters and a set of input items, there would be two possible types of operations: group tests, and model parameter inquiries. Group tests cost one bit, and model parameter inquiries cost k bits, where k depends on the desired precision for the model parameter. The goal would be to minimize the total cost of finding all significant items. The adaptive approach to encoding would use no model parameter inquiries; the non-adaptive approach would estimate the unknown parameters solely through the use of model parameter inquiries. A balanced approach that uses both methods has the potential to work better, although it makes the group testing problem more difficult to solve.

Chapter 6

GROUP TESTING FOR WAVELETS

This chapter explains the new Group Testing for Wavelets (GTW) algorithm [20], a wavelet-transform based image coder that uses group testing. Our work was motivated by seeing group testing as a generalization of zerotree coding. Thus, our image coder was created to determine whether using group testing instead of zerotree coding would make a significant performance improvement on actual images.

Our new coder shares many of the same techniques of previous zerotree coding algorithms, with the chief difference being that our significance pass replaces the previous zerotree coding method with a group testing method. In fact, GTW is exactly SPIHT with the significance pass replaced by a group testing method. Note that SPIHT has the best compression performance out of the zerotree coding techniques that use the wavelet transform. Thus, comparing the performance of GTW with SPIHT is really comparing group testing with the best zerotree coding method.

We start by explaining GTW in section 6.1. Next, we compare the rate-distortion performance of GTW with SPIHT in section 6.2. We then discuss some alternative approaches for image compression that slightly modify how the original GTW codes coefficients in section 6.3. We conclude in section 6.4 with a comparison to other image coders and a summary of our results.

This chapter shows that the rate-distortion performance of GTW is better than SPIHT (without arithmetic coding) and comparable to SPIHT-AC (with arithmetic coding). GTW is significantly better than SPIHT and close to SPIHT-AC over a large range of bit-rates. This is a significant result because GTW does not employ arithmetic coding. It shows that the simple group testing procedure can supplant the need for arithmetic coding in a practical

image coder.

6.1 The GTW Algorithm

6.1.1 Algorithm Overview

Our algorithm follows the standard transform coding process (shown in figure 2.3), where we first apply the DWT on the image data, and then code the transform coefficients. The transform coefficients are coded in a bit-plane by bit-plane fashion, with each bit-plane coded by a significance pass and a refinement pass. In the significance pass, our coder divides the coefficients into *GTW classes*. As was mentioned in section 5.3.3, the purpose of these classes is to separate coefficients with different probabilities of being significant. Each class is then coded with its own group tester that adapts to the probability of significance for that class. Our method of adapting is similar to previous adaptive Golomb coding work, but also relies on entropy arguments found in section 5.1.2.

Our division of coefficients into classes is similar to choosing different contexts for the coefficients, and coding each context with a different entropy coder. Thus, we must design our classes with the context dilution problem (explained in section 2.6.3) in mind. Having too many classes may result in poor adaptation to the classes' characteristics, because the number of coefficients belonging to a class may be too small. On the other hand, having many classes is beneficial in that coefficients in a class will be more likely to have similar characteristics, which leads to better overall coding performance. The number of classes must be chosen to balance the effectiveness of many classes with the ineffectiveness of very few coefficients in each class.

6.1.2 GTW Classes

We define the *GTW classes* based on the properties of typical wavelet transform coefficients of image data such as those presented in section 2.5.

The characteristics that distinguish between GTW classes are the *subband level*, the

significant neighbor metric, and the *pattern type*. We now proceed to describe these characteristics.

Subband Level

In a bit-plane of a wavelet-transformed image, the coefficients in lower frequency subbands are more likely to be significant. The lowest frequency subband counts as a subband level. There is one additional subband level for each level of the wavelet transform. Figure 6.1 shows the 4 subband levels when 3 levels of the wavelet transform are performed.

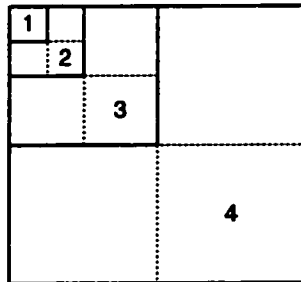


Figure 6.1: Subband levels in GTW. Solid lines separate subband levels; dotted lines separate subbands.

Significant Neighbor Metric

A coefficient in a bit-plane is likely to be significant if more of its *neighbors* are significant. A neighbor is defined to be the following: one of up to 8 spatially adjacent coefficients in the same subband, one of the 2 spatially identical coefficients in another subband at the same level, the parent coefficient in the next lower subband, or one of the 4 child coefficients in the next higher subband. The parent/child relations are exactly as defined in SPIHT. Figure 6.2 shows the neighbors of a coefficient.

There are 4 values in the significant neighbor metric; 0, 1, 2, and 3+; corresponding to whether 0, 1, 2, or more than 2 neighbors are significant, respectively. In counting the significant neighbors we count one for each spatially adjacent, spatially identically, and

parent coefficient, but just one for all 4 children coefficients. That is, if one or more children are significant then we add one to the count of significant neighbors. Thus, the maximum neighbor count is 12 even though there are 15 neighbors. In determining the significant neighbor metric for a coefficient in the current bit-plane, a neighbor can be known to be significant from the coding of a previous bit-plane or from the coding of the current bit-plane.

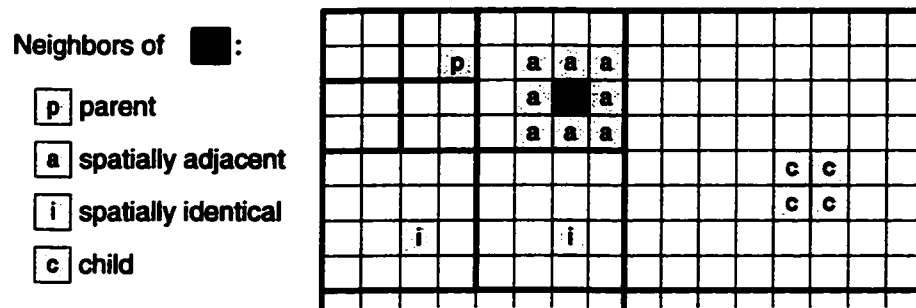


Figure 6.2: Illustration of neighbors.

We derived our significant neighbor metric from several considerations. We chose to limit the 4 children to count as at most one because the 4 children, taken together, represent one spatially equivalent spot in the next lower subband level. Also, knowing that 4 child coefficients are significant does not seem to bear 4 times as much information as knowing that one child coefficient is significant.

We reduce the resulting count of significant neighbors to four values for several reasons. One is to avoid having too many classes, and suffering from a context dilution problem. Another is the observation that once the metric for a coefficient reaches about 4, then the probability of significance for that coefficient is high enough that a group iteration size of one is sufficient. Inequalities 3.6 show that $k = 1$ when $p \leq (\sqrt{5} - 1)/2 \approx .62$, implying that when the probability of being significant is greater than about 0.38, a group size of 1 is optimal.

Pattern Type

Coefficients adjacent to each other are assigned different pattern types. The pattern type is based solely on position in a subband. In GTW, each coefficient belongs to one of 4 distinct pattern types as shown in figure 6.3.

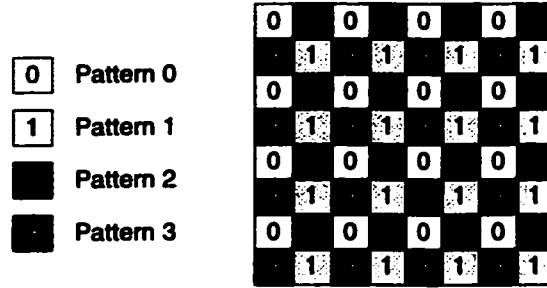


Figure 6.3: Illustration of pattern type.

One idea behind using the pattern type is to make coefficients in any class less likely to be correlated. This will make the coefficients more likely to be independent of each other, which is what Hwang's generalized binary splitting expects.

We can also view the pattern type as an attempt to control the order in which the information known about a coefficient's neighbors propagates. Let C_i represent the set of coefficients with pattern type i . If we assume all coefficients in C_i are coded before those in C_{i+1} , then figure 6.4 shows that a coefficient in C_{i+1} knows more information about its neighbors than in C_i . The greater amount of available information should make it easier to code C_{i+1} than C_i . We call this the *bootstrapping* effect. Controlling the coding order of the pattern type will change how accurately the coefficients in C_i are classified, and how well they are coded. This may help improve the net coding efficiency.

6.1.3 Class Ordering

A GTW class is defined by these three characteristics: subband level, significant neighbor metric, and pattern type. For our study we used 7 subband levels, 4 significant neighbor

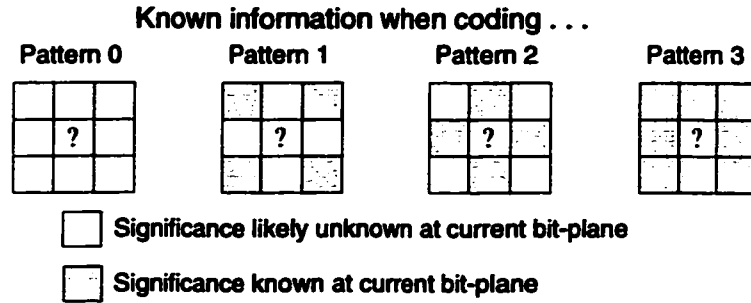


Figure 6.4: Illustration of bootstrapping.

metric types, and 4 pattern types for a total of 112 classes. With these specified classes, what is the best order in which to code them? From the perspective of generating an embedded bit stream, we should code the coefficients that contain the most information first. Since finding significant coefficients increases the fidelity of the image and finding insignificant coefficients has no effect on the fidelity, those coefficients more likely to be significant should be coded first.

Based on this heuristic, we should code the low subband level classes before the high subband level classes. We should also code the classes with more significant neighbors before those with fewer or no significant neighbors. After some testing (see section 6.3.3), we determined that the following ordering resulted in good rate-distortion performance:

GTW Ordering: Always code the classes with the greatest significant neighbor metric first. If there are several classes with the same significant neighbor metric, then code the classes with the lowest pattern type first. If there are classes with the same significant neighbor metric and pattern type, then code the class with the lowest subband level first.

One way to verify that this ordering works well is through figure 6.5. It is a scatter plot of the log of the group iteration size k for each class. Every group iteration performed is placed in a rectangular bin. Darker bins correspond to bins containing more elements.

Classes are ordered according to the GTW ordering; thus, the classes on the left of the plot are always coded before the classes on the right. This graph was created from the results of encoding the standard Barbara image to 1.0 bits per pixel with the GTW coder. It shows that the classes with the lower group iteration sizes tend to be coded first. The classes with low group iteration size also tend to contain those coefficients most likely to be significant.

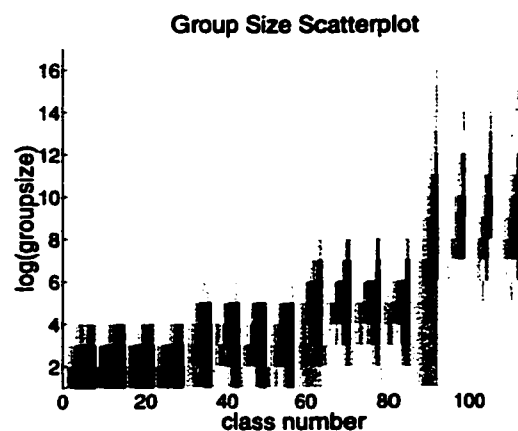


Figure 6.5: Group iteration size scatter plot for Barbara at 1.0 bpp.

Note that the ordering depends mostly on the significant neighbor metric because it happens to be the best predictor for when a coefficient is significant.

In addition to improving the image quality, finding significant coefficients quickly contributes to better classification of their neighbors. Once a significant coefficient is found, its neighbors could change classes because they now have one more significant neighbor. The new classes will be coded earlier than the old classes in which the coefficients were formerly contained. This has an effect of searching for clusters of significant coefficients because once a significant coefficient is found, then its neighbors are more likely to be coded next. If any of these neighbors were tested as significant, then the coefficients around them will also be more likely be coded earlier.

6.1.4 GTW Details

In our study we first use the Daubechies 9/7-tap filters [3] and perform enough levels of the wavelet transform so that the lowest frequency subband is of size 8×8 or smaller. On 512×512 images, this translates into 6 levels of wavelet transform, leading to 7 subband levels. Next, the average of the coefficients in the lowest frequency subband is subtracted from the lowest frequency subband before the transform coefficients are coded. This average is coded and sent in the first bits of GTW. After this step, the value of the maximum magnitude coefficient is coded so that the coefficients can be normalized for bit-plane encoding.

The bit-plane encoding uses a significance pass and a refinement pass. The refinement pass keeps a list of the coefficients that have been found significant and sends one additional refinement bit for each coefficient in this list; this is exactly the same as in previous zerotree coders. The significance pass for finding significant coefficients organizes the coefficients into classes, and uses adaptive group testing to code each class. For each class C , we let k_C represent the group iteration size used for the coefficients in that class; the adaptive group testing method will update k_C as coefficients in that class are coded.

Note that the significance pass and its associated adaptive group testing method are the parts of GTW that are significantly different from previous image coders; we now proceed to describe the significance pass, group iterations, and the adaptive group testing strategy in more detail.

Significance Pass

The GTW algorithm to code the significant coefficients in a bit-plane is fairly simple. The basic idea is to repeatedly code coefficients in the GTW classes with group iterations until all have been coded. We will code the coefficients in class C with group iteration size k_C , where k_C is a dynamically changing value determined by the adaptive group testing strategy. Initially, k_C starts out at 1 for all classes.

We try to code the coefficients in a class only when the GTW class is *adequate*. A class

C is adequate if there are at least k_C coefficients in that class, where k_C is the current group iteration size for class C . Coding only adequate classes ensures that the appropriate group iteration size is used for the class; this is necessary for good coding performance.

Our algorithm starts off coding coefficients only from adequate classes. When there are no more adequate classes, most coefficients will have been coded. However, there will be a few coefficients left in inadequate classes. At this point, we start combining coefficients together from inadequate classes and code the combined groups. In this context, we code only combined groups of coefficients that are *end-adequate*. We say a group is end-adequate when it contains k coefficients, where k is the minimum k_C over all classes C such that a coefficient of the group is a member of class C . After coding all end-adequate groups, there may be some remaining coefficients which when combined together, do not form an end-adequate group. At this point, these remaining coefficients are tested in one combined group. This process is repeated until all coefficients are coded.

Group Iteration

As mentioned in section 4.2.1, a group iteration of size k is the process of performing a group test on k items, and then, if necessary, performing binary splitting to identify a significant item in the group. Due to the entropy argument found in section 5.1.3, our group iteration procedure will use optimal binary splitting (procedure 5.1). In this procedure, a set of size 11 is partitioned into sets of size 4 and 7 with the set of size 4 tested first, and a set of size 13 is partitioned into sets of size 5 and 8 with the set of size 5 tested first.

Adaptive Group Testing Strategy

The main difference between our group testing strategy and Hwang's generalized binary splitting algorithm is in our adaptive method of choosing the group iteration size k . Unlike his algorithm, we do not restrict k to be a power of 2.

For each class C , our method keeps track of s_C , the number of significant coefficients

Procedure 6.1 GTW Significance Pass

Initially, we are given a set of uncoded coefficients, each belonging to a particular class.

While there are coefficients left to be coded,

- If there is an adequate GTW class,
 1. Let C be the first adequate GTW class, according to the GTW ordering of the classes.
 2. Pick the first k_C coefficients from class C , and perform a group iteration on this group. The group iteration will output bits corresponding to the group test results.
 3. Let s and i represent the number of significant and insignificant coefficients found in the group iteration, respectively.
 4. If a significant coefficient is found ($s = 1$), output its sign and update the neighbors of that coefficient (coefficients could change classes at this point).
 5. Using the adaptive group testing strategy, update k_C knowing that s items were found significant and i items were found insignificant in class C .
 - Else if there is no adequate class,
 1. Let $G \leftarrow \emptyset$.
 2. While G is not end-adequate and there are still uncoded coefficients not in G ,
 - Let x be the first coefficient in the first non-empty class, according to the GTW ordering.
 - $G \leftarrow G \cup \{x\}$.
 3. Perform a group iteration on G . If a significant coefficient is found, output its sign and update the neighbors of that coefficient.
-

Procedure 6.2 Group Iteration

Initially, we are given a group K of size k .

Perform a group test on K , outputting the result.

- If K is insignificant, then mark the coefficients in K as insignificant, and return k as the number of insignificant coefficients found.
- Else if K is significant,

Perform the optimal binary splitting on K (procedure 5.1). This procedure will return x , the significant coefficient found, as well as I , a set of coefficients identified as insignificant.

Output the results of the group tests performed in the optimal binary splitting procedure.

Mark the coefficients in K as significant, insignificant, or not yet identified, according to the results x and I .

Return x and $|I|$.

Procedure 6.3 Adaptive Group Testing

At the beginning of each significance pass, for each class C , set $n_C \leftarrow 0$, $s_C \leftarrow 0$, and $k_C \leftarrow 1$.

After s significant items and i insignificant items have been found in class C , we perform the following:

- Set $s_C \leftarrow s_C + s$, and $n_C \leftarrow n_C + s + i$.
 - If $s_C = 0$ then $k_C \leftarrow k_C \cdot 2$.
 - Else if $s_C > 0$, then $k_C \leftarrow k^*(s_C/n_C)$.
-

identified so far in class C , as well as n_C , the total number of coefficients identified so far in C . Our adaptive strategy will choose k_C , the group iteration size that will be used for class C . Our method consists of two phases, an initial doubling phase, and a steady-state estimation phase. We start in the doubling phase with $k_C = 1$. In this phase, while no significant coefficient is found in the previous group iterations for C , we double k_C . Once the first significant coefficient is found, we enter the estimation phase. In this phase, we use $p = (n_C - s_C)/n_C$ as the probability estimate of insignificance, and choose $k_C = k^*(p)$ (i.e. choose the k to satisfy the inequalities 3.6 in section 3.3.4).

Note that the initial phase of our strategy is the same as both the doubling strategy taken from Bar-Noy *et al.* [4] and Langdon's [29] adaptive Golomb coding method. This scheme quickly adapts to the characteristics of the source.

6.2 Results

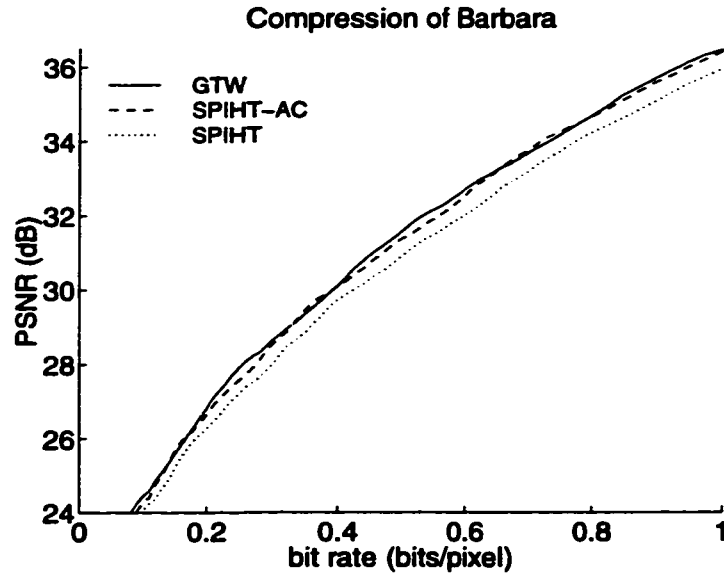


Figure 6.6: Comparing performance of GTW and SPIHT.

We present our results on the 512×512 Barbara image and compare our PSNR curve

with the ones for SPIHT with and without arithmetic coding. Figure 6.6 shows that GTW performs about 0.7 dB better than SPIHT and about the same as SPIHT-AC.

Table 6.1 compares GTW against SPIHT with arithmetic coding for a wide variety of natural images. All images are 512×512 in size, except for the man image which is 1024×1024 . All are 8-bit gray-scale images. The standard test images Goldhill, Lena, and Barbara can be obtained from UCLA's web site [28]. The other images are part of the USC image database [50]. All results are measured in terms of PSNR, in dB. The Δ SPIHT-AC rows represent the difference in PSNR between SPIHT-AC and GTW. Positive numbers indicate the amount of PSNR improvement of SPIHT-AC over GTW. We can see that the difference between SPIHT-AC and GTW is very small.

6.3 Alternative Approaches

6.3.1 Alternative Significant Neighbor Metric

It is clear that our significant neighbor metric is *ad hoc* in nature, since all neighbors of a coefficient c (except its children) have an equal say towards whether c is likely to be significant. By performing statistical analysis across an image, it should be possible to calculate a set of weights w_1, w_2, \dots, w_{15} for the 15 neighbors of c that represent how much each neighbor contributes to c 's significance status. By taking the statistics into account, we are able to better predict whether c will become significant, resulting in better compression.

As a more "principled" approach, we redefined the significant neighbor metric to be $\sum_{i=1}^{15} w_i b_i$, where w_i is the weight given to the i th neighbor of a coefficient, and b_i is the bit value of the i th neighbor (1 for significant, and 0 for insignificant or unknown). Although the numeric weights could be calculated by many different methods, we used the logit [31] approach, which is a standard method in statistics for assigning weights when the training data are discrete. To compress an image, we calculated the weights using that image and then sent the weights in the initial part of the compressed bitstream.

Figure 6.7 is a graph that shows the percentage of significant coefficients found in the

Table 6.1: Comparing PSNR results of GTW and SPIHT-AC. $\Delta\text{SPIHT-AC} = \text{SPIHT-AC} - \text{GTW}$.

Image	Algorithm	Rate (Bits/pixel)			
		0.1	0.25	0.5	1.0
rough wall	GTW	24.37	27.22	29.51	32.51
	$\Delta\text{SPIHT-AC}$	+ .31	+ .10	0.00	- .07
couple	GTW	26.12	29.13	32.41	36.45
	$\Delta\text{SPIHT-AC}$	+ .05	+ .08	+ .04	+ .13
man	GTW	27.80	31.31	34.13	37.42
	$\Delta\text{SPIHT-AC}$	+ .32	+ .05	+ .12	- .15
boat	GTW	27.31	30.93	34.27	39.01
	$\Delta\text{SPIHT-AC}$	+ .05	+ .04	+ .18	+ .11
tank	GTW	27.50	29.35	31.17	33.86
	$\Delta\text{SPIHT-AC}$	- .05	+ .01	+ .01	- .08
Goldhill	GTW	27.86	30.49	33.09	36.42
	$\Delta\text{SPIHT-AC}$	+ .08	+ .07	+ .04	+ .13
Lena	GTW	30.17	34.17	37.28	40.45
	$\Delta\text{SPIHT-AC}$	+ .05	- .06	- .07	- .04
Barbara	GTW	24.41	27.86	31.49	36.42
	$\Delta\text{SPIHT-AC}$	- .15	- .28	- .09	- .01

set of coefficients that had a given value as its significant neighbor metric, on the Barbara image. Comparing with figure 6.8, it appears that the logit metric and the GTW significant neighbor count give about the same information. The logit metric is slightly better because at low values of the metric, the percentage of significant coefficients is higher. However, we feel the added complexity of training to obtain the weights was not worth the slight benefits. Overall, this approach performed slightly better than our significant neighbor metric. The improvements over the *ad hoc* method ranged from 0.0 dB to 0.15 dB at various bit rates on the Barbara image.

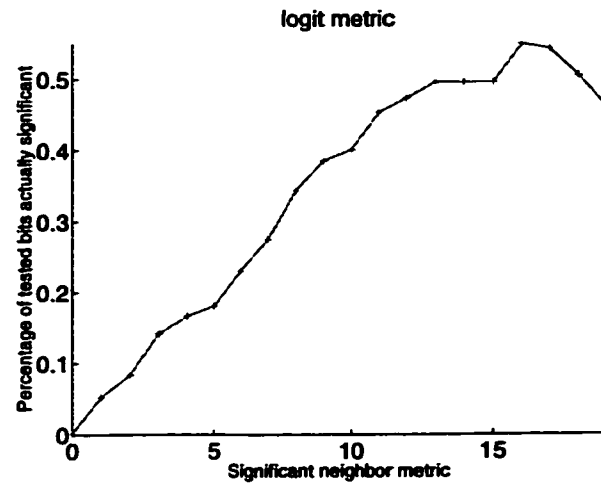


Figure 6.7: Significance results of the logit metric value, uniformly quantized into 20 intervals, on the Barbara image. The logit weights were calculated from training on Barbara.

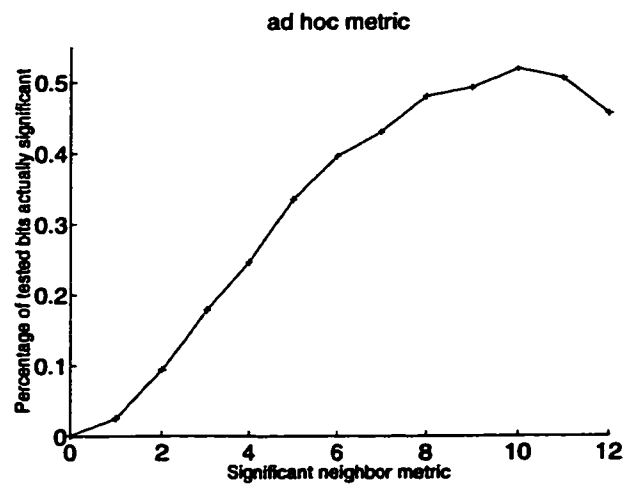


Figure 6.8: Significance results of the *ad hoc* significant neighbor count, on the Barbara image.

6.3.2 Alternative Pattern Types

We observed that the coefficients that just became significant in the current bit-plane had a substantial effect on the value of the significant neighbor metric for their neighbors. Thus, it would be nice to maximize the amount of information known about a coefficient's neighbors. Unfortunately, this may not be possible in an overall sense: We must pick a specific ordering in which to code the coefficients, and on average, when coding a specific coefficient, we expect roughly half of its currently insignificant neighbors to not yet be coded at the current bit-plane.

Although maximizing the information known might not be possible, we can change the rate at which the information about neighboring coefficients increases. Recall that the bootstrapping effect describes how information increases as more coefficients are coded. This effect is different for different patterns, as illustrated in figure 6.9. We tried the two alternative patterns in figure 6.9 and found no appreciable difference in coding efficiency between these different patterns and the original pattern. The difference between the rate-distortion curves for the Barbara image using different pattern types was less than 0.01 dB on average.

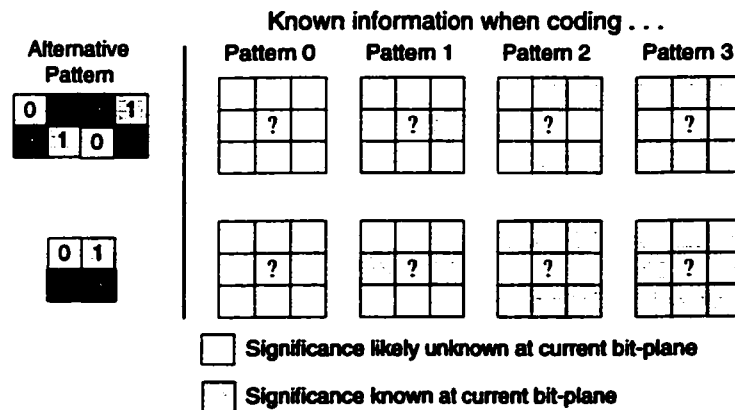


Figure 6.9: How information propagates in different patterns.

We also tried eliminating the pattern type completely; this reduces the number of classes

by a factor of four. This change also performed roughly the same as when using the pattern type. In this case, the difference between the two curves on the Barbara image averaged about 0.03 dB. Having no patterns was a bit better at low bit-rates, while having patterns was a bit better at higher bit-rates.

6.3.3 Varying Ordering of Testing Groups

Varying GTW Class Ordering

The class ordering in GTW, described in section 6.1.3, considers the 3 class characteristics in the following order: significant neighbor metric first, then pattern type, then subband level. We tried our algorithm with each of the 6 possible total orderings of the 3 different characteristics. From experimentation we found that these alternative orderings did not perform as well; figure 6.10 shows the performance of these orderings on the Barbara image. Intuitively, it would appear that considering the subband level first would give good results because lower frequency subbands tend to have more significant coefficients in the early bit planes. However, ordering with subband level first produced concave dips in the rate-distortion curve. This could mean that it is more worthwhile to identify significant coefficients in higher frequency subbands early. Furthermore, the orderings with pattern type first were significantly worse, because the pattern type is not useful for predicting significance of coefficients.

Group Iteration Size Ordering

We found that significant neighbor metric is better at predicting significance than subband level. In terms of rate-distortion theory, we would like to find the significant coefficients as soon as possible to reduce the distortion. Thus, it is better to test all the coefficients in order of most likely to be significant, to least likely to be significant.

One way to apply this principle is to use the adaptive group testing statistics to set the ordering of which classes should be tested first. Classes with a smaller group size are more

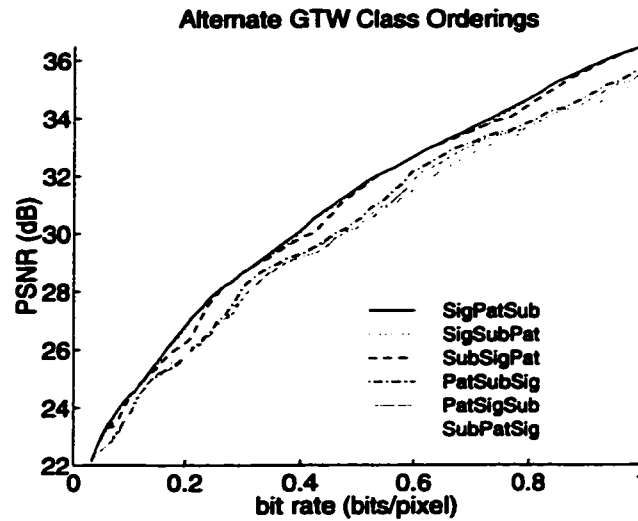


Figure 6.10: Different class orderings on the Barbara image. Sig, Sub, and Pat stand for Significant Neighbor Metric, Subband Level, and Pattern Type, respectively. Each line is labeled by the order of importance of the characteristics, so that SigPatSub represents the normal GTW ordering.

likely to have significant coefficients and thus should be tested first. We tried this ordering where classes were tested in the order of smallest group iteration size first. Ties were broken by the regular GTW ordering. On the Barbara image, the rate-distortion performance did not change with this technique.

Generalized Coefficient Ordering

We could also try different ways of ordering the coefficients within a particular class. Instead of arbitrarily group testing the first k coefficients that are in a GTW class, we could instead look for k most important coefficients in that class, and code them first. The criterion of importance could be based on the likelihood that a coefficient is significant, or on the amount of information a coefficient could give to its neighbors. This is the idea behind the two new orderings described in this section: the *parent-first ordering* and the *neighbors-first ordering*. Figure 6.11 shows how these orderings compare to the original

GTW ordering on Barbara.

In the parent-first ordering, only coefficients whose parents are already coded are allowed to be tested in a group iteration. In this case, a class is considered inadequate if it does not have enough coefficients with coded parents to fulfill the group iteration size quota. The normal GTW ordering was used to decide which classes to code first. Note that with this ordering, it is still possible for coefficients in high-frequency subbands to be coded before coefficients in low-frequency subbands. This can happen when the coefficients in the high-frequency subband are not descendants of the coefficients in the low-frequency subband. This ordering performed slightly worse than the normal GTW ordering.

In the neighbor-first ordering, we allowed a coefficient of pattern type i to be coded only when its neighbors that have pattern type $< i$ are already coded. In this case, a class is considered inadequate if it does not have enough coefficients with an appropriate number of coded neighbors to fulfill the group iteration size quota. We can view this an attempt to maximize information flow, based on the following scenario: Suppose that c_a and c_b are both uncoded coefficients in the current bit-plane, and that all of c_a 's neighbors are coded while none of c_b 's neighbors are coded yet. Then it may be more beneficial to code c_b before c_a because c_b will provide information about itself to its neighbors, where as c_a will not. The neighbor first method performed significantly worse than the normal GTW ordering.

The results of these two orderings imply that trying to maximize the information flow between coefficients does not improve the net performance. Although better prediction can be obtained for coefficients when we maximize the information available, it appears that the slightly better prediction is not enough to offset the loss incurred from not coding the most likely to be significant coefficients first.

6.3.4 GTW with Arithmetic Coding

There is a significant gain in going from SPIHT to SPIHT-AC for Barbara. Such a gain does not seem possible in adding arithmetic coding to GTW. The SPIHT zerotrees take into account the correlation between a coefficient and its parent and children, but not its

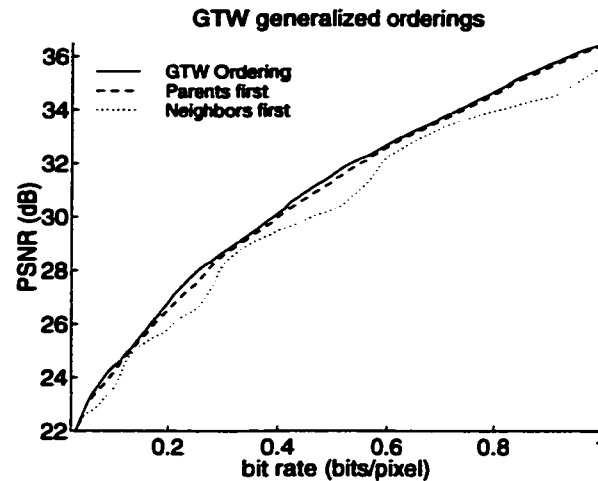


Figure 6.11: Generalized coefficient orderings on the Barbara image.

spatially adjacent coefficients in the same subband. Arithmetic coding in SPIHT-AC helps account for correlation between spatially adjacent coefficients by using different contexts for adjacent neighbors of different values. In GTW, spatially adjacent neighbor information is used as part of the context for group testing. Out of curiosity we added an adaptive first-order arithmetic coder on a binary symbol set to GTW. There was no appreciable difference in rate-distortion performance.

6.3.5 Varying Group Testing Strategies

Another way to optimize GTW is to change the exact group testing strategy. Instead of using adaptive group testing, it is possible to initially send statistics saying what the group iteration size should be for each class. This method performed worse than our adaptive strategy.

6.4 Conclusions

6.4.1 Comparison with Wavelet Image Coders

To put our work in context, we first highlight the key differences between GTW and other recent work wavelet image compression techniques.

Ordentlich, Weinberger, and Seroussi's algorithm (OWS) [40] and Malvar's algorithm (PWC) [34] are similar to GTW in that they both use Golomb codes. The main differences are in what contexts are used and in what order the bits are coded. In OWS the coefficients are coded in a zig-zag order with fractional bit-planes, where as in PWC the coefficients are reordered in a complicated but fixed way. GTW orders its coefficients dynamically, optimizing for information flow and for likelihood of being significant.

Wu's ECECOW algorithm [60] does not use group testing, but strict arithmetic encoding with context. Training is used to calculate the contexts that will be used in the coding. ECECOW has impressive rate-distortion performance, certainly better than SPIHT-AC and GTW, but neither SPIHT-AC nor GTW rely on training to achieve their results.

Chai *et al.*'s SLCCA [7] coder identifies significant coefficients in clusters. It initially codes the starting positions of a cluster of significant coefficients, and expands a cluster by coding the coefficients around the starting position. When a significant coefficient's child is the start of a new cluster, then a significance link symbol is sent, so that the position of the new cluster need not be sent. This approach of looking for clusters of significant coefficients is similar to the way that GTW updates the neighbors of a significant coefficient, and codes the coefficients with the highest significant neighbor metric. Including the parent in the significant neighbor metric helps "link" parent clusters to their children clusters.

As described in section 2.6.6, Taubman's EBCOT coder [52] uses a combination of zero-tree coding, sign coding, and fractional bit-planes to encode a bit-plane. Each of these techniques is followed by arithmetic coding with carefully chosen contexts. They achieve slightly better performance than GTW mainly due to the sign coding, and through keeping track of the manner in which the orientation of a subband affects the correlation between

neighbors. Note that the fractional bit-plane method is similar to the way that GTW chooses the ordering of its GTW classes to find significant coefficients quickly. Since GTW propagates the information about significant neighbors as soon as they are discovered, we can view GTW as making many fractional bit-plane passes. This is in contrast to EBCOT which has only four passes in its fractional bit-plane method.

6.4.2 *Summary of Results*

We have shown that the proposed method of coding GTW classes has better rate-distortion performance than the non-arithmetic-coded version of SPIHT. This shows that the group testing method can offer a significant improvement over the zerotree coding method. We also have shown that it is possible to get performance comparable to the arithmetic-coded version of SPIHT without using arithmetic coding.

Unfortunately, our work in exploring alternative approaches in using group testing for image compression did not result in a coder with better PSNR performance than GTW. However, our work is helpful in understanding why certain image compression techniques work better than others. Thus, our work on alternatives is useful for helping future researchers understand image compression better, and may help others invent even better techniques in the future.

Furthermore, our work unifies many previously proposed wavelet coding methods into a single framework based on group testing; these previous methods include zerotree coding and Golomb coding. This leads to a better fundamental understanding of previous image coders, and thus is a significant contribution to the literature.

Chapter 7

GROUP TESTING FOR WAVELET PACKETS

7.1 Introduction

In this chapter, we introduce our new Group Testing for Wavelet Packets algorithm (GTWP) [21], which is an extension of GTW to the wavelet packet transform. The wavelet packet transform is a generalization of the standard dyadic wavelet decomposition of an image that adapts to the particular image being coded to obtain better coding performance. Our main goal in this chapter is to show that using group testing on wavelet packets results in better PSNR performance than previous techniques that applied zerotree coding to wavelet packets.

In [64], Xiong *et al.* presented the space-frequency quantization for wavelet packets (SFQ-WP) image coder. It was an extension of the space-frequency quantization algorithm (SFQ) [63], which was originally defined for the dyadic wavelet decomposition. Based on this work, it is known that using the wavelet packet decomposition on an image can improve the PSNR by 0.8-1.3 dB for some images, when compared with the standard wavelet transform. Unfortunately, SFQ and SFQ-WP are not embedded coders.

Other researchers [38, 8, 39] have tried many different techniques to encode wavelet packet transform coefficients of an image. In particular, Rajpoot *et al.* [42] and Khalil *et al.* [27] applied zerotree techniques to wavelet packet transforms to generate an embedded bitstream. Unfortunately, these techniques define the parent-child relationships in an *ad hoc* manner. This motivated us to find more natural methods to code wavelet packet transform coefficients.

Like our previous algorithm, GTWP algorithm basically divides the coefficients into

different classes, and codes each class with a different group tester. The chief difference is in the transform used, and in the definition of the classes. We demonstrate the flexibility of the group testing framework by implementing two different class definitions. One definition is a simplification of the GTW classes, as defined in section 6.1.2. The other is an adaptation of the contexts in EBCOT [52], which is the basis for JPEG 2000.

Our results indicate that GTWP can be up to 1.3 dB better than GTW on images where an adaptive wavelet packet decomposition is helpful. We also show that GTWP is better than previous techniques that combine wavelet packets and zerotree coding. This is expected because our method is not restricted to coding trees and is thus inherently more flexible than standard zerotree coding. Furthermore, we show that GTWP is often better than JPEG 2000 and never significantly worse.

This chapter is organized as follows: Section 7.2 introduces the wavelet packet transform, with a focus on image compression. Section 7.3 introduces two variants of GTWP. Section 7.4 evaluates our algorithm's rate-distortion performance. Section 7.5 summarizes our results.

7.2 Wavelet Packets Background

As described in [58], wavelet packets are a generalization of the standard dyadic wavelet decomposition of a signal. The standard dyadic wavelet transform (as discussed in section 2.4.2) decomposes the signal by applying a series of successive filters to the lowest frequency subband. Wavelet packets are a generalization of this where the successive filters can be applied to any subband of any orientation, not just the lowest frequency LL subband. Any one particular choice of subbands to decompose is known as a basis; the choice of exactly which basis to use depends on the characteristics of the input. Figure 7.1 shows the subbands after transforming an image with wavelet packet transform using one particular basis.

A basis that adapts well to the input signal can be chosen via Coifman and Wicker-

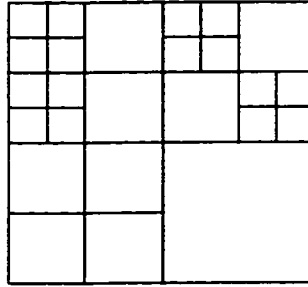


Figure 7.1: Sample subbands of a wavelet-packet transformed image.

hauser's entropy-based technique [9] or by Ramchandran and Vetterli's rate-distortion optimization technique [43]. These methods work by fully decomposing all subbands to a predefined maximum depth, thus forming a decomposition tree where each decomposed subband is represented in the tree by a parent node with four child nodes. Then the best basis is found by pruning this decomposition tree in a recursive bottom-up fashion. The entropy-based technique prunes the tree to minimize the overall estimated entropy of the wavelet packet structure. The rate-distortion method is given a particular target bit rate for the image and prunes the tree to minimize the distortion of the image.

Xiong *et al.* [62] first explored the combination of a wavelet packet decomposition of an image with the SFQ coder, a coder that uses zerotree quantization techniques. The difficulty in applying a zerotree quantization to wavelet packets is that it is no longer clear how to define the parent-child relationships in the trees. As noted by Rajpoot *et al.* [42], there is a *parenting conflict*, where some child coefficients could have multiple parents. This problem has typically been solved by limiting the space of possible wavelet packet decompositions so that no parenting conflict occurs, or by assigning the parent-child relationships in a somewhat *ad hoc* manner (see [62, 42, 27]).

7.3 Group Testing for Wavelet Packets

We propose a new coder, Group Testing for Wavelet Packets (GTWP), that extends our previous GTW algorithm to use wavelet packets for encoding images. In this new algorithm, we find the best basis for the input image, and encode the structure of this basis in the first bits of our compressed image. Then we define new GTWP classes based on the characteristics of the wavelet packet decomposition of the image, so that the classes are encoded efficiently. We also specify the order in which we plan to code the classes.

Once both the GTWP classes and the ordering between them are defined, then we can code each class with a different group tester, and proceed in the same manner as GTW. That is, we still proceed by bit-plane encoding with a significance pass and a refinement pass. The significant pass is exactly the same as procedure 6.1 defined in section 6.1.4 except that GTWP classes are used instead of GTW classes. As in GTW, we also compute average value of the lowest frequency subband and subtract it from the subband before starting the bit-plane encoding.

We first describe how we choose the best basis and encode it; then we describe two different methods for defining the GTWP classes with their associated ordering.

7.3.1 Best Basis

We investigated using both the entropy-based technique and the rate-distortion technique for computing the best wavelet packet basis. For the entropy-based technique, we explored many different metrics for calculating the entropy of a particular subband. Let v_i represent the value of the coefficients of a subband. Then the entropy metrics we tried are as follows:

- log energy metric: $\sum_i \ln(v_i^2)$.
- Shannon metric (used in [9]): $\sum_i v_i^2 \ln(v_i^2)$.
- L1-norm metric: $\sum_i |v_i|$.
- threshold metric: Given a threshold value T , calculate $|\{v_i : |v_i| > T\}|$.

- first-order entropy metric (used in [27]): Given a quantization step size Q , divide the coefficients into quantization bins, and estimate the probability p_i of a bin occurring by $p_i = n/t$, where n is the number of coefficients in that bin, and t is the total number of coefficients. Calculate $-\sum p_i \log_2(p_i)$.

We also tried the rate-distortion optimization technique, optimizing for a wide variety of bit-rates for various different possible scalar quantizers. Note that this technique is not well suited to our problem because it forces us to pick artificial parameters, namely, the final bit-rate for which to optimize and the quantizer step sizes to consider. Since GTWP is an embedded coder, the final bit-rate we choose for the purpose of obtaining the best basis does not correspond to the actual final bit-rate to which we encode the image. Furthermore, since GTWP codes the transform coefficients bit-plane by bit-plane, it cannot choose to code a subband with a particular quantizer step size; the step size it ends up using may not have any relation to the quantizer step size parameters that we chose to run the rate-distortion optimization technique.

It is interesting to note that the optimal calculated quantizer step size for all the subbands under the rate-distortion technique differed from each other by no more than a factor of 2, at least for the Barbara image. In the bit-plane encoding technique, if we stop coding in the middle of a bit-plane, then the coefficients that have not yet been coded in the current bit-plane are quantized with a step size of 2 times the step size of those coefficients that have been coded. This suggests that GTWP's bit-plane encoding technique may be a good approximation to the quantization step sizes that the rate-distortion optimization best basis produces.

The log energy, Shannon, and L1-norm metrics are the simplest in that they do not require additional parameters (such as threshold value or quantization step size) to compute. The top performers for our algorithm are the log energy metric and the rate-distortion optimization metric. Seeing that the log-energy metric was simpler and did not require selecting artificial parameters, we used it exclusively. As an example, we show the best

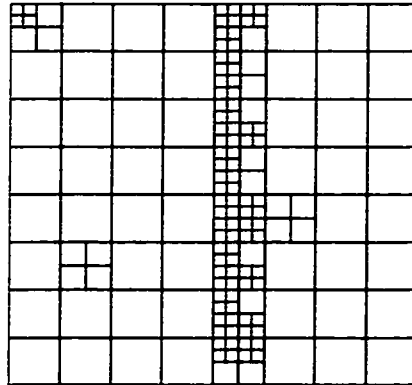


Figure 7.2: Illustration of the best basis for the Barbara image.

basis chosen by the log energy metric on the Barbara image in figure 7.2. For simplicity, we show only 5 levels of decomposition even though our algorithm uses a maximum of 6 levels.

To encode the decomposition tree, we simply perform a depth-first traversal of the tree, and encode a 1 when that particular node is split into children, and a 0 when the node is a leaf.

7.3.2 GTWP Classes

To illustrate the flexibility of the group testing methodology, we implemented two different ways of choosing the GTWP classes: GTWP-S and GTWP-J. The first method is a simplification of the definition GTW classes (S for Simple), and the second method is based on the contexts used in the JPEG-2000 image coder (J for JPEG-2000).

GTWP-S

The GTWP-S classes are a simplification of GTW classes and are defined by three different characteristics: the subband level, the significant neighbor metric, and the pattern type.

Subband Level. The lowest frequency subband representing the average of the entire image counts as subband level one. There is one additional subband level for each level of the wavelet transform. Figure 7.3 shows the 4 subband levels when 3 levels of the wavelet transform are performed. Note that because we are using wavelet packets, each subband level may contain more than 3 actual subbands.

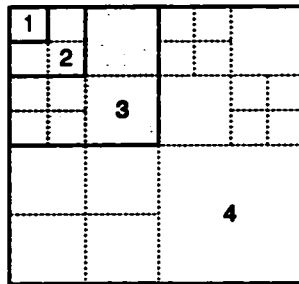


Figure 7.3: Subband levels of a wavelet-packet transformed image in GTWP-S. Solid lines separate subband levels; dotted lines separate subbands. All shaded coefficients are in subband level 3. Subband levels 2, 3, and 4 have 3, 6, and 15 subbands, respectively.

Significant Neighbor Metric. To finesse the problem of defining parent-child relationships in the wavelet packet transform, we restrict the neighbors of a coefficient to be one of up to 8 spatially adjacent coefficients in the same subband. Like GTW, there are 4 values in the significant neighbor metric, 0, 1, 2, and 3+, depending on whether 0, 1, 2, or more than 2 neighbors are significant. Because there are only up to 8 neighbors, the maximum neighbor count is 8.

Pattern Type. This characteristic is exactly the same as in GTW, where the coefficients in a subband are assigned into 4 different pattern types based solely on its position in a subband.

Overall, there are 7 subband levels, 4 significant neighbor types, and 4 pattern types resulting in 112 classes total. These classes are ordered according to the same ordering as

the GTW classes, namely, with significant neighbor metric first, pattern type second, and subband level last. Putting the pattern type or subband level first gives inferior results.

GTWP-J

The GTWP-J classes are based on the contexts in Taubman’s EBCOT coder [52], and are also found in the JPEG 2000 coder. In this class definition, there are only two characteristics that define the classes: the orientation type and the neighborhood significance label.

Orientation Type. The orientation type of subband s contained in subband level i is based on the orientation of the largest parent subband in subband level i that contains s . Here, the subband levels are defined as specified in the GTWP-S classes. There are only 3 orientation types, LH (vertically high-pass), HL (horizontally high-pass), and HH subbands. The subband at subband level 1 (the LL subband) is considered to have orientation type LH. The orientation type of a coefficient in subband s is the orientation type of subband s . This is illustrated in figure 7.4.

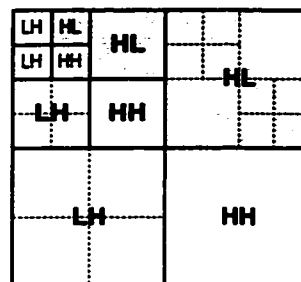


Figure 7.4: Illustration of orientation type of a wavelet-packet transformed image. All shaded coefficients have orientation type HL.

Neighborhood Significance Label. Let h , v , and d represent the number of significant neighbors that a coefficient has which are adjacent to it horizontally, vertically, and diagonally, respectively. Thus, h and v both have a value of up to 2, whereas d has a maximum

value of 4. The neighborhood significance label is assigned according to table 7.1. Note that the labeling is dependent on the orientation type of the coefficient. This label is taken from the context classifier in the EBCOT coder.

Table 7.1: Neighborhood significance label.

Assigned Label	LH subband			HL subband			HH subband	
	h	v	d	h	v	d	d	$h + v$
0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	1
2	0	0	≥ 2	0	0	≥ 2	0	≥ 2
3	0	1	≥ 0	1	0	≥ 0	1	0
4	0	2	≥ 0	2	0	≥ 0	1	1
5	1	0	0	0	1	0	1	≥ 1
6	1	0	≥ 1	0	1	≥ 1	2	0
7	1	≥ 1	≥ 0	≥ 1	1	≥ 0	2	≥ 0
8	2	≥ 0	≥ 0	≥ 0	2	≥ 0	≥ 2	≥ 0

With 3 orientation types and 9 significant neighbor labels for each orientation type, there are a total of 27 classes. The classes are ordered according to the group iteration size. Classes with smaller group iteration size are coded first, since they are more likely to be significant. Ties are broken arbitrarily.

7.4 Results

Here we present our results on some standard 8-bit monochrome images: the 512×512 images Barbara, Goldhill, and Lena (available from [28]); and a 768×768 fingerprint image from the FBI's fingerprint compression standard [5]. We present results for several different algorithms, including GTW, GTWP-S, GTWP-J, JPEG 2000 and SFQ-WP. All algorithms use the Daubechies 9/7-tap filters [3]. JPEG 2000 results were produced with a beta version

of a codec [2] for the JPEG 2000 image compression standard. SFQ-WP represents the practical version of Xiong *et al.*'s SFQ algorithm applied with wavelet packets; results are taken from [64]. To our knowledge, SFQ-WP is the current state-of-the-art method for image compression with wavelet packets.

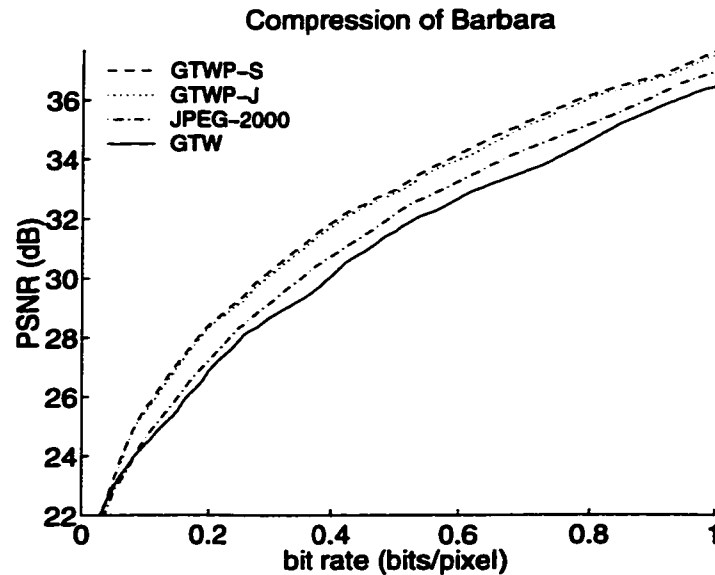


Figure 7.5: Comparing performance of GTW, GTWP, and JPEG-2000.

Figure 7.5 compares the PSNR curves for GTW, GTWP-S, GTWP-J, and JPEG 2000 on the Barbara image. As can be seen, there is little difference between the two GTWP variants, and using the wavelet packets increases the PSNR by about 1.2 dB over GTW. Table 7.2 lists PSNR results for all four images on the different algorithms.

The table shows how the amount of improvement for using wavelet packets instead of the dyadic wavelet decomposition is highly dependent on the type of image. Some images (like Barbara) benefit significantly from the wavelet packet decomposition; some images (like Goldhill) benefit slightly; and some (like Lena) do not benefit at all. In fact, the best basis for the Lena image was calculated to be the standard dyadic wavelet decomposition with one additional decomposition of a highest frequency subband. Thus, we would expect the results for GTWP on Lena to be roughly the same as that for GTW; this is what the

Table 7.2: Comparing PSNR results of various algorithms. GTW is used as the baseline, so that for algorithm A, $\Delta A = A - \text{GTW}$.

<i>Image</i>	<i>Algorithm</i>	<i>Rate (Bits/pixel)</i>			
		0.1	0.25	0.5	1.0
Barbara	GTW	24.37	27.87	31.59	36.47
	$\Delta \text{JPEG2000}$	+19	+46	+50	+46
	$\Delta \text{GTWP-S}$	+1.27	+1.37	1.39	+1.18
	$\Delta \text{GTWP-J}$	+1.15	+1.27	+1.28	+1.07
	$\Delta \text{SFQ-WP}$?	+1.38	+1.53	+1.22
Lena	GTW	30.06	34.13	37.27	40.51
	$\Delta \text{JPEG2000}$	-.27	-.12	-.13	-.46
	$\Delta \text{GTWP-S}$	-.04	+.06	0.00	-.09
	$\Delta \text{GTWP-J}$	-.02	+.09	-.01	-.09
	$\Delta \text{SFQ-WP}$?	+.22	+1.13	+1.04
Goldhill	GTW	27.76	30.46	33.10	36.47
	$\Delta \text{JPEG2000}$	-.05	+.04	+.05	-.07
	$\Delta \text{GTWP-S}$	0.00	+1.12	+1.17	+1.17
	$\Delta \text{GTWP-J}$	+.01	+.09	+1.17	+1.13
fingerprint	GTW	28.35	32.80	36.06	39.98
	$\Delta \text{JPEG2000}$	+.24	+1.17	+1.10	+1.16
	$\Delta \text{GTWP-S}$	+.50	+.35	+.73	+1.38
	$\Delta \text{GTWP-J}$	+.45	+.31	+.70	+1.38

results show. The slight performance differences are due mostly to differing significant neighbor metrics.

If we compare our results with the published results of previous zerotree coding techniques on wavelet packets, we see that we outperform Rajpoot *et al.*'s technique by over 1.0 dB on the fingerprint image, and we outperform Khalil *et al.*'s technique by about 0.3 dB on the Barbara image.

It is interesting to note that there was little difference between GTWP-S and GTWP-J. It appears that as long as something reasonable is chosen, the exact method of classifying coefficients whose neighbors are significant does not matter that much. In fact, we also tested GTW-S, a version of GTW simplified so that the significant neighbor metric did not include spatially identical neighbors in different subbands (making GTW-S similar to GTWP-S). There was also little difference between GTW-S and GTW. It appears that the significance of a coefficient depends almost entirely on its 8 immediately adjacent neighbors, and very little on the parents and other neighbors in different subbands. This agrees with the findings in [52].

As can be seen in the table, GTWP's performance is often better than JPEG 2000, and never significantly worse. Furthermore, GTWP's performance is not too far from that of SFQ-WP. Although GTWP is worse, GTWP is an embedded coder, while SFQ-WP is not. Furthermore, GTWP is much simpler than SFQ in that it does not use arithmetic coding, and does not perform rate-distortion optimization.

7.5 Conclusion

In conclusion, we have shown that group testing is a general method that can be easily adapted to encode wavelet packet transform coefficients. We have shown that GTWP has good PSNR performance, in that it outperforms previously published zerotree coders that used wavelet packets. Since our group testing coder outperforms all embedded image coders that we know of which apply zerotree coding techniques to wavelet packets, we

conclude that the group testing technique is superior to the zerotree coding technique.

We have also shown that GTWP can outperform GTW by 0.8 dB to 1.3 dB on some images. Furthermore, GTWP sometimes outperforms JPEG 2000, a benchmark algorithm that does not use wavelet packets. These facts confirm that using wavelet packets for image compression instead of the standard dyadic wavelet decomposition can result in significant PSNR improvements on certain types of images.

Chapter 8

GROUP TESTING FOR BLOCK TRANSFORMS

In this chapter, we introduce our group testing for block transforms algorithm (GTBT), which applies GTW's group testing technique to code coefficients generated from various different *block transforms*. A block transform is simply any transform typically applied by dividing the input data into blocks, and transforming the blocks independently. The particular block transforms we use will be those that have previously been used for image compression; this includes the popular discrete cosine transform [46], as well as many lapped transforms introduced by Malvar [36] and Tran *et al.* [53].

Although zerotree coding is typically used to code wavelet-transformed coefficients, it has also been applied to block transforms with some success. Recall that the zerotree technique was motivated by the multi-resolution structure of the dyadic wavelet decomposition, where coefficients could be organized into trees formed across different subbands. Since there is a mismatch between the zerotree structure and the statistical characteristics of block transforms, using zerotree coding on block transform coefficients leads to inefficiencies in coding performance. As a generalization of zerotree coding, group testing is not hampered by the zerotree structure and can easily be adapted to more efficiently code block-transform coefficients. In this chapter we show the flexibility and efficacy of the group testing framework by applying it to block transforms.

Our technique will be very similar to GTW in that we divide the coefficients in a bit-plane into different classes, and code each class with a different group tester. For our algorithm to compress efficiently, the class definitions must reflect the statistical characteristics of the block-transformed coefficients. The main goal of this chapter is to discover the best way to define the classes for the block transform, in order to achieve efficient compression.

Our results indicate that group testing performs significantly better than previous zerotree coding algorithms on some images. In particular, we perform about 1.6 dB better than both GTW [20] and SPIHT [45] at a wide range of bit-rates on the Barbara image. On this image, we also perform about 0.3 dB better than previous zerotree coding techniques that use the same transforms.

Since we need to understand block transforms before we can define group testing classes for them, we start out in section 8.1 by describing the block transforms we use, showing how they relate to the DWT, and giving an overview of image coding algorithms that use these transforms. We then present our GTBT algorithm in section 8.2, compare the results of our algorithm to the results of previously published image coders 8.3, and finally conclude in section 8.4.

8.1 Block Transform Background

8.1.1 Discrete Cosine Transform

The discrete cosine transform (DCT) decomposes a signal into coefficients representing functions of varying frequency. It is closely related to the *Fourier transform*, which is a standard mathematical method of representing a periodic signal as a linear combination of an infinite number of sinusoidal functions that vary in frequency. We can think of the Fourier transform as representing a function $f(t)$ with $\sum_{i=0}^{\infty} a_i g_i(t)$, where $\{a_i\}$ are the transform coefficients, and $\{g_i(t)\}$ are sinusoidal functions that increase in frequency as i increases. We can relate the DCT of a sequence to the transform coefficients of a discrete version of the Fourier transform as follows: Given a length n sequence $\{x_i\}$, construct a length $2n$ sequence by reflecting $\{x_i\}$ across one border. Take this length $2n$ sequence, and periodically extend it out to infinity, thinking of this infinite sequence as a function f . The coefficients of the discrete Fourier transform of f are the transform coefficients produced by the DCT on $\{x_i\}$.

The DCT of a length n vector \mathbf{x} is computed by multiplying \mathbf{x} by the $n \times n$ matrix \mathbf{C} ,

defined as follows:

$$C_{i,j} = \begin{cases} \sqrt{\frac{1}{n}} \cos\left(\frac{(2j+1)i\pi}{2n}\right), & \text{for } i = 0; j = 0, 1, \dots, n-1 \\ \sqrt{\frac{2}{n}} \cos\left(\frac{(2j+1)i\pi}{2n}\right), & \text{for } i = 1, 2, \dots, n-1; j = 0, 1, \dots, n-1 \end{cases}$$

Recall that coding gain (see section 2.4.1) is one widely used measure of the efficacy of a transform. It is known that the coding gain of the DCT on highly-correlated AR(1) sources is close to the optimal coding gain achievable by any transform for that source [36]. A highly-correlated AR(1) source is simply a mathematically defined statistical process for generating data where adjacent data points are probabilistically likely to have the same value. This source is frequently used as a model for images. The widespread use of DCT for image coding can be partly attributed to the fact that DCT performs close to optimal on the source used to model images.

In standard block transforms such as DCT, the input signal is divided into blocks of length M , with each block transformed separately into an output block of size M . For the forward DCT transform, the output consists of blocks of length M transform coefficients, where block i was obtained by multiplying input block i by C , a matrix of size $M \times M$. The first coefficient in each block represents an average of the values of the coefficients in that block; for historical reasons it is known as the *DC coefficient*. The other coefficients in a block are known as *AC coefficients*. The inverse transform is computed by multiplying each block by C^{-1} .

When applied to two-dimensional data such as images, the DCT is first performed in the vertical direction, and then in the horizontal direction. We can view this as dividing the image pixels into two-dimensional blocks of size $M \times M$, and applying the DCT to each column of the $M \times M$ block, and then applying the DCT to each row of the resultant data. Each input block of size $M \times M$ gets transformed into an output block of size $M \times M$, where the coefficient at $(0, 0)$ is the DC coefficient, and all other coefficients are AC coefficients.

8.1.2 *Lapped Transforms*

A lapped transform [36] is a generalization of the standard block transform where the input is divided into overlapping blocks of length L , with each block transformed into an output block of size M ; we call this an $M \times L$ lapped transform. An $M \times L$ lapped transform can be computed by multiplying the input row vector of length L with an $L \times M$ size matrix representing the transform, resulting in a output block of length M . In a typical example where $L = 2M$, each input data point is used in two adjacent output blocks. In this case, the inverse transform to recover one original block of M input data points is computed by taking two adjacent output blocks of coefficients ($2M$ coefficients total) and multiplying it with another $2M \times M$ matrix representing the inverse transform. In the two-dimensional case, we can view a lapped transform as mapping overlapping input blocks of size $L \times L$ into output blocks of size $M \times M$.

Unlike non-lapped block transforms, lapped transforms can take correlation between adjacent blocks into account; this makes it more efficient at decorrelating signals. Lapped transforms can also reduce blocking artifacts because their basis functions decay smoothly to near zero at their boundaries. Both lapped orthogonal transforms (LOT) and lapped biorthogonal transforms (LBT) have been studied. LBT's have more degrees of freedom than LOT's since the biorthogonality condition is weaker than the orthogonality condition. Aase and Ramstad [1] have shown that these extra degrees of freedom can be used to design better lapped transforms for image coding.

8.1.3 *Organization into Subbands*

The block-transform coefficients of an $LM \times LM$ image are typically stored in a block-by-block fashion, so that the output of a block transform that uses $M \times M$ blocks consists of an $L \times L$ grid of $M \times M$ blocks, where each block represents an $M \times M$ block of the original input image. However, we can conceptually reorder the transform coefficients into a grid of $M \times M$ subbands, each of size $L \times L$. This reordering puts all the DC coefficients into

one subband, ordered so that the DC coefficient in block (x, y) is at position (x, y) in the DC subband. Similarly, there will be a separate subband for each AC coefficient; subband (i, j) will contain AC coefficients from position (i, j) within their block, ordered so that the AC coefficient at position (i, j) in block (x, y) will be located at position (x, y) in subband (i, j) . Figure 8.1 illustrates this reorganization when $M = 4$ and $L = 6$.

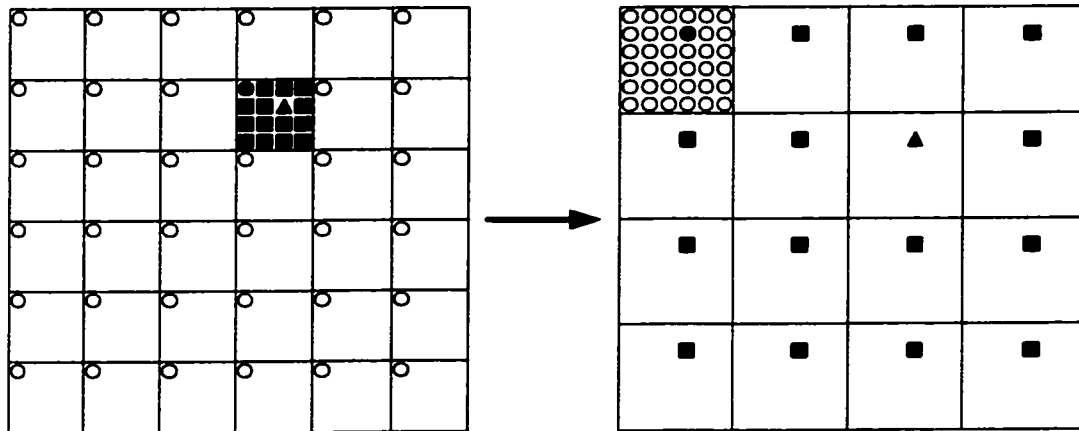


Figure 8.1: Coefficients from a block transform on the left are reorganized into subbands on the right. Circular coefficients are the DC coefficients that end up together in one subband; black coefficients from one block are scattered out to all subbands.

In this reorganized picture, each of the M^2 subbands represents the entire original image at a different frequency decomposition. Note that with this organization, these subbands are similar to the subbands from a dyadic wavelet decomposition in that coefficients in a subband represent the same frequency decomposition of an image over differing spatial locations. Furthermore, the upper-left block of DC coefficients (see figure 8.1) represents a postage-stamp size overview of the entire image, much like the lowest frequency subband in a dyadic wavelet decomposition. The principal difference between the dyadic wavelet decomposition and this reorganized block transform picture is that all the subbands from block-transforms are the same size, whereas in the wavelet transform, the subbands' sizes decrease by a factor of 2 with every additional level of the DWT performed. In other words, block transforms offer a *uniform-band* frequency partitioning of the input, in contrast to the

octave-band frequency partitioning of the wavelet transform.

For the DCT transform, the DC coefficient of an output block represents an average of the 8×8 input block. Since adjacent image blocks often are similar, adjacent coefficients in the DC subband will be correlated. For the lapped transforms, each traditional $M \times M$ output block is computed from an $L \times L$ input block of the original image. Most of the energy in the DC coefficient of the lapped transform is from the average of the entire $L \times L$ block. Since blocks are overlapping, some image pixels are used in more than one average and contribute their energy to adjacent coefficients in the DC subband.

8.1.4 Relation to the Wavelet Transform

With the subband organization, it becomes clear that we can also perform several levels of block transforms by recursively reapplying the block transform to the DC subband. We use the term *hierarchical block transform* to refer to any block transform scheme that decorrelates its DC subband by applying another transform. Note that hierarchical block transforms are similar to the levels of the DWT in a dyadic wavelet decomposition. Since the DC subband represents a small low-resolution overview of the entire image, we expect there to be significant correlation in the DC subband. Hierarchically reapplying a block transform to the DC subband should decorrelate it and enable better compression performance. We could continue to perform levels of the block transform as long as the lowest-frequency DC subband is not too small. Note that after every block transform step, we are always reorganizing the transform coefficients so that a DC subband is always present. Also note that in principle, any transform could be used to decorrelate the DC subbands; in addition to the lapped transforms and the DCT, even a DWT could be used to decorrelate the DC subband.

Another relationship between lapped transforms and the DWT is that a lapped transform can be thought of as a generalization of one level of the DWT. Recall that the output coefficients of a wavelet transform can be computed via convolution. For a k -tap wavelet transform, any one output coefficient depends on at most k consecutive input coefficients.

Thus, an $M \times (M + k)$ lapped transform can use the overlap of k data points on the input to compute the convolution of the input with the wavelet filter coefficients as would be done by the DWT. In other words, the DWT can be implemented as a lapped transform. Furthermore, hierarchical lapped transforms can completely implement the DWT's that use many levels. In its full generality, hierarchical block transforms should perform better than the DWT.

8.1.5 Previous Algorithms

JPEG

The most widespread image compression format using DCT is the standard JPEG [57] format. It uses 8×8 DCT blocks. The DC coefficients are predictively coded, meaning that the differences between adjacent DC coefficients are stored as symbols. For the AC coefficients, a simple quantization scheme is used to store them; there is a standard table dictating the quantization step size for each coefficient based on its position in the block. The symbols are entropy coded with a Huffman code [23], which is a well-known block-to-variable-length code (see section 3.3.2). Due to the advances in image compression and the old age of the JPEG standard, it is significantly worse than wavelet-based zerotree coders and other state-of-the-art coders in terms of rate-distortion performance.

EZ-DCT

Xiong *et al.*'s Embedded Zerotree DCT algorithm (EZ-DCT) [61] applied the zerotree technique to the DCT-transformed coefficients of an image. Although the coefficients of a DCT transform are not naturally tree-structured, this coder showed that by imposing a somewhat arbitrary tree structure on the coefficients, reasonable performance could be achieved. Their tree structure consisted of taking each 8×8 output block of the DCT transform, and organizing the coefficients of that block into a single tree by pretending that it was produced by a 3-level DWT of an 8×8 block.

This coder performed much better than JPEG, and slightly worse than wavelet-based zero-tree coders such as Shapiro's EZW. At the time, EZ-DCT was the best image coder that used the DCT transform.

EZ-LOT and EZ-LBT

Malvar applied the zerotree technique to lapped transform coefficients [35]. He basically used the same method as EZ-DCT, but replaced the DCT transform with lapped transforms. He defined an 8×16 LOT transform as well as a 8×16 LBT transform that were optimized for both image compression efficiency and low computational requirements. We use EZ-LOT (EZ-LBT) to refer to Xiong *et al.*'s embedded zerotree technique when applied to the Malvar's fast version of the LOT (LBT) transforms.

Tran's Hierarchical Lapped Transforms

Tran *et al.* [54] focused on designing the best lapped transforms for image compression, and did not consider the speed of computation to be a crucial factor. They designed several lapped transforms, including 8×40 generalized LOT (GLOT), 8×16 generalized LBT (GLBT), and 16×32 GLBT. These transforms were optimized solely for good coding performance on images. In his coding algorithm, he used a hierarchical coder, where after the first level of the lapped transform was performed, an additional 3 levels of the 9/7 Daubechies biorthogonal wavelet transform [3] were performed.

They also used a zerotree coding scheme on the transform coefficients that was very similar to the EZ-DCT scheme. One difference is that the hierarchical transform provides the opportunity to create deeper trees than previous non-hierarchical versions. We will use EZ-GLBT to refer to Tran *et al.*'s hierarchical zerotree coder when applied to the GLBT transform.

PTC

Malvar's Progressive Transform Coder (PTC) was an extension of his previous coder PWC (see section 2.6.5) that replaced the wavelet transform with a hierarchical LBT. This coder basically uses elementary Golomb codes to code the significance pass of a bit-plane. In PTC, one level of Malvar's fast LBT is performed, followed by a second level of DCT on the DC subband of the LBT transform.

The least complex transform (the DCT) was used at the second level because fast performance was one of the goals of this coder. Furthermore, a lower quality transform at the second level does not hurt the overall compression performance much. However, the quality of the first level transform does matter; using LBT at the first level results in much better compression performance than using DCT. Although the standard dyadic wavelet decomposition is slightly better for image compression than the fast LBT, the LBT also requires fewer computations. This makes the tradeoff worthwhile for some applications.

8.2 Group Testing for Block Transforms

In this section, we introduce our group testing for block transforms algorithm (GTBT) [22], which applies the group testing framework to code coefficients generated from block transforms.

Our technique will be identical to the GTW method presented in section 6.1.4 with the exception of the group testing class definitions. This means that after applying the block transform, we proceed by coding one bit-plane at a time with significance and refinement passes. The significance pass will be exactly like GTW's significance pass (procedure 6.1) except that it will use GTBT classes instead of GTW classes. Similar to GTW, we compute the average value of the DC subband and subtract it from the DC subband before starting the bit-plane encoding.

We will use the same GTBT class definitions for each of the block transforms we use. These transforms include the quickly computable LBT and LOT by Malvar, the GLBT by

Tran *et al.*, and the DCT. We will also try both hierarchical and non-hierarchical versions of each of these block transforms; the hierarchical transforms we study will decompose the DC subband with another block transform.

Our GTBT class definitions are based on the statistical characteristics of the transform coefficients generated from applying block transforms to images. We now proceed to present GTBT classes, followed by the ordering we use when we code these classes.

8.2.1 GTBT Classes

Exactly as in the definition of GTW classes, our GTBT classes have three different defining characteristics: the subband level, the significant neighbor metric, and the pattern type.

In the definition of classes, we will exclusively use the subband view of the block-transform coefficients, as shown in figure 8.1. Thus, we picture the transform coefficients as having M^2 subbands, where M is the output block size of the transform.

Subband Level. It is well known that for any particular block, the energy in the AC coefficients decrease as you move away from the DC coefficient. This means that coefficients in subbands closer to the DC coefficient are more likely to be significant than those in subbands farther away from the AC coefficient. To model this behavior, we classify subband (i, j) according to its distance from the DC coefficient at position $(0, 0)$.

The DC subband is in subband level 0. Subband level 1 contains those subbands (i, j) which satisfy $i + j = 1$ or $i + j = 2$. Subband level 2 contains those subbands (i, j) which satisfy $2 < i + j \leq 5$. Subband level 3 contains those subbands (i, j) which satisfy $5 < i + j \leq M$, where M is the output block size. Subband level 4 contains those subbands whose position (i, j) satisfy $M < i + j$, where M is once again the block size. Figure 8.2 shows the 5 subband levels. This method of defining the subband levels was made based on the two competing goals: preventing context dilution (by having fewer of subband levels) and making the probability of significance of coefficients in one subband level about the same (by having more subband levels).

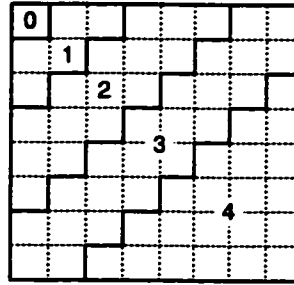


Figure 8.2: The subband levels of a block-transformed image in GTBT. Solid lines separate subband levels; dotted lines separate subbands.

For hierarchical transforms that have two levels, we basically double the number of subband levels; one set of levels contain subbands from the first transform, and the other set contains the subbands from the second-level transform of the DC subband of the first transform. Since a two-level hierarchical transform does not have any DC coefficients from the first level, there are actually only 9 subband levels in a two-level hierarchical transform. The subbands from the second level are considered more important than the subbands from the first level of the transform.

Significant Neighbor Metric. Looking at the statistics of the transformed coefficients, we found that adjacent coefficients in a subband showed statistical dependency on each other. That is, the probability that the coefficient is significant increases as more of its neighbors are found to be significant. Here, we consider the neighbors of a coefficient to only be the 8 coefficients in the same subband that surround a coefficient. Thus, just like in GTW classes, we define 4 values in the significant neighbor metric: 0, 1, 2, and 3+, depending on whether 0, 1, 2, or more than 2 neighbors are significant. Because there are only up to 8 neighbors, the maximum neighbor count is 8. Note that this definition is a simplification of the original GTW scheme that omits parent-child relationships.

Pattern Type. This characteristic is exactly the same as in GTW, where the coefficients in a subband are assigned into 4 different pattern types based solely on its position in a subband.

For non-hierarchical transforms, there are 5 subband levels, 4 significant neighbor types, and 4 pattern types resulting in 80 classes total. For two-level hierarchical transforms, there are 9 subband levels, resulting in 144 classes total. These classes are ordered according to the same ordering as the GTW classes, namely, with significant neighbor metric first, pattern type second, and subband level last.

8.3 Results

Here we present our results on some standard 8-bit monochrome images: the 512×512 images Barbara, Goldhill, and Lena (available from [28]); and a 768×768 fingerprint image from the FBI's fingerprint compression standard [5]. We present results for our GTBT on several different transforms: the 8×8 DCT; the 8×16 type-I fast LOT with angles $(\theta_0, \theta_1, \theta_2) = (0.145\pi, 0.17\pi, 0.16\pi)$ [36]; the 8×16 LBT from [33]; and the 8×16 GLBT from [54]. For convenience, we refer to GTBT using the DCT, LOT, LBT, or GLBT transforms as GT-DCT, GT-LOT, GT-LBT, and GT-GLBT, respectively.

First, we compare PSNR results of a non-hierarchical version of GT-DCT with Xiong *et al.*'s EZ-DCT scheme; their results are taken from [61]. As table 8.1 shows, GT-DCT performs somewhat better than EZ-DCT on Barbara, and about the same on Lena. The 0.3 dB difference on Barbara illustrates the improvement from using group testing instead of zerotree coding.

Next, we show our results for hierarchical versions of the four different transforms we considered. For a given first level block transform, we did try using many different transforms at the second level of the hierarchy. However, we found that the rate-distortion performance difference between different transforms at the second level was insignificant. Using any second level transform did show a noticeable improvement over the non-

Table 8.1: Comparing PSNR results of coders using the non-hierarchical 8×8 DCT transform.

<i>Image</i>	<i>Algorithm</i>	<i>Rate (Bits/pixel)</i>			
		0.25	0.5	0.75	1.0
Barbara	GT-DCT	27.25	31.14	34.06	36.20
	EZ-DCT	26.83	30.82	33.70	36.10
Lena	GT-DCT	32.31	35.90	38.06	39.52
	EZ-DCT	32.25	36.00	38.06	39.62

hierarchical version; however, the specific transform used at the second level did not appear to be important. Given this fact, we only present results for hierarchical transforms that apply the same transform for the first and second level of the hierarchy. PSNR Results are shown in table 8.2.

Note that the transforms we use all produce 8×8 output blocks of coefficients, and 64 total subbands. This implies that the size of the DC subband is a factor of 64 smaller than the size of the input. Thus, after a two level hierarchy on 512×512 images, the size of the second level DC subband is 8×8 ; this is small enough that applying another level of transform would not help.

In general, coding performance improves as we proceed in order from DCT to LOT to LBT to GLBT. This is expected, because each transform is of higher quality than the previous one. LOT is better than DCT because it is lapped; LBT is a general form of the LOT, and the GLBT throws away the fast-computability characteristic inherent in the design of LBT.

It also appears that the lapped transform methods perform much better than the standard GTW on only Barbara and fingerprint, the two images containing the most edges. As mentioned in the end of section 2.4.2, these edges lead to more energy in the coefficients representing the high-frequencies. This suggests that the lapped transforms are better than the dyadic wavelet decomposition at decorrelating high frequency content. One possible

Table 8.2: Comparing PSNR results of various algorithms. GTW is used as the baseline, so that for algorithm A, $\Delta A = A - \text{GTW}$.

<i>Image</i>	<i>Algorithm</i>	<i>Rate (Bits/pixel)</i>			
		0.1	0.25	0.5	1.0
Barbara	GTW	24.37	27.87	31.59	36.47
	$\Delta\text{GT-DCT}$	-.28	-.30	-.26	-.11
	$\Delta\text{GT-LOT}$	+.81	+1.19	+1.25	+1.09
	$\Delta\text{GT-LBT}$	+1.11	+1.53	+1.67	+1.46
	$\Delta\text{GT-GLBT}$	+1.20	+1.64	+1.74	+1.61
Lena	GTW	30.06	34.13	37.27	40.51
	$\Delta\text{GT-DCT}$	-1.87	-1.63	-1.23	-.86
	$\Delta\text{GT-LOT}$	-1.11	-.99	-.79	-.62
	$\Delta\text{GT-LBT}$	-.59	-.53	-.39	-.39
	$\Delta\text{GT-GLBT}$	-.60	-.37	-.21	-.31
Goldhill	GTW	27.76	30.46	33.10	36.47
	$\Delta\text{GT-DCT}$	-.70	-.57	-.56	-.49
	$\Delta\text{GT-LOT}$	-.25	-.04	-.09	-.08
	$\Delta\text{GT-LBT}$	-.05	+.10	+.13	+.09
	$\Delta\text{GT-GLBT}$	-.06	+.12	+.08	+.14
Fingerprint	GTW	28.35	32.80	36.06	39.98
	$\Delta\text{GT-DCT}$	-1.19	-1.09	-.49	+.24
	$\Delta\text{GT-LOT}$	-.22	-.30	+.15	+.88
	$\Delta\text{GT-LBT}$	+.03	+.06	+.45	+1.05
	$\Delta\text{GT-GLBT}$	+.17	+.20	+.67	+1.35

reason for this behavior is that the lapped transforms offer a uniform-band frequency partitioning of the input, compared with the octave-band partitioning found in the wavelet transform. As mentioned in [54], the finer frequency partitioning increases the frequency resolution that can often generate more insignificant coefficients.

It is interesting to note that the wavelet packets version (GTWP) also handles images with lots of high-frequency content well. Surprisingly, the GT-GLBT outperforms GTWP by about 0.30 dB on the Barbara image. This is especially significant because GTWP already outperforms GTW on this image by about 1.3 dB.

As expected, GT-GLBT shows some performance improvement over Tran *et al.*'s Embedded Zerotree technique applied to the GLBT (EZ-GLBT). Improvements of up to 0.45 dB were observed on the Barbara image, as illustrated in the PSNR curves of figure 8.3. EZ-GLBT results are taken from [54]. This figure shows the performance gain of using group testing instead of zerotree coding in image compression.

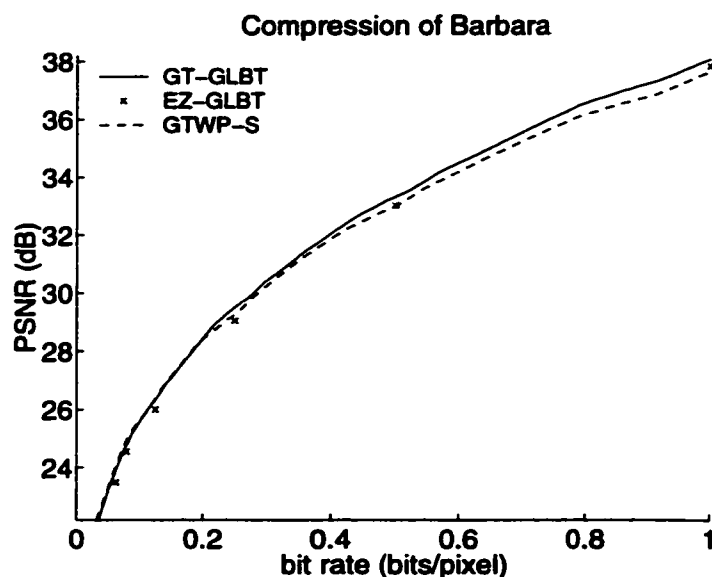


Figure 8.3: Comparing performance of coders using the GLBT transform and GTWP-S.

8.4 Conclusion

In conclusion, we have shown that the group testing framework is flexible in that it can be easily adapted to code coefficients from block transforms. We have also shown that the group testing of both DCT and GLBT transform coefficients improves upon zerotree coding of these coefficients by an average of 0.3 dB, over a range of bit-rates for the Barbara image. Finally, we have also shown that the GLBT transform can lead to results that surpass even our previous GTWP algorithm by about 0.3 dB. Our coder shows one way of effectively coding block transform coefficients. It gives more insight on the statistical characteristics of block-transformed coefficients, insight which can be used to create better image coders using block-transforms. Thus, our image coder is an interesting addition to the field of image compression.

Chapter 9

THEORETICAL PERFORMANCE OF GROUP TESTING

In this chapter, our goal is to explore the efficacy of group testing when used on transform coefficients of an image. We are particularly concerned with group testing's efficacy in the context of GTW, the image coder defined in chapter 6. To explore this problem, we first define two different probabilistic models for representing the bits of the transform coefficients: an independence model that does not model dependencies between bits, and a dependence model that does include some probabilistic dependencies.

We study the independence model because it is simple and easy to analyze. However, we know it is not completely accurate because the coefficients do in fact have probabilistic dependencies with each other (see section 2.5). This motivated us to study the dependence model; although it is more complicated, it is also a more accurate representation of the bits to be compressed. Understanding group testing's performance when the input bits are probabilistically dependent helps us obtain a more accurate view of group testing's performance on images.

The independence model is basically composed of simple i.i.d. binary sources, while the dependence model is composed of the two-state Markov model first discussed in section 5.3.4. For both i.i.d. and Markov sources, we define group testing methods for efficiently encoding the source, calculate the bit-rates of the different methods, and compare the bit-rates with the entropy of the appropriate source. The bulk of this chapter is concerned with showing the effectiveness of group testing as a general entropy coder on these two sources. Although we are primarily concerned with examining group testing's performance on images, this chapter also shows the performance of group testing on any data that can be modeled by an i.i.d. binary source or a two-state Markov source.

The group testing methods we study for encoding sources are all based on using group iterations (procedure 6.2) to code source bits. For the i.i.d. source, our method is to repeatedly perform group iterations of an appropriate size. For the two-state Markov source, we present three different encoding techniques: simple sequential coding, two-state sequential coding, and interleaved coding. For each technique, we calculate the bit-rate of the simple version of these methods with group iteration size two. We keep the group iteration size small so that the methods can be more easily analyzed.

In section 9.1, we discuss general characteristics of transform coefficients of an image in order to determine how they should be modeled. In section 9.2, we introduce the independence model. We also discuss the efficacy of group testing on the model, as well as the model's relation to actual images. Section 9.3 introduces the dependence model that uses Markov sources and discusses its relation to actual images. Section 9.4 studies efficacy of the three different group testing methods for the Markov source. As we introduce each method, we also relate them to the actual group testing methods used in GTW. We summarize and draw some conclusions about the capabilities of different group testing methods in section 9.5.

9.1 *Modeling Image Transform Coefficients*

In this thesis, the image compression algorithms that we study are all transform coders that use bit-plane coding. That is, the coders apply a transform to the image and encode the transform coefficients in a bit-plane by bit-plane manner. To study the theoretical performance of this type of compression, we first need to define a probabilistic model for the bits in each bit-plane. This is necessary before the bit-rates of different group testing techniques can be calculated.

Note that although we principally talk about modeling wavelet transform coefficients, the models that we study apply equally well to coefficients generated from other transforms. Similarly, although we explain our methods of encoding the models by relating them to the

GTW algorithm, they are also related to GTWP and GTBT.

9.1.1 *Bit Characteristics of Transform Coefficients*

We can apply knowledge about the statistical characteristics of wavelet coefficients (see section 2.5) to infer the statistical characteristics of the bits in a bit-plane. We know that the bits in a bit-plane can be divided into refinement bits and significance bits, depending on whether they were coded in the significance pass or the refinement pass. For refinement bits b , $\Pr[b = 1] \approx 0.5$. For significance bits b , $\Pr[b = 1]$ is typically close to zero, but can vary widely for bits in differing subbands as well as bits with differing number of significant neighbors.

Furthermore, the significance bits in earlier bit-planes have a higher probability of being 1 than the significance bits in later bit-planes. Using the Barbara image as an example, only a few hundred coefficients are significant in the first few bit-planes. This is out of a total of 512×512 coefficients, implying that $\Pr[b = 1] \approx 0.0001$ for significance bits b in the first few bit-planes. By the bit-plane 7, there are about 10,000 significant coefficients in the Barbara image, implying $\Pr[b = 1] \approx 0.04$ for significance bits b in bit-plane 7.

The significance bits exhibit dependencies on each other, with bits both in the same bit-plane as well as bits in other bit-planes. This is directly implied by our knowledge about the dependencies between coefficients; we illustrate this by using the dependencies between spatially adjacent coefficients as an example. For coefficient c , let $b_k(c)$ denote the k th bit of c , i.e. the bit of c encoded by bit-plane k . Let $b_k(c)$ be a significance bit of coefficient c , and let c_a be a coefficient adjacent to c . Then if $b_j(c_a) = 1$ for some $j \leq k$, then c_a is significant and $b_k(c)$ is much more likely to be significant; if $b_j(c_a) = 0$ for all $j \leq k$, then c_a is insignificant, and $b_k(c)$ is less likely to be significant. Thus, $b_k(c)$ is dependent upon $b_j(c_a)$, for any adjacent coefficient c_a for all $j \leq k$.

Of course, there are other dependencies besides those between adjacent coefficients, such as those between parent coefficients and their children.

Based on the above observations, we could construct a probabilistic model that gener-

ates the bits of each bit-plane in accordance with the known dependencies and probabilities of significance. However, because there are many dependencies, such a model would be fairly complex; it would be difficult to analytically determine the bit-rate of applying group testing to this complex model. Instead, we construct much simpler models that generate the bits of a bit-plane. In these simpler models, the bit-rate of applying group testing can be analytically determined. The analysis of these simpler models gives us insight on group testing's performance on real images.

9.2 Independence Model

In this section, we define a simple independence model for generating the transform coefficient bits of an image. We then show how to use group testing to encode the bits from this source model, and compare the bit-rate of group testing with the entropy of this model. We then relate the results of analyzing this model to the GTW algorithm.

In the independence model, we assume the bits are generated independently. However, we do not assume the bits are identically distributed. Under this model, an n -bit source would be specified by n numbers, (p_1, \dots, p_n) , where $p_i = \Pr[\text{bit } i = 0]$. To model the bit-planes of the transform coefficients, the p_i 's for refinement bits would be close to 0.5, and the p_i 's for bits in bit-plane one would be close to 1.

9.2.1 Encoding Independence Model

Since the p_i 's are known for all bits, one way to encode this source is to divide the bits into classes based on p_i , and separately encode each class. We define the classes so that all bits in any class have approximately the same probability of being significant. We encode each class using group iterations, where the size of the group iteration is chosen optimally based on the p_i 's for the bits in that class. In effect, we are reorganizing the bits of the model so that they seem to be coming from a composite source consisting of many i.i.d. binary sources, where each binary source has a different probability of generating 0's. As long as

n is large enough, our method of dividing the bits into classes is approximately the same as using group iterations on this composite source.

9.2.2 *Group Testing Performance on Independence Model*

Figure 5.1 in section 5.1.2 shows the efficacy of performing group iterations on an i.i.d. source relative to entropy when the optimal group iteration size is used. As can be seen, the group testing bit-rate is always within 5% of entropy. On a composite source consisting of i.i.d. binary sources, the group testing bit-rate would be a weighted average of the bit-rates for each binary source, where the weights depend on the relative proportion of bits produced by each source. Since all the bits are independent, the entropy of the composite source is a weighted average of the entropy of each binary source. Thus, we can see that group testing bit-rate on the composite source is also within 5% of the entropy of the composite source. This suggests that group testing applied to the independence model is also within 5% of entropy.

To complete the analysis of this model, we present one method for calculating the bit-rate of using group iterations on an i.i.d. binary source in section 9.2.4.

9.2.3 *Relation to GTW*

The independence model examines the performance of compressing the transform coefficients under the assumption that all bits of the transform coefficients are independent of each other. The technique of using different classes for bits with different probabilities is clearly similar to GTW's use of GTW classes. Our method of encoding the independence model represents the performance of GTW when given a black box capable of perfectly classifying each bit, giving every bit's conditional probability of significance given the values of all other bits.

Unfortunately, a black box capable of perfect classification is impossible to construct. Since the bits are encoded bit-plane by bit-plane, the values of the bits in previous bit-

planes are known when encoding the current bit-plane. This knowledge makes it possible to separate the significance bits from refinement bits. However, it is not possible to perfectly classify all the significance pass bits. As an example, a classifier cannot always separate out coefficients with many significant neighbors from coefficients without many significant neighbors, because the values of the neighbors at the current bit-plane are not yet known when the bit-plane is being encoded.

We can view GTW's use of GTW classes as an attempt to approximate this perfect classification black box, based on all the available information about the coefficients. We believe that the GTW classes provide a reasonable approximation to the black box because of GTW's excellent performance compared with other image coders. Although the GTW classes do not perfectly represent the black box, examining the performance of encoding the independence model still provides useful insight on the theoretical performance of GTW.

Recall that GTW only uses group testing on the significance pass bits, and that the refinement pass bits are left uncompressed. If we were to apply group testing to the refinement pass bits, then a group iteration size of one would be optimal since the probability of significance of these bits is about 0.5; this is exactly the same as leaving these bits uncompressed. Thus, we can view the overall GTW algorithm as dividing all the bits in a bit-plane into classes and coding the classes with group iterations, where there is one class representing the refinement bits that will be coded with group iteration size one. Our efficacy results of encoding the independence model represents encoding the entire image, not just the performance of encoding all the significance pass bits in GTW.

9.2.4 Group Testing Bit-rate for i.i.d. Sources

In this subsection, we present one method for calculating the bit-rate of using group iterations on an i.i.d. binary source. Because making a group iteration results in an elementary Golomb code (see section 5.1.2), calculating the bit-rate of using repeated group iterations is equivalent to calculating the bit-rate for the corresponding elementary Golomb code. We can use equation 3.5 to calculate the bit-rate of an elementary Golomb code on a i.i.d.

source. We are given a binary i.i.d. source S_p where p is the probability that a source bit is 0. Let k be the group iteration size that we choose; our goal is to calculate the bit-rate of EG_k (defined in section 3.3.4) on this source S_p .

Let $(b_0, b_1, \dots, b_k) = (1, 01, 001, \dots, 0^{k-2}1, 0^{k-1}1, 0^k)$ be the $k + 1$ input messages of EG_k , and let (c_i) be the corresponding $k + 1$ output codewords such that b_i maps to c_i . We now summarize our knowledge about this code:

$$|b_i| = \begin{cases} i + 1, & \text{for } 0 \leq i < k \\ k, & \text{for } i = k \end{cases} \quad (9.1)$$

$$\Pr[b_i] = \begin{cases} (1 - p)p^i, & \text{for } 0 \leq i < k \\ p^k, & \text{for } i = k \end{cases} \quad (9.2)$$

$$|c_i| = \begin{cases} 1 + \lfloor \log_2 k \rfloor, & \text{for } 0 \leq i < k' = 2^{\lfloor \log_2 k \rfloor + 1} - k \\ 2 + \lfloor \log_2 k \rfloor, & \text{for } k' \leq i < k \\ 1, & \text{for } i = k \end{cases} \quad (9.3)$$

Note that although the expression $2 + \lfloor \log_2 k \rfloor$ given for $|c_i|$ when $k' \leq i < k$ is different from the expression $1 + \lceil \log_2 k \rceil$ given in section 3.3.4, these two expressions are in fact equivalent.

Using equation 3.5, the bit-rate can be calculated as

$$\frac{\sum_{i=0}^k \Pr[b_i] |c_i|}{\sum_{i=0}^k \Pr[b_i] |b_i|}.$$

We can plug in the appropriate values from equations 9.1 – 9.3 and use Mathematica to solve for the final result:

$$R(EG_k, S_p) = (1 - p) \left(\lfloor \log_2 k \rfloor + 1 + \frac{p^{k'}}{1 - p^k} \right), \quad (9.4)$$

where $k' = 2^{\lfloor \log_2 k \rfloor + 1} - k$.

Recall (from section 3.3.4) that $k^*(p)$ refers to the optimal group iteration size on source S_p ; it is chosen to satisfy the inequalities 3.6. Note that using the above expression for the bit-rate, it is possible to prove that the integer k satisfying the inequalities 3.6 is the integer that minimizes the bit-rate over all possible group iteration sizes. Also recall (from section 5.1.2) that $\text{GTR}(p)$ refers to the group testing bit-rate on source S_p using the optimal group iteration size. This means $\text{GTR}(p) = R(\text{EG}_{k^*(p)}, S_p)$, and that $\text{GTR}(p)$ is defined by equation 9.4 with k set to $k^*(p)$.

9.3 Modeling Dependencies

In this section, we define a model for generating the transform coefficient bits of an image that includes dependencies between the bits. We first discuss general issues about modeling dependencies, before explaining the actual dependence model. We then discuss the accuracy of the model in accounting for the transform coefficient bits. Bit-rates for the dependence model are calculated in section 9.4.

Our primary goal in defining a dependence model is to represent the statistics of transform coefficient bits more accurately than possible with a model assuming independence. A secondary goal is to keep the model simple, so that performance of different group testing methods can be more easily determined analytically. The dependencies between coefficient bits in our model gives us more confidence in the applicability of the our group testing performance results towards real images.

9.3.1 Choosing Dependencies to Model

As previously mentioned in section 9.1.1, trying to model all the dependencies present in the transform coefficient bits would lead to an unwieldy, complex model. In choosing the appropriate dependencies to model, we are motivated by the following fact:

Given two bits b_1 and b_2 , and the associated probabilities $\Pr[b_1 = 1]$ and $\Pr[b_2 = 1]$, if b_1 and b_2 are probabilistically independent, then their total entropy is greater than if

they are dependent on each other. Furthermore, the more dependent they are upon each other, the less total entropy they have. This means that to obtain a good approximation of the transform coefficient bits, we should include the dependencies between bits that are strongest in our model. In order help the model to remain simple, we can ignore the dependencies between bits that are not that strong, but still exist.

We know from our image coding work that the strongest dependencies are between coefficients that are spatially adjacent to each other in the same subband. Thus, our dependence model will include dependencies between adjacent bits, and ignore other dependencies such as those between a parent coefficient and a child coefficient in the dyadic wavelet decomposition. Another advantage of using this simplified model of the dependencies is that it also works equally well for the wavelet packet transform and block transform coefficients.

Another simplification that we make in the dependence model is that the dependence model consists of one-dimensional streams of bits, while transform coefficient bits are inherently two-dimensional in nature. One-dimensional models have many fewer dependencies than their two-dimensional counterpart; this makes it easier to analytically determine the bit-rates of using group testing on the one-dimensional models.

Since the entropy of dependent bits is less than the entropy of independent bits, we expect algorithms that do not know about bit dependencies but assume bits are independent to perform worse than algorithms that know about the bit dependencies. Designing compression algorithms with the dependencies in mind gives the algorithms a chance to exploit the dependencies to obtain a lower overall bit-rate.

9.3.2 *Dependence Model*

Our dependence model is a composite source, consisting of a set S ; each element of S is a source generated from the two-state Markov chain shown previously in figure 5.3. Note that although the elements of S are generated independently from each other, the bits that any one element of S produces do have dependencies with each other. Each element of S

has its own parameters p and q , where p is the probability of transitioning from state 0 to state 1, and q is the probability of transitioning in the reverse direction. We assume each source starts out in the stationary distribution.

To encode this source with group testing, we simply need to specify a method of encoding $S_{p,q}$, which is the Markov source with parameters p and q . Compressing the composite source would consist of applying our group testing method to each individual Markov source. It is easy to see that if the bit-rate for our group testing method on $S_{p,q}$ for any values of p and q is always within $x\%$ of the entropy of $S_{p,q}$, then the bit-rate for the composite source is also within $x\%$ of the entropy. Section 9.4 compares the entropy of $S_{p,q}$ with many different group testing methods for encoding $S_{p,q}$.

9.3.3 *Relation to Transform Coefficient Bits*

Markov Sources for a Bit-plane

We first look at how Markov sources can represent the bits in one bit-plane of the transform coefficients. The bits in a bit-plane can be divided into significance pass bits and refinement pass bits. We can model the refinement pass bits as generated by the source $S_{0.5,0.5}$; thus, these bits are generated independently and uniformly at random. For significance pass bits, we model each subband as generated by a different Markov source with different parameters p and q . Thus, we are ignoring dependencies between subbands, and modeling only dependencies between neighboring bits.

The dependencies in the Markov source are similar to those present in a subband in that bits of similar value tend to be clustered together. Thus, 1 bits are more likely to be found next to other 1 bits, and likewise for 0 bits. However, it is clear that the Markov source is not a completely accurate representation of the dependencies within a subband. As previously mentioned, the main difference is that the Markov source generates a one-dimensional stream of bits, while subbands are two-dimensional in nature, with dependencies existing between horizontally and vertically adjacent neighbors. In this work, we take

the path of using simple dependency models so that the bit-rates of using group testing on these models can be more easily determined. Since the one-dimensional model is a special case of the two-dimensional model, we can view our work as a stepping stone to dealing with the two-dimensional model.

Markov Sources for All Bit-planes

We know that the probability characteristics of a bit-plane differ from one bit-plane to the next; earlier bit-planes have fewer 1 bits in their significance pass. Thus, we use separate Markov source models for each bit-plane. For the significance pass bits, one Markov source is used to model the bits from one subband of one bit-plane. Although this seems to imply that our model ignores all dependencies between bit-planes, this need not be the case. We know that image compression techniques code the bits in bit-plane order, so that all previous bit-planes are known before starting to code the current bit-plane. Thus, the parameters p and q for the Markov sources in the second bit-plane can be specified dependent upon the actual values of the bits in the first bit-plane. In other words, if a subband happened to have more significant coefficients than expected in the first bit-plane, the value of p for that subband in the second bit-plane could be higher than it would have been otherwise. Similarly, the p 's and q 's for sources in bit-plane i can be dependent on the values of all bits in previous bit-planes. Note that knowledge of previous bit-planes is already assumed when dividing bits into significance and refinement passes.

Choosing p and q

We can use our knowledge about transform coefficients to estimate reasonable values for the parameters p and q in our Markov source. For the first few bit-planes, there are very few significant coefficients, and p should be very close to zero. For any given subband, the value of p should increase with each successive bit-plane. Furthermore, lower level subbands that contain more significant coefficients should have higher values of p than

higher level subbands.

Values of p between 0 and 0.5 appear to be reasonable for modeling the bits of a subband. Since most transform coefficients are close to zero, we expect most subbands to be modeled by sources where p is close to zero. However, there are subbands where $p \approx 0.4$ is not an unreasonably high value for its associated Markov model. Looking at figure 6.8, we see that when a coefficient has a significant neighbor metric of 6 out of a maximum of 12, then its probability of significance is about 0.4. These coefficients are probably found only in subbands where a high percentage of the coefficients in that subband are significant. This would most likely happen in one of the lower level subbands at one of the higher bit-planes. In fact, for any subband, there is a threshold level at which approximately $\frac{1}{2}$ the coefficients are significant. This means that for some bit-plane i , we expect about $\frac{1}{2}$ the significance bits to be 1. Since lower frequency subbands have larger coefficient values, the bit-plane in which a significant fraction of the bits become significant is lower for low frequency subbands. It can be empirically verified that when encoding an image at a high quality setting, the number of bit-planes encoded surpasses this threshold level for the subbands in the lower levels.

The parameter q controls how likely the 1's are to be clustered, with lower values corresponding to longer clusters of 1's. As we explain below, values of q between p and $1 - p$ appear to be reasonable for modeling the bits of a subband. In the Markov sources applicable to our modeling task, q should be low enough so that there is a significant probability of generating clusters of 1's. Note that if $q = 1 - p$, then the Markov source is in fact an i.i.d. source and no clustering of 1's occurs. If $q < 1 - p$ then some clustering occurs, whereas if $q > 1 - p$ then the 1's are unlikely to be clustered together. Thus, $q < 1 - p$ is necessary to ensure clustering occurs.

An additional constraint on q is that q should not be so low that too many 1's are generated by the source. Figure 6.8 shows that the probability that any significance bit is 1 never rises significantly above 0.5, even for bits whose neighbors are all significant. This implies that when coding the significance pass bits of any particular subband, we expect no

more than $\frac{1}{2}$ the bits to be significant. Thus, our source that models the bits in a subband should generate no more 1 bits than 0 bits. We need to ensure that $\pi_0 \geq 0.5$, where π_0 is the probability that the Markov chain is in state 0 in the stationary distribution. As shown in section 9.4.1, π_0 is easily calculated to be $q/(q + p)$, which implies that choosing $q \geq p$ ensures this additional constraint.

In our work, we examine our model for all values of p in the range $0.1 < p < 0.5$ when $q = 0.5$ and $q = 0.4$. Although this is a somewhat arbitrary choice, this choice is motivated by several factors. One factor is that $q = 0.5$ satisfies the reasonableness condition $p \leq q < 1 - p$ for the values of p that we test. In addition, when $q = 0.5$, the expected length of a run of 1 bits is 2. We feel this is a reasonable approximation of the occurrence of clusters of significant coefficients in image transform coefficients. Finally, we chose not to examine the model when $p < 0.1$ because of the complexity of analytically determining the bit-rate for appropriately designed group testing methods for small p . With small p , it is necessary to use large group iteration sizes to obtain good compression performance. Since we only analytically determine the performance of group testing methods with group iteration sizes 2 or smaller, our results are not applicable to Markov models with small p .

9.4 Two-state Markov Chain Model

In this section, we analyze the performance of encoding the two-state Markov chain model with the different methods of group testing. We start by defining the two-state Markov source and calculating its entropy. We then present three different methods for encoding this source: simple sequential coding, two-state sequential coding, and interleaved coding. We relate each method to the corresponding image compression algorithm that the method models. We also calculate the bit-rates of these methods on our Markov source, and compare them to the entropy of the source.

In the simple sequential coding method, we repeatedly perform group iterations of the

same size to encode the source. In the two-state sequential coding method, we repeatedly perform group iterations of two different sizes; the size we use depends upon the state we are in. In the interleaved coding method, we no longer encode the source bits in order, but instead start by coding every other bit before coming back to encode the uncoded bits. These methods are explained in more detail in the following subsections.

The three methods are ordered from simplest to most complicated to implement. As we progress to the more complex methods, the method more closely resembles the actual method for encoding bits used in GTW. For each of these methods, we calculate the bit-rate assuming we use a group iteration size of two. Keeping the group iteration size small makes the bit-rate calculation tractable. We then compare our group testing methods to each other, and to the entropy of the source.

9.4.1 Two-state Markov Chain Source

We now present some notation for the binary source that generates bits according to the two-state Markov chain as described previously in section 9.3.2. Let $S_{p,q}$ represent this source, where p is the probability of transitioning from state 0 to state 1, and q is the probability of transitioning in the reverse direction. The source starts out in the stationary distribution $\pi = (\pi_0, \pi_1)$, where π_i is the probability of starting in state i . We also use b_i to denote the i th bit produced by the source; $b_i = 0$ if the i th step of the Markov chain transitions into state 0; otherwise $b_i = 1$. Furthermore, we define $\bar{p} = 1 - p$ and $\bar{q} = 1 - q$ to make some of the formulas more readable.

Note that the stationary distribution π is fairly simple to calculate for this Markov chain; solving the simultaneous equations:

$$\begin{aligned}\pi_0 &= \bar{p}\pi_0 + q\pi_1 \\ \pi_0 + \pi_1 &= 1\end{aligned}$$

leads us to conclude that

$$(\pi_0, \pi_1) = \left(\frac{q}{p+q}, \frac{p}{p+q} \right).$$

9.4.2 Entropy of Markov Source

The entropy of Markov chain sources is known to be the average value of the entropy at each state [46]; for a chain C with m states U_i , we have

$$H(C) = \sum_{i=1}^m \mathbf{Pr}[U_i] H(U_i). \quad (9.5)$$

Thus, the entropy of $S_{p,q}$ is simply

$$H(S_{p,q}) = \pi_0(-p \log_2 p - \bar{p} \log_2 \bar{p}) + \pi_1(-q \log_2 q - \bar{q} \log_2 \bar{q}). \quad (9.6)$$

9.4.3 Simple Sequential Coding

One way of encoding $S_{p,q}$ with group testing is by encoding the bits of the output source sequentially. The most obvious way to encode $S_{p,q}$ is to pick a group iteration size k that is suited for the source, and to then encode the bits of the source with repeated group iterations of size k . In this method, k never changes. We denote this method $SC(k)$. Intuitively, this is not the best method because the distribution of the source symbols changes depending on the current state of the Markov chain. With different probability distributions, different group iteration sizes could perform better.

We would expect the bit-rate of this simplistic method to be about $GTR(\pi_0)$, since this method “assumes” the underlying source $S_{p,q}$ is like the i.i.d. source S_{π_0} . With this intuition, we would expect the optimal value of k for this source to be about $k^*(\pi_0)$. In the context of image compression, this method represents encoding the bits ignoring the existing dependencies and assuming the transform coefficient bits are in fact independent. Since the source is a model of the bit-plane of a subband, we can think of this method as defining GTW classes based solely on the subband level, and not upon the significant neighbor metric nor on the pattern type.

9.4.4 Bit-rate of Simple Sequential Coding

Here we calculate $R(\text{SC}(2), S_{p,q})$, the bit-rate for simple sequential coding using group iteration size two. We can calculate the bit-rate by using recurrence equations. Let U_n represent the expected number of bits used to encode n bits from $S_{p,q}$, given that the last bit from the source was a 0. Let V_n represent the expected number of bits used to encode n bits from $S_{p,q}$, given that the last bit from the source was a 1. We can think of U_n as encoding a string of length n knowing that the source is in state 0 of its Markov chain. Similarly, the V_n 's know that the source is in state 1 of its Markov chain.

$U_0 = V_0 = 0$	Initial Conditions
$U_1 = V_1 = 1$	
$U_n = \bar{p}\bar{p}(1 + U_{n-2})$	(Processing 00)
$\quad + \bar{p}p(2 + V_{n-2})$	(Processing 01)
$\quad + p(2 + V_{n-1})$	(Processing 1?)
$V_n = q\bar{p}(1 + U_{n-2})$	(Processing 00)
$\quad + qp(2 + V_{n-2})$	(Processing 01)
$\quad + \bar{q}(2 + V_{n-1})$	(Processing 1?)

Note that 1? means we saw and encoded a 1, and the value of next bit ? is still uncoded and will be coded next. To solve this recurrence, we use the method of generating functions presented in [19]. First we simplify the recurrence equations, obtaining:

$$U_n = a_1 U_{n-2} + a_2 V_{n-2} + a_3 V_{n-1} + a_4 \quad (9.7)$$

$$V_n = b_1 U_{n-2} + b_2 V_{n-2} + b_3 V_{n-1} + b_4, \quad (9.8)$$

where

$$\begin{aligned}
 a_1 &= \bar{p}^2 \\
 a_2 &= \bar{p}p \\
 a_3 &= p \\
 a_4 &= a_1 + 2a_2 + 2a_3 \\
 b_1 &= q\bar{p} \\
 b_2 &= qp \\
 b_3 &= \bar{q} \\
 b_4 &= b_1 + 2b_2 + 2b_3.
 \end{aligned}$$

Next, we put the recurrence equations in generating function form. We illustrate this process for equation 9.7. The first step is to write down a single equation that expresses U_n for all values of n , assuming that $U_n = 0$ and $V_n = 0$ for all $n < 0$. In this case, we get the equation:

$$U_n = a_1 U_{n-2} + a_2 V_{n-2} + a_3 V_{n-1} + a_4 - a_4 \delta_{n0} + (1 - a_4) \delta_{n1},$$

where

$$\delta_{nk} = \begin{cases} 0, & \text{for } n \neq k \\ 1, & \text{for } n = k. \end{cases}$$

Next, we multiply both sides by z^n and sum over all n . This produces:

$$\begin{aligned}
 \sum_n U_n z^n &= \sum_n a_1 U_{n-2} z^n + \sum_n a_2 V_{n-2} z^n + \sum_n a_3 V_{n-1} z^n + \sum_n a_4 z^n \\
 &\quad - \sum_n a_4 \delta_{n0} z^n + \sum_n (1 - a_4) \delta_{n1} z^n.
 \end{aligned}$$

Simplifying, we get:

$$\begin{aligned}
 \sum_n U_n z^n &= a_1 z^2 \sum_n U_{n-2} z^{n-2} + a_2 z^2 \sum_n V_{n-2} z^{n-2} + a_3 z \sum_n V_{n-1} z^{n-1} + a_4 \sum_n z^n \\
 &\quad - a_4 z^0 + (1 - a_4) z^1 \\
 \sum_n U_n z^n &= a_1 z^2 \sum_n U_n z^n + a_2 z^2 \sum_n V_n z^n + a_3 z \sum_n V_n z^n + a_4 \frac{1}{1 - z} \\
 &\quad - a_4 + (1 - a_4) z
 \end{aligned}$$

Since the generating functions $U(z)$ and $V(z)$ are defined as

$$U(z) = \sum_{n=0}^{\infty} U_n z^n = \sum_n U_n z^n, \quad V(z) = \sum_{n=0}^{\infty} V_n z^n = \sum_n V_n z^n,$$

we then replace the infinite sums with either $U(z)$ or $V(z)$ obtaining

$$U(z) = a_1 z^2 U(z) + a_2 z^2 V(z) + a_3 z V(z) + \frac{a_4}{1-z} - a_4 + (1 - a_4)z.$$

Using a similar process, we also can obtain

$$V(z) = b_1 z^2 U(z) + b_2 z^2 V(z) + b_3 z V(z) + \frac{b_4}{1-z} - b_4 + (1 - b_4)z.$$

We can now use Mathematica to solve for $U(z)$ and $V(z)$ in terms of p , q , and z :

$$U(z) = \frac{z(1 + (1 - q + pq)z + (-1 + p)(1 - p - q)z^2 - (-1 + p)(1 - p - q)z^3)}{(1 - z)^2(1 - (-1 + p)z)(1 - (1 - p - q)z)}$$

$$V(z) = \frac{z(1 - (1 - 3p + p^2 - q)z + p(1 - p - q)z^3)}{(1 - z)^2(1 - (-1 + p)z)(1 - (1 - p - q)z)}$$

From the standard properties of generating functions, we know that

$$W(z) = \frac{V(z)}{z} = \frac{(1 - (1 - 3p + p^2 - q)z + p(1 - p - q)z^3)}{(1 - z)^2(1 - (-1 + p)z)(1 - (1 - p - q)z)},$$

is a generating function for the sequence (V_1, V_2, V_3, \dots) . Let $W_{\uparrow}(z)$ and $W_{\downarrow}(z)$ represent the numerator and denominator, respectively, of $W(z)$. Then the general expansion theorem for rational generating functions [19] applied to $W(z)$ tells us:

$$W_n = V_{n+1} = (c_0 n + c_1)1^n + c_2(-1 + p)^n + c_3(1 - p - q)^n,$$

where c_i 's are real numbers, and $c_0 = (2W_{\uparrow}(1))/(W_{\downarrow}''(1))$. In the bit-rate calculation (equation 3.1) the bit rate is $\lim_{n \rightarrow \infty} \frac{1}{n} V_n$. Since all other terms besides the c_0 are lower order terms that approach 0 as n approaches ∞ , we can see that c_0 actually represents the bit-rate.

Using Mathematica once again, we find the bit-rate of this method is

$$R(\text{SC}(2), S_{p,q}) = c_0 = \frac{q + p(4 - q) - 2p^2}{(2 - p)(p + q)}. \quad (9.9)$$

Note that using $U(z)$ to calculate the bit-rate gives the same result. Section 9.4.9 presents plots of this bit-rate for various values of p at $q = 0.5$ and $q = 0.4$.

9.4.5 Two-state Sequential Coding

The more sensible alternative way of sequentially encoding $S_{p,q}$ is to choose the group iteration size k dependent upon the current state of the Markov chain for $S_{p,q}$. This method has two group iteration sizes k_0 and k_1 ; when the Markov chain is in state b , group iteration size k_b is used. This means that if the last bit processed was an b , then the next group iteration size used is k_b . We denote this method $SC(k_0, k_1)$. Note that when $k_0 = k_1$, we are choosing the same group iteration size no matter what state we are in; this is equivalent to the simple sequential coding method. Thus, $SC(k_0, k_0) = SC(k_0)$.

We can view the encoder of this method as a two-state algorithm: in state 0, it uses k_0 as the group iteration size; in state 1, it uses k_1 as the group iteration size. Intuitively, k_0 should be chosen to be about $k^*(\bar{p})$ since the probability of a zero occurring in state 0 is \bar{p} . Similarly, k_1 should be chosen to be about $k^*(q)$. This method can be thought of as a simplified version of the adaptive version of group testing where the group testing iteration size changes as more bits are encoded.

In our application to image compression, we are concerned with sources where $q \approx 0.5$ (see section 9.3.3); this models a clustering effect on the 1 bits. Since $k^*(0.5) = 1$, we study the case where $k_1 = 1$. This implies that after seeing a 1 bit, the sequential encoding method tests the next bits individually until it sees a 0 bit.

Since we use these sources to represent one bit-plane of one subband of the transform coefficients, this technique is like defining two classes for each subband, one for bits less likely to be significant, and one for bits more likely to be significant. In fact, this can represent defining GTW classes based on subband level and on a simple significant neighbor metric. Unlike in GTW, these bits are always coded in a sequential order. This means that when coding any bit, only up to one adjacent bit can be known to have value 1.

9.4.6 Bit-rate of Two-state Sequential Coding

Here we calculate the bit-rate for SC(2, 1). We assume the encoder starts out in state 0, meaning that it first chooses a group iteration size of 2. There are three possible input messages we could process in state 0: 00, 01, and 1?. As described above, if the last bit processed is a 1, then we transition into state 1; otherwise, we remain in state 0. In state 1, the group iteration size is one; we either process a 0 and transition into state 0 or process a 1 and remain in state 1.

We now calculate the bit-rate by using recurrence equations in a manner similar to section 9.4.4. Let U_n represent the expected number of bits to encode n bits from $S_{p,q}$, given that we are in state 0 (or equivalently, that the last source bit seen was a 0). Similarly, let V_n represent the expected number of bits to encode n bits from $S_{p,q}$, given that we are in state 1. The equations are as follows:

$U_0 = V_0$	$= 0$	Initial Conditions
$U_1 = V_1$	$= 1$	
U_n	$= \bar{p}\bar{p}(1 + U_{n-2})$	(Processing 00)
	$+ \bar{p}p(2 + V_{n-2})$	(Processing 01)
	$+ p(2 + V_{n-1})$	(Processing 1?)
V_n	$= q(1 + U_{n-1})$	(Processing 0)
	$+ \bar{q}(1 + V_{n-1})$	(Processing 1)

Note that with the exception of the equation for V_n , these equations above are the same as the equations for the simple sequential coding method.

Once again, we use the method of generating functions to solve this recurrence. Simplifying the equations, and putting them into generating function form yields:

$$\begin{aligned}
U_n &= a_1 U_{n-2} + a_2 V_{n-2} + a_3 V_{n-1} + a_4 \\
V_n &= b_1 U_{n-1} + b_2 V_{n-1} + b_3 \\
U(z) &= \sum_{i=0}^{\infty} U_i z^i = a_1 z^2 U(z) + a_2 z^2 V(z) + a_3 z V(z) + a_4/(1-z) - a_4 + (1-a_4)z \\
V(z) &= \sum_{i=0}^{\infty} V_i z^i = b_1 z U(z) + b_2 z V(z) + b_3/(1-z) - b_3 + (1-b_3)z,
\end{aligned}$$

where

$$\begin{aligned}
a_1 &= \bar{p}^2 \\
a_2 &= \bar{p}p \\
a_3 &= p \\
a_4 &= a_1 + 2a_2 + 2a_3 \\
b_1 &= q \\
b_2 &= \bar{q} \\
b_3 &= b_1 + b_2 = 1.
\end{aligned}$$

Mathematica can solve this, resulting in:

$$\begin{aligned}
U(z) &= \frac{z(1 - (1 - 3p + p^2 - q)z - p(1 + (-2 + p)q)z^2)}{(1 - z)^2(1 - (-1 + p)z)(1 - (1 - p - q)z)} \\
V(z) &= \frac{z(1 - (1 - p)^2 z^2 + qz(1 + 2pz - p^2 z))}{(1 - z)^2(1 - (-1 + p)z)(1 - (1 - p - q)z)}
\end{aligned}$$

We can then use the general expansion theorem for rational generating functions to derive that

$$U_n = (c_0 n + c_1)1^n + c_2(-1 - p)^n + c_3(1 - p - q)^n,$$

where c_i 's are real numbers, and c_0 represents the bit-rate.

Using Mathematica once again, we find the bit-rate of this method is

$$R(\text{SC}(2, 1), S_{p,q}) = c_0 = \frac{q + 2p(1 + q) - p^2(1 + q)}{(2 - p)(p + q)}.$$

Section 9.4.9 presents plots of this bit-rate for various values of p at $q = 0.5$ and $q = 0.4$.

9.4.7 Interleaved Coding

Instead of sequentially encoding the bits of $S_{p,q}$ with group testing, we could encode the bits in a different order to try to obtain a better bit-rate.

Let (b_0, b_1, \dots) represent the output bits from $S_{p,q}$. Let $B_0 = \{b_i : i \text{ is even}\}$, and $B_1 = \{b_i : i \text{ is odd}\}$. Then one possible encoding scheme is to first code bits from B_0 followed by bits from B_1 . The B_0 bits can be encoded with any sequential coding method, however, we will typically use SC(2, 1) because its bit-rate has already been calculated (see section 9.4.6). After coding bits from B_0 , we next encode the bits from B_1 using knowledge about B_0 . This will result in more efficient coding than coding B_1 without any previous knowledge. For all $b_i \in B_1$, we will know the values of both b_{i-1} and b_{i+1} when we are coding b_i . In contrast, the sequential methods only (potentially) know about b_{i-1} when trying to encode b_i . Knowing more information should enable us to code b_i more efficiently than in the sequential case. Of course, this is offset by the fact that we know less information about bits in B_0 when coding them than the sequential method knows.

After encoding the even-numbered bits B_0 , we can divide the odd-numbered bits B_1 into 4 different classes depending upon the values of its neighboring bits in B_0 ; we can label these classes 0X0, 1X0, 0X1, and 1X1, where the X is the unknown bit with the values of the two known bits surrounding it. Bits in the same class will have the same probability distribution because of the way probabilistic dependence is modeled in the Markov chain. The bits in the different classes can then be coded with a different group tester. The optimal group iteration size can be chosen for each class; this size depends upon the probability distribution for that class. We use $IC(k_0, k_1)$ to denote the interleaved coding method that uses SC(k_0, k_1) to encode the B_0 bits.

Practical implementations would not encode all of B_0 before starting on bits in B_1 . Once enough B_0 bits have been encoded so that there is a significant number of bits from B_1 in each class, then coding of the B_1 bits can begin. Coding of the B_0 and B_1 bits can then be done in parallel, with the B_1 bits encoded only after their neighboring bits in B_0

have been encoded.

This idea of interleaved coding is the most similar method to how GTW actually works with the standard definition of GTW classes that includes subband level, significant neighbor metric, and pattern type as characteristics. The initial classification of bits based on their location in the bitstream is similar to classifying bits based solely on their position in the subband, which is the idea of pattern types (section 6.1.2). Furthermore, we can think of interleaved coding as classifying the bits of B_1 into three classes, one for each of the possible number of significant neighbors (either 0, 1 or 2). This is similar to the classification by significant neighbor metric. The subband level characteristic is accounted for in that the different subbands are represented by different Markov sources.

9.4.8 Bit-rate of Interleaved Coding

The bit-rate for our interleaved coding scheme IC on $S_{p,q}$ can be calculated in two parts: the rate R_{B_0} for encoding the bits in B_0 , and the rate R_{B_1} for encoding the bits in B_1 . Since exactly half the bits are in B_0 and half the bits are in B_1 , the final bit-rate for interleaved coding is $\frac{1}{2}(R_{B_0}(\text{IC}, S_{p,q}) + R_{B_1}(\text{IC}, S_{p,q}))$.

To calculate R_{B_0} , we first need to calculate the statistical characteristics of the even-numbered bits. This can be done by looking at the bits output after taking two steps in the Markov chain for $S_{p,q}$. Let

$$\mathbf{M}_{p,q} = \begin{pmatrix} \bar{p} & p \\ q & \bar{q} \end{pmatrix}$$

represent the transition matrix of the Markov chain. Then

$$\mathbf{M}_{p,q}^2 = \begin{pmatrix} \bar{p}^2 + pq & \bar{p}p + p\bar{q} \\ \bar{p}q + \bar{q}q & pq + \bar{q}^2 \end{pmatrix}$$

represents the transition matrix of the new Markov chain that takes two steps for every one step of the original Markov chain. We can see that taking every other bit from source $S_{p,q}$ is equivalent to the source $S_{p',q'}$, where $p' = p(\bar{p} + \bar{q})$ and $q' = q(\bar{p} + \bar{q})$. Thus,

$R_{B_0}(\text{IC}, S_{p,q}) = R(\text{SC}, S_{p',q'})$, where SC is the sequential coding method that is used for the bits in B_0 .

To calculate R_{B_1} , we can split this term up into the bit-rates of each individual class. Let (C_1, C_2, C_3, C_4) represent the classes, $\Pr[C_i]$ represent the probability that a bit from B_1 is in class C_i , and $R(C_i, S_{p,q})$ represent the bit-rate of encoding class C_i . Then

$$R_{B_1}(\text{IC}, S_{p,q}) = \sum_{i=1}^4 \Pr[C_i] R(C_i, S_{p,q}).$$

By using the optimal group iteration size for each class C_i , the rate $R(C_i, S_{p,q})$ is simply $\text{GTR}(\Pr[b_i = 0 | b_i \in C_i])$. The bit-rate for the interleaved code method can thus be summarized as:

$$R(\text{IC}(k_0, k_1), S_{p,q}) = \frac{1}{2} \left(R(\text{SC}(k_0, k_1), S_{p',q'}) + \sum_{i=1}^4 \Pr[C_i] \text{GTR}(\Pr[b_i = 0 | b_i \in C_i]) \right) \quad (9.10)$$

We now need to calculate the probability of a zero bit occurring in each class, as well as the probability distribution of bits among the classes. As shorthand notation, we use $\Pr[XYZ]$ to refer to $\Pr[b_{i-1} = X, b_i = Y, b_{i+1} = Z]$. We first calculate the probabilities for sequences of length 3:

X	Y	Z	$\Pr[XYZ]$
0	0	0	$\pi_0 \bar{p} \bar{p}$
0	0	1	$\pi_0 \bar{p} p$
0	1	0	$\pi_0 p \bar{q}$
0	1	1	$\pi_0 p q$
1	0	0	$\pi_1 q \bar{p}$
1	0	1	$\pi_1 q p$
1	1	0	$\pi_1 \bar{q} \bar{q}$
1	1	1	$\pi_1 \bar{q} q$

Now for each of the four classes, we want to calculate the probability that a bit is in a particular class, as well as the probability that a bit is zero given that it belongs in a particular class. This is simply calculated as follows:

Class	Label	$\Pr[b_i \in \text{class}]$	$\Pr[b_i = 0 b_i \in \text{class}]$
C_1	0X0	$\Pr[000] + \Pr[010]$	$\frac{\Pr[000]}{\Pr[0X0]} = \frac{\bar{p}^2}{\bar{p}^2 + pq}$
C_2	0X1	$\Pr[001] + \Pr[011]$	$\frac{\Pr[001]}{\Pr[0X1]} = \frac{\bar{p}}{\bar{p} + q}$
C_3	1X0	$\Pr[100] + \Pr[110]$	$\frac{\Pr[100]}{\Pr[1X0]} = \frac{\bar{p}}{\bar{p} + q}$
C_4	1X1	$\Pr[101] + \Pr[111]$	$\frac{\Pr[101]}{\Pr[1X1]} = \frac{pq}{q^2 + pq}$

We can now compute the bit-rate for the interleaved coding using equation 9.10 along with the probabilities calculated above. Section 9.4.9 presents plots of $R(IC(2, 1), S_{p,q})$ and $R(IC(1, 1), S_{p,q})$ for various values of p at $q = 0.5$ and $q = 0.4$.

9.4.9 Comparison of Methods

In this subsection, we first compare the various bit-rates for the group testing methods on the source $S_{p,0.5}$. We then compare these results to the entropy of the source. Next, we repeat our result comparison on the source $S_{p,0.4}$ to show the effect of changing the amount of clustering of 1 bits in the source. We end by discussing the implications of these results towards encoding the dependence model as well as actual images.

Figure 9.1 is a plot of the bit-rate achieved for the different group testing methods on $S_{p,0.5}$. Overall, this plot shows that no one method is best for different probabilities. As can be seen, $IC(1, 1)$ is the best coding scheme when $p > 0.36$. Note that when $p > 0.41$, $IC(1, 1)$'s bit-rate of 1.0 is equal to that of $SC(1)$ because $IC(1, 1)$ decides to use group iterations of size 1 for both odd-numbered and even-numbered bits. At $.36 < p < 0.41$, $IC(1, 1)$ achieves a bit-rate better than $SC(1)$ because some of the odd-numbered bits are coded with group iteration size 2. For $.21 < p < 0.36$, $SC(2, 1)$ is the best coding scheme. When $p < 0.2$, methods with group iteration size of more than two would result in better bit-rates. Clearly as $p \rightarrow 0$, the best group iteration size should approach infinity. The graph does not show bit-rates for $p < 0.1$ because a group size of two is inappropriate for those parameters. Note that the simple sequential method $SC(2)$ is always significantly worse than the two-state sequential method $SC(2, 1)$ for this source.

Now we compare our group testing results to the entropy of the source, on $S_{p,q}$. Here our

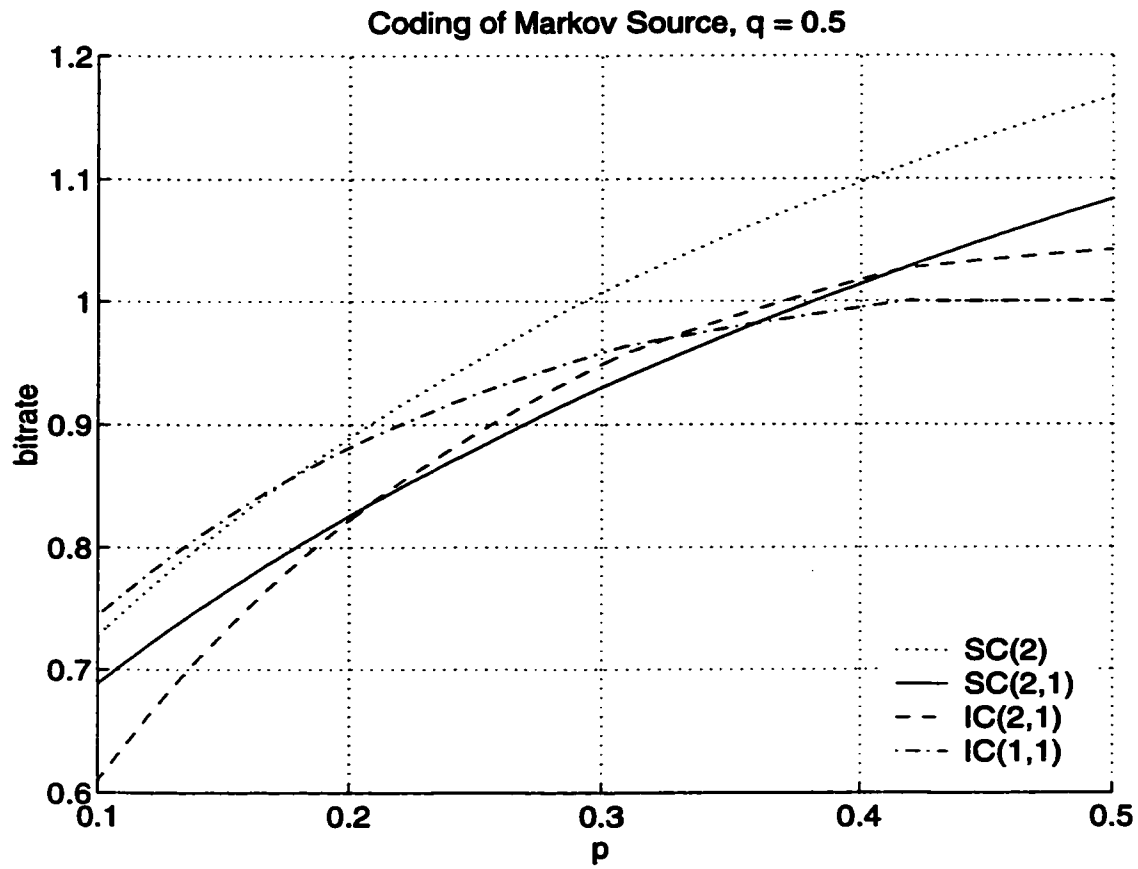


Figure 9.1: Comparing bit-rates of various group testing methods on $S_{p,0.5}$.

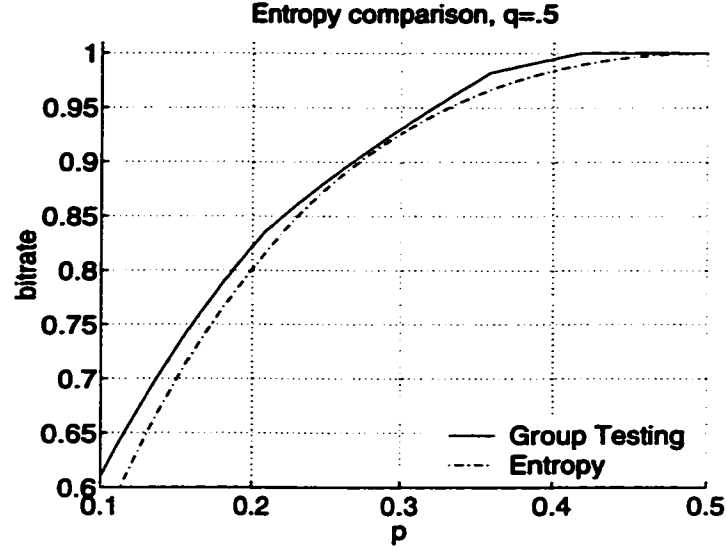


Figure 9.2: Comparing the bit-rates of the best group testing methods to the entropy on $S_{p,0.5}$.

group testing bit-rate $g(p, q)$ is the rate of the best group testing method presented in figure 9.1. Thus, over the set of coding methods $M = \{SC(2), SC(2, 1), IC(2, 1), IC(1, 1)\}$,

$$g(p, q) = \min_{m \in M} R(m, S_{p,q}).$$

Figure 9.2 shows the group testing bit-rate and the entropy of the source. Figure 9.3 shows the group testing bit-rate as a percentage of entropy, meaning $-1 + g(p, q)/H(S_{p,q})$.

Note that for the case when $p > 0.22$, the group testing method is within 2% of entropy. For values of $p < 0.22$, we know that the best sequential coding method and the best interleaved coding method would choose a group iteration size bigger than 2; thus the bit-rates of the methods shown here are not optimal. If we calculated group testing bit-rate for $SC(k_0, 1)$ for all values of k_0 , and picked the optimal k_0 to minimize the group testing rate for source $S_{p,q}$, we would expect to be much closer to entropy when $p < 0.22$.

We now show the bit-rate calculations for the case where $q = 0.4$. Note that in figure 9.4, the best method is always either $IC(1, 1)$ or $IC(2, 1)$. However, there appears to be a point at around $p = 0.29$ where the interleaved coding methods and $SC(2, 1)$ all have

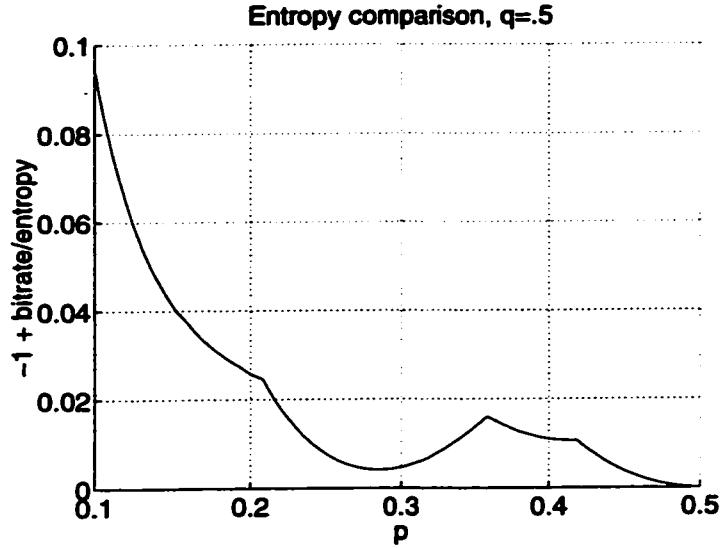


Figure 9.3: The group testing bit-rate as a percentage of entropy on $S_{p,0.5}$; this is a measure of the efficacy of group testing.

roughly the same bit-rate.

We can compare our new results with the previous results on i.i.d. sources. Putting the previous results in the context of Markov sources, figure 5.1 results for the best group testing algorithm on $S_{p,1-p}$ for values of p between 0.5 (on the left side of the graph) and 0 (on the right). It is interesting that our bit-rates on dependent sources are within 3% of entropy, which is lower than the peak of about 4.3% on the i.i.d. source. This shows that, in some cases, group testing on dependent sources actually performs better than on group testing on i.i.d. sources.

Conclusions

Our results show that using the appropriate group testing method on $S_{p,q}$, we can achieve a bit-rate that comes within 3% of entropy for the values of p and q that we examined. Relating these results back to the dependence model presented in section 9.3.2, we see that our results suggest it is possible to get within a small percentage of entropy (maybe 5%)

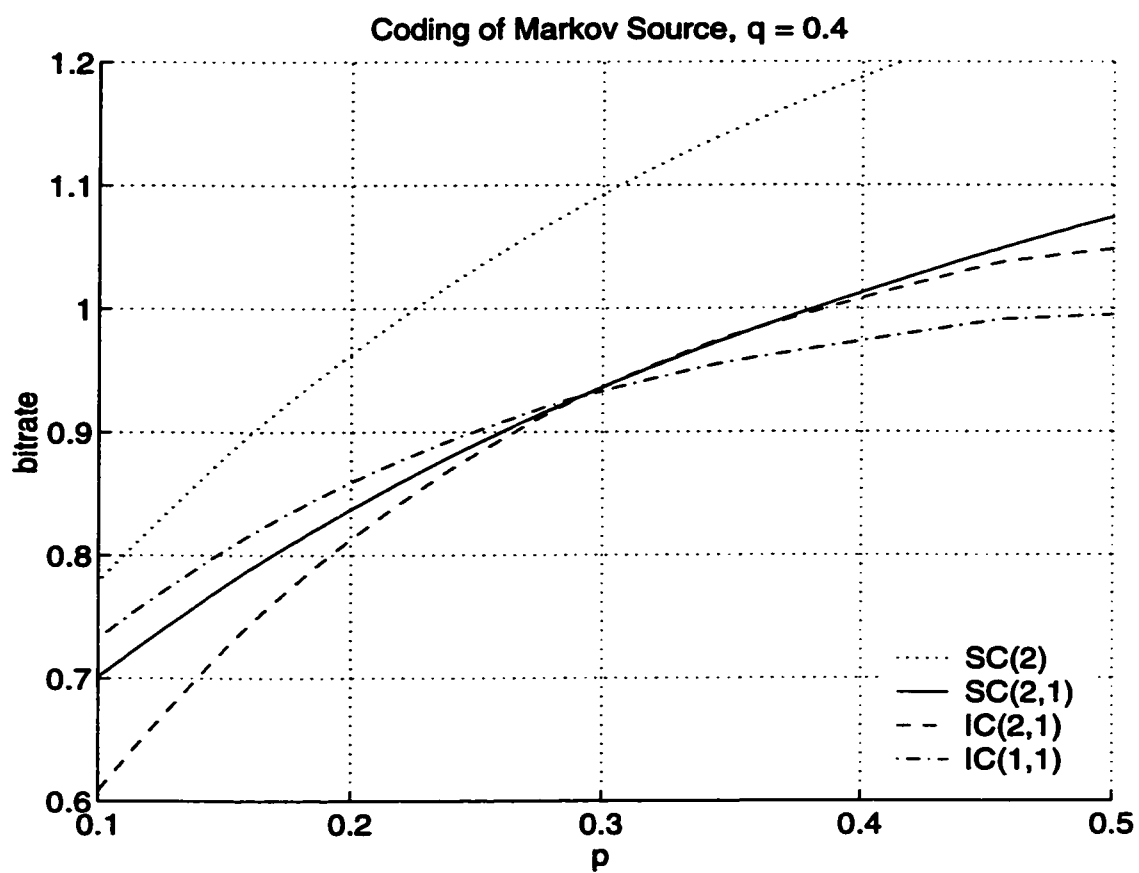


Figure 9.4: Comparing bit-rates of various group testing methods on $S_{p,0.4}$.

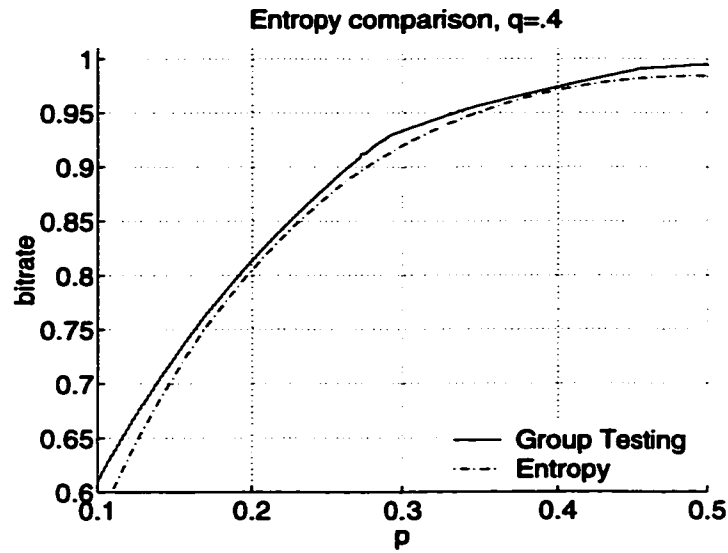


Figure 9.5: Comparing the bit-rates of the best group testing methods to the entropy on $S_{p,0.4}$.

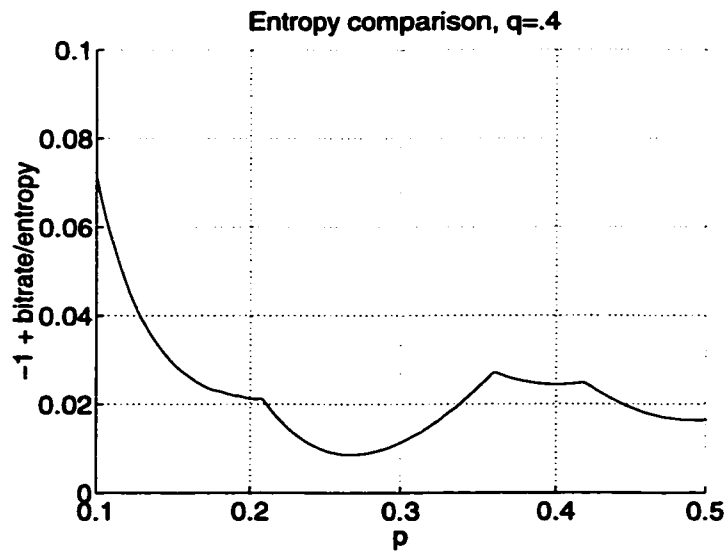


Figure 9.6: The group testing bit-rate as a percentage of entropy on $S_{p,0.4}$; this is a measure of the efficacy of group testing.

for using group testing on the dependence model.

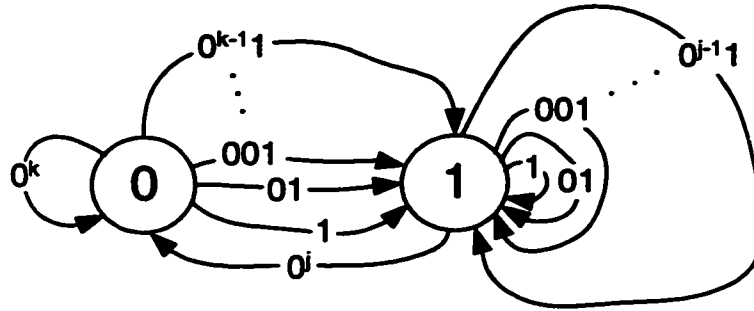
One interesting fact from studying the different group testing methods is that no particular method is consistently better than the other methods. In particular, for different values of p and q , sometimes interleaved coding works better than sequential coding, and sometimes the order is reversed. We know interleaved coding represents using pattern types in the definition of GTW classes, and sequential coding represents ignoring pattern type in the GTW classes.

This suggests the following: Suppose we use a parameter p to model the overall probability of significance for the coefficients in a two-dimensional subband, and a parameter q to model the amount of clustering of significant coefficients that occurs in the subband, according to some mathematical model. Then for some values of p and q , encoding the coefficients using pattern type would be better than encoding the coefficients without using pattern type. Furthermore, for different values of p and q , not using pattern type would be better. Whether or not the values of p and q for which pattern type is better actually occur in real subbands remains an open question. Regardless of this fact, our work also suggests that the difference between the bit-rates of using pattern type and not using pattern type are not very large.

9.4.10 *Alternative Method for Calculating Bit-rates*

In this section we present an alternative method for calculating $R(\text{SC}(k_0, k_1), S_{p,q})$ that is simpler than the previous method of using recurrence equations. Unfortunately, we currently have no proof that this method produces the correct result. However, this method produces the same results as the method of using recurrence equations on the two cases we examine; we believe this method also makes sense intuitively.

Figure 9.7 is a diagram of how the encoder $\text{SC}(k, j)$ processes the bits from $S_{p,q}$. State i of this diagram represents the fact that the last source bit was i , and that the source is currently in state i . From state i , there are transitions labeled by the possible input messages. The last bit of the input message determines the state to which the associated edge points.

Figure 9.7: Diagram of encoder for $SC(k, j)$.

Since a group iteration of size k is performed in state 0, there are $k + 1$ edges leaving state 0; the edges are labeled with the input messages $(1, 01, \dots, 0^{k-1}1, 0^k)$. Similarly, using a group iteration of size j in state 1 means that there are $j + 1$ edges leaving state 1.

Associated with each edge is a probability of following that edge when processing input from $S_{p,q}$. Table 9.1 shows these probabilities. Each edge also has a particular codeword that the encoder outputs after seeing the associated input message.

Table 9.1: Edge probabilities for the two-state Markov encoder.

starting state	label	probability
0	0^k	\bar{p}^k
0	$0^i 1, 0 \leq i \leq k - 1$	$\bar{p}^i p$
1	0^j	$q \bar{p}^{j-1}$
1	$0^i 1, 1 \leq i \leq j - 1$	$q \bar{p}^{i-1} p$
1	1	\bar{q}

We can view this diagram of the encoder as a Markov chain. Let $\pi' = (\pi'_0, \pi'_1)$ be the stationary distribution of this Markov chain. Furthermore, for $0 \leq i \leq k$, let b_i^0 , r_i^0 , and c_i^0 represent the input message, the probability, and the output codeword, respectively, of an edge starting in state 0. Similarly, for $0 \leq i \leq j$, let b_i^1 , r_i^1 , and c_i^1 represent the input

message, the probability, and the output codeword, respectively, of an edge starting in state 1. Then the bit-rate calculation is:

$$R(\text{SC}(k, j), S_{p,q}) = \frac{\pi'_0 \sum_{i=0}^k r_i^0 |c_i^0| + \pi'_1 \sum_{i=0}^j r_i^1 |c_i^1|}{\pi'_0 \sum_{i=0}^k r_i^0 |b_i^0| + \pi'_1 \sum_{i=0}^j r_i^1 |b_i^1|} \quad (9.11)$$

Note that this equation represents the bit-rate as the division of the expected codeword length by the expected input message length. The bit-rate for block codes on i.i.d. sources is also calculated by dividing the expected codeword length by the expected input message length (see equation 3.5, section 3.3.2). This similarity between the two bit-rates leads us to believe that equation 9.11 is correct. If we use this equation to calculate the bit-rate, we can obtain the rate for $\text{SC}(k, 1)$ as

$$R(\text{SC}(k, j), S_{p,q}) = \frac{p}{(1 - \bar{p}^k) \bar{p}(p + q)} \cdot \left(1 - \bar{p}^k - p + \bar{p}^k p + \bar{p}^{j'} q - \bar{p}^{j'+k} q + \bar{p}^{j+k'} q + (1 - \bar{p}^k)(1 - p - \bar{p}^j q) \lfloor \log_2 j \rfloor + (1 - \bar{p}^k) \bar{p}^j q \lfloor \log_2 k \rfloor \right),$$

where $k' = 2^{\lfloor \log_2 k \rfloor + 1} - k$ and $j' = 2^{\lfloor \log_2 j \rfloor + 1} - j$.

Note that the above formula agrees exactly with the previously calculated bit-rates for $\text{SC}(2, 2) = \text{SC}(2)$ and $\text{SC}(2, 1)$; this is another reason why we believe equation 9.11 is correct.

9.5 Conclusion

In this chapter, we sought to gain a better understanding of the effectiveness of group testing methods on images by defining models of the transform coefficient bits of images, and then by calculating the bit-rates of different group testing methods on the source models.

Our independence model is basically a collection of binary i.i.d. sources. We showed that the group testing bit-rate using group iterations of the appropriate size is always within 5% of the entropy of the independence model.

Our dependence model is basically a collection of two-state Markov sources that contain probabilistic dependencies between its bits. These sources serve as a model for the significance bits of a bit-plane of one subband. We defined three different methods of encoding the two-state Markov source. For this source, we showed that when $0.2 \leq p < 0.5$ and when $q = 0.4$ or $q = 0.5$, the bit-rate of the best group testing method we studied is within 3% of the entropy of the source. This suggests it is possible to compress the dependence model of an image using group testing to within a small percentage of the entropy of the model.

Chapter 10

CONCLUSIONS

This thesis discusses the application of group testing to the field of image compression. We have shown how group testing is related to Golomb codes and zerotree coding, two image compression techniques that have been widely used. We have defined new image compression algorithms that are primarily used group iterations, the group testing method equivalent to elementary Golomb codes, to compress data. Furthermore, we have demonstrated that our algorithms improve upon previous ones and are often competitive with the best image compression algorithms in terms of rate-distortion performance.

Although our goal has been to find the best method of group testing for image compression, some of our work can be thought of in more general terms. Our analysis of the performance of various group testing techniques on different source models can be used to answer the question, “If I wanted to use group testing to compress this source, what group testing method should be used?” It does not matter if the source relates to images. Much of the work in compressing the transform coefficients of an image was in finding the correct definition for the classes. Thus, our work can be viewed as finding the statistical characteristics of the transform coefficients, and exploiting them for efficient compression. Although we applied the group testing technique to compress these coefficients, this is not strictly necessary. Once the statistical characteristics are known, any compression technique that can take advantage of these statistics can be used, not necessarily group testing.

We end this thesis by discussing some of the possible directions for future work. There are other compression problems for which we could study the effectiveness of our group testing method. One example is coding images consisting of two colors (binary bit-maps). In addition, it would be worthwhile to explore other group testing methods that do not use

group iterations as their main subroutine. And completely new group testing algorithms could be studied for the group testing problem that included queries for learning about the source.

Much more work could be done on calculating the theoretical performance of group testing. Besides verifying that the alternative method of calculating the bit rate (section 9.4.10) is correct, we could explore the efficiency of other group testing methods. It would be interesting to look at other methods of interleaved coding, such as coding every n th bit of the source, before proceeding on to the rest of the bits. It would also be interesting to consider other sources that may represent images better, such as Markov random fields. With all the sources that we define, it would be interesting to explore statistically how well the image models match up with the actual characteristics of real images.

BIBLIOGRAPHY

- [1] S. O. Aase and T. A. Ramstad. On the optimality of nonunitary filter banks in subband coders. *IEEE Transactions on Image Processing*, 4(12):1585–91, Dec. 1995.
- [2] M. Adams. JasPer: A software based JPEG-2000 codec implementation. <http://www.ece.ubc.ca/~mdadams/jasper>.
- [3] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1:205–220, April 1992.
- [4] A. Bar-Noy, F. Hwang, H. Kessler, and S. Kutten. A new competitive algorithm for group testing. *Discrete Applied Mathematics*, 52:29–38, July 1994.
- [5] C. Brislawn. FBI/LANL wavelet/scalar quantization (WSQ) fingerprint image compression specification. <ftp://wwwc3.lanl.gov/pub/misc/WSQ>.
- [6] J. Capon. A probabilistic model for run-length coding of pictures. *IRE Transactions on Information Theory*, pages 157–163, 1959.
- [7] B. Chai, J. Vass, and X. Zhuang. Significance-linked connected component analysis for wavelet image coding. *IEEE Transactions on Image Processing*, 8(6):774–784, June 1999.
- [8] H. D. Cho and J. B. Ra. A rearrangement algorithm of wavelet packet coefficients for zerotree coding. In *Proceedings 1999 International Conference on Image Processing*, volume 3, pages 556–9.

- [9] R. R. Coifman and M. V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory*, 38(2):713–8, March 1992.
- [10] I. Daubechies. *Ten lectures on wavelets*. SIAM, 1992.
- [11] G. Davis, J. Danskin, and R. Heasman. Baseline wavelet transform coder construction kit. <ftp://ftp.cs.dartmouth.edu/pub/gdavis/wavelet.0.3.tar>, January 1997. Version 0.3.
- [12] A. Deever and S. S. Hemamai. What's your sign?: Efficient sign coding for embedded wavelet image coding. In *Proceedings DCC 2000, Data Compression Conference*, pages 273–282.
- [13] R. Dorfman. The detection of defective members of large populations. *Annals of Mathematical Statistics*, 14:436–440, 1943.
- [14] D. Du and F. Hwang. *Combinatorial Group Testing*. World Scientific, 1993.
- [15] D. Z. Du, G. L. Xue, S. Z. Sun, and S. W. Cheng. Modifications of competitive group testing. *Siam Journal of Computing*, 23(1):82–96, February 1994.
- [16] F. Fabris. Variable-length-to-variable-length source coding: A greedy step-by-step algorithm. *IEEE Transactions on Information Theory*, 38(5):1609–1617, September 1992.
- [17] R. Gallager and D. Van Voorhis. Optimal source codes for geometrically distributed integer alphabets. *IEEE Transactions on Information Theory*, 21:228–230, March 1975.
- [18] S. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 12:399–401, July 1966.

- [19] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 1989.
- [20] E. S. Hong and R. E. Ladner. Group testing for image compression. In *Proceedings DCC 2000, Data Compression Conference*, pages 3–12.
- [21] E. S. Hong, R. E. Ladner, and E. A. Riskin. Group testing for wavelet packet image compression. In *Proceedings DCC 2001, Data Compression Conference*, pages 73–82.
- [22] E. S. Hong, R. E. Ladner, and E. A. Riskin. Group testing for block transform image compression. In *Thirty-Fourth Asilomar Conference on Signals, Systems and Computers*, 2001. To appear.
- [23] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proceedings IRE*, volume 40, pages 1098–1101, 1962.
- [24] F. Hwang. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association*, 67:605–8, 1972.
- [25] F. K. Hwang. A generalized binomial group testing problem. *Journal of the American Statistical Association*, 70(352):923–926, December 1975.
- [26] N. S. Jayant and P. Noll. *Digital Coding of Waveforms*. Prentice Hall, 1984.
- [27] H. Khalil, A. Jacquin, and C. Podilchuk. Constrained wavelet packets for tree-structured video coding algorithms. In *Proceedings DCC'99, Data Compression Conference*, pages 354–63, 1999.
- [28] UCLA Image Communications Lab. images. http://www.icsl.ucla.edu/~ipl/psnr_images.html.

- [29] G. G. Langdon, Jr. An adaptive run-length coding algorithm. *IBM Technical Disclosure Bulletin*, 26(7B):3783–3785, December 1983.
- [30] S. LoPresto, K. Ramchandran, and M. Orchard. Image coding based on mixture modeling of wavelet coefficients and a fast estimation-quantization framework. In *Proceedings Data Compression Conference, DCC'97*, pages 221–230, March 1997.
- [31] G. S. Maddala. *Introduction to Econometrics*. Macmillan Publishing Company, second edition, 1992.
- [32] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.
- [33] H. Malvar. Fast progressive coding without wavelets. In *Proceedings DCC 2000, Data Compression Conference*, pages 243–252, March.
- [34] H. Malvar. Fast progressive wavelet coding. In *Proceedings DCC'99, Data Compression Conference*, pages 336–343, March 1999.
- [35] H. S. Malvar. Lapped biorthogonal transforms for transform coding with reduced blocking and ringing artifacts. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 2421–4.
- [36] H. S. Malvar. *Signal Processing with Lapped Transforms*. Artech House, 1992.
- [37] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek. An overview of JPEG-2000. In *Proceedings DCC 2000, Data Compression Conference*, pages 523–541.
- [38] F. G. Meyer, A. Averbuch, and J. O. Stromberg. Fast adaptive wavelet packet image compression. *IEEE Transactions on Image Processing*, 9(5):792–800, May 2000.

- [39] R. Oktem, L. Oktem, and K. Egiazarian. A wavelet packet transform based image compression algorithm. In *Proceedings NORSIG 2000*, pages 77–80, June.
- [40] E. Ordentlich, M. Weinberger, and G. Seroussi. A low-complexity modeling approach for embedded coding of wavelet coefficients. In *Proceedings DCC'98, Data Compression Conference*, pages 408–417, March 1998.
- [41] M. Rabbani and P. W. Jones. *Digital Image Compression Techniques*. SPIE Optical Engineering Press, 1991.
- [42] N. M. Rajpoot, F. G. Meyer, R. G. Wilson, and R. R. Coifman. On zerotree quantization for embedded wavelet packet image coding. In *Proceedings 1999 International Conference on Image Processing*, volume 2, pages 283–7.
- [43] K. Ramchandran and M. Vetterli. Best wavelet packet bases in a rate-distortion sense. *IEEE Transactions on Image Processing*, 2(2):160–75, April 1993.
- [44] R. F. Rice. Some practical universal noiseless coding techniques. Technical Report JPL-79-22, Jet Propulsion Laboratory, 1979.
- [45] A. Said and W. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6:243–250, June 1996.
- [46] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann Publishers, 1996.
- [47] J. W. Schwartz and R. C. Baker. Bit-plane encoding: a technique for source encoding. *IEEE Trans. Aerospace Electron. Syst.*, 2:385–392, July 1966.
- [48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

- [49] J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41:3445–3462, December 1993.
- [50] USC Signal and Image Processing Institute. image database. <http://sipi.usc.edu/services/database/Database.html>.
- [51] M. Sobel and P. A. Groll. Group testing to eliminate efficiently all defectives in a binomial sample. *Bell System Technical Journal*, 38:1179–1252, 1959.
- [52] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Transactions on Image Processing*, 9(7):1158–70, July 2000.
- [53] T. D. Tran, R. L. de Queiroz, and T. Q. Nguyen. Linear-phase perfect reconstruction filter bank: Lattice structure, design, and application in image coding. *IEEE Transactions on Signal Processing*, 48(1):133–147, January 2000.
- [54] T. D. Tran and T. Q. Nguyen. A progressive transmission image coder using linear phase uniform filterbanks as block transforms. *IEEE Transactions on Image Processing*, 8(11):1493–1507, November 1999.
- [55] B. P. Tunstall. *Synthesis of Noiseless Compression Codes*. PhD thesis, Georgia Institute of Technology, September 1967.
- [56] G. Uytterhoeven, D. Roose, and A. Bultheel. Wavelet transforms using lifting scheme. Technical Report ITA-Wavelets-WP1.1, Katholieke Universiteit Leuven, Department of Computer Science, April 1997.
- [57] G. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, April 1991.

- [58] M. V. Wickerhauser. Acoustic signal compression with wavelet packets. In C. K. Chui, editor, *Wavelets: A Tutorial in Theory and Applications*, pages 679–700. Academic, 1992.
- [59] I. Witten, R. Neal, and J. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30:520–540, June 1987.
- [60] X. Wu. High-order context modeling and embedded conditional entropy coding of wavelet coefficients for image compression. In *Thirty-First Asilomar Conference on Signals, Systems and Computers*, volume 23, pages 1378–82, 1998.
- [61] Z. Xiong, O. G. Guleryuz, and M. T. Orchard. A DCT-based embedded image coder. *IEEE Signal Processing Letters*, 3:289–290, November 1996.
- [62] Z. Xiong, K. Ramchandran, and M. T. Orchard. Wavelet packets-based image coding using joint space-frequency quantization. In *Proceedings 1994 International Conference on Image Processing*, volume 3, pages 324–8.
- [63] Z. Xiong, K. Ramchandran, and M. T. Orchard. Space-frequency quantization for wavelet image coding. *IEEE Transactions on Image Processing*, 6(5):677–693, May 1997.
- [64] Z. Xiong, K. Ramchandran, and M. T. Orchard. Wavelet packet image coding using space-frequency quantization. *IEEE Transactions on Image Processing*, 7(6):892–8, June 1998.

VITA

Edwin Hong was born and raised in Seattle, Washington. He earned a B. S. in Computer Science and Mathematics at Yale University in 1995, and then proceeded to the University of Washington, where he earned his M. S. in computer science in 1998. He also earned his Doctorate in computer science at the University of Washington in 2001. His research interests include data compression, where where he has many publications about image compression, and online algorithms.