1994

Some new developments in image compression

Keshi Chen
Iowa State University

Order Number 9518365

Some new developments in image compression

Chen, Keshi, Ph.D.

Iowa State University, 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Some new developments in image compression

by

Keshi Chen

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Department: Electrical Engineering and Computer Engineering
Major: Electrical Engineering (Communications and Signal Processing)

Approved:

Signature was redacted for privacy.

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1994

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DCT	Discrete cosine transform
DPCM	Differential pulse code modulation
dB	Decibal
ECDPCM	Proposed entropy-coded differential pulse code modulation scheme
ECGTSVQ	Entropy-constrained greedily-grown tree-structured vector quantization
ECSQ	Entropy-constrained scalar quantization
ECVQ	Entropy-constrained vector quantization
GLA	Generalized Lloyd algorithm
GSVQ	Gain-shape vector quantization
GTSVQ	Greedily-grown tree-structured vector quantization
JPEG	Joint Photographics Experts Group
KLT	Karhunen-Loeve transform
MPEG	Moving Picture Experts Group
MR	Magnetic resonance
MRVQ	Mean-removed vector quantization
PAE	Peak absolute error
PDF	Probability density function

PSNR	Peak signal-to-noise ratio
PTSVQ	Pruned tree-structured vector quantization
SNR	Signal-to-noise ratio
SQNR	Signal-to-quantization-noise ratio
SQ	Scalar quantization
TSVQ	Tree-structured vector quantization
US	Ultrasound
VLSI	Very large scale integration
VQ	Vector quantization
WHT	Walsh-Hadamard transform
WT	Wavelet transform

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my major professor Dr. Tenkasi V. Ramabadran for his constant help, support, and invaluable advice throughout the course of this study.

My appreciation is extended to Dr. John P. Basart, Dr. Glenn R. Luecke, Dr. Satish S. Udpa, Dr. Vijay Vittal, and Dr. Robert J. Weber who served on my committee and supported my research in various ways including the privilege of using various experimental facilities. Without this support, it would have been very difficult for me to pursue my study and research.

I am also indebted to Dr. C. B. Chittineni of Imaging Systems Department, DuPont Company, for providing us with several digitized images used in the experiments.

This thesis is dedicated to my wife Huiping and my parents who have sacrificed a lot to support my education.

1. INTRODUCTION

1.1 Introduction to Image Compression

Until recently, visual information has been essentially manipulated in analog fashion. With the evolution of digital techniques for information processing, transmission and storage, the need has arisen for digital representation of visual information, *i.e.*, the representation of images by a sequence of integer numbers. In this form, computer processing and digital circuit techniques can be utilized for manipulation of visual information which were undreamed of only a short time ago.

Conversion of analog images into digital form offers several advantages. Digital signals can be more reliably transmitted over noisy channels than analog signals thanks to effective detection techniques, such as matched filters and correlation receivers, and error control coding, such as parity check codes. As a result, they provide the possibility of making better use of interference-limited and noise-limited communication media [53]. Also, digitizing signals permits signal regeneration in digital repeaters, thereby allowing long-haul communication with negligible line loss. Furthermore, because of the flexibility in the scope and nature of digital operations, machine manipulation and interpretation of visual information becomes possible. For example, image acquisition, transmission, processing, storage, and display operations can all be carried out and controlled by computers via a network. These advantages

promote ingenuity and further innovation in information services, which, in turn, generate even greater demand for effective image processing, transmission, and storage techniques.

Doubtless, advances in such hardware technologies as VLSI fabrication, fiber optical communication, and high density data storage equipment have been responsible in improving the efficiency and capability of digital image manipulations. Interestingly, these advances do not appear to have discouraged investigation of new manipulating methods. On the contrary, they have made some of the more sophisticated algorithms tractable.

Image transmission and storage are two major issues in digital image manipulation. Image transmission applications occur in broadcast/cable television, video telecommunication, remote sensing via satellite, facsimile transmission, and the like. Image storage is required for educational and business documents, medical images, motion pictures, *etc.* As far as these two issues are concerned, there arises a serious problem, *viz.*, the amount of data required to represent digital images is very large, so large that its transmission and/or storage would require enormous channel and/or storage capacity. For example, typical data rate for reasonable spatial and temporal quality of a moving image in such applications as broadcast television is 100 megabits per second, about 1600 times more than that for a speech signal of good intelligibility [23]. These large channel and storage capacity requirements, naturally, make it desirable to consider image compression techniques.

Image compression (or coding) is concerned with reducing the amount of data required to represent one or a series of images. Besides its primary application in image transmission and storage, image compression is also applicable in the devel-

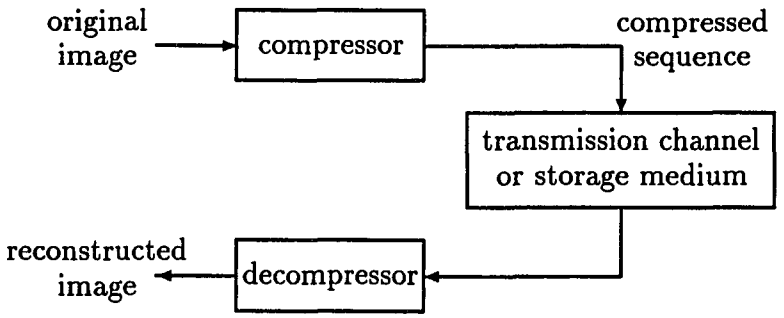


Figure 1.1: Block diagram of image transmission or storage process

opment of fast algorithms where the number of operations required to implement a processing algorithm is reduced by working with the compressed data [52, 26]. The block diagram of a typical image transmission or storage process accompanied with a compression-decompression operation is depicted in Figure 1.1. Normally, it is unnecessary to make a distinction between the transmission and the storage applications in order to formulate efficient compression algorithms. The distinction comes about when a particular compression technique is to be implemented. For example, transmission applications might be more sensitive to the number of operations it takes to compress and decompress an image while storage applications might be more concerned with the compression efficiency.

1.2 Overview of Image Compression Techniques

In general, the compression of a digital signal such as a sequence of digitized image pixels is realized by performing one or both of two operations: *quantization* and *variable-length coding*. In quantization, samples (or blocks of contiguous samples) of the signal being compressed are mapped in a many-to-one fashion into

symbols taken from a finite set of size smaller than that of the original samples (or blocks of samples). These symbols are then transmitted/stored, usually in their binary representations, or undergo the variable-length coding operation before transmission/storage. In variable-length coding, the sequence of symbols resulting from quantization, or the original signal being compressed if no quantization has been performed, is transformed into a sequence of codewords typically by using longer codewords to represent rare symbols and shorter codewords for those occurring more frequently. Clearly, some information will be lost through quantization due to the many-to-one mapping, while variable-length coding is typically lossless.

Many techniques have evolved over the years for image compression. Some have the virtue of easy implementation, while others have special features which make them well suited for specific applications. In whatever manner they differ from each other, all of the compression methods rely on the common basic statistical property of digital images that there is a high degree of interpixel dependencies, especially interpixel correlation, among image data. In other words, most of the raw image data are redundant; and this redundancy makes compression possible. Therefore, the degree to which images may be compressed while still allowing satisfactory reconstruction after storage or transmission in compressed form is crucially dependent upon their dependency properties. Also, the performance of a specific compression algorithm for a given image is mainly dependent upon how much the algorithm can capture and utilize the interpixel dependency characteristics of the image. In the following, a brief description of different types of image compression techniques and discussions on some related issues are presented.

1.2.1 Interframe versus intraframe compression

Image compression techniques can roughly be classified into two categories: *interframe* and *intraframe* compression. The former, with application in motion picture transmission, aims at optimizing the efficiency of representation of several related images, *e.g.*, consecutive images of a time series, by taking into account the correlations that exist in the temporal domain whereas the latter treats images mainly on an individual basis.

1.2.2 Lossy versus lossless compression

Based on the nature of the reconstructed images, image compression methods can also be divided into two other classes. In the first class, referred to as *lossy* (or *noisy, irreversible*) compression, are methods which compress images by retaining only the "important" information components and disregarding the "less important" ones [23]. Consequently, the reconstructed image is not an exact copy but only a distorted version of the original. Clearly, the methods in this class make use of quantization in one way or another (and possibly variable-length coding as well), and the compression efficiency of such methods is determined by the techniques used for identification and selection of "important" components as well as the desired degree of fidelity. A great amount of research effort has been spent on this class of methods for decades [80, 50, 86], searching for and improving the techniques with which the original image is transformed so that the "important" and "less important" components are signified and separated, or suitable rules by which appropriate amount of data are allocated to represent the components of different "importance". In contrast, another class of image compression methods, referred to as *lossless* (or *noiseless, reversible*) com-

pression, has received relatively less attention mainly because of the lower achievable compression performance. With these methods, the original images can be exactly reconstructed from the compressed data, that is, no distortion is introduced at all. All lossless image compression methods [98, 19, 88] utilize variable-length coding, and sometimes quantization as a front stage of compression.

Most image transmission and storage applications can benefit from both lossy and lossless image compression. The choice as to which class of techniques to use is primarily application dependent. In general, lossy compression methods can provide better and predictable compression, and therefore are more suitable for applications where some loss in image fidelity can be tolerated in order to utilize a transmission channel or storage medium more efficiently. On the other hand, lossless methods are preferred in situations where even a little amount of distortion cannot be tolerated and/or post-processing of images is involved. In this case, the information distortion introduced by lossy compression may lead to critical interpretation errors, thereby justifying the use of lossless techniques for such images. Moreover, lossless methods have the advantage that they can be easily evaluated based on their compression performances, whereas there is no unique and adequate measure to compare lossy methods.

1.2.3 Scalar versus vector compression

The compression of an image can be performed in a scalar fashion or a vector fashion. In the former, compression operations are applied to each pixel on an individual basis and compression is achieved by taking into account the interpixel dependencies between the pixel and its close neighbors; whereas in the latter, the

image to be compressed is first divided into blocks of typical size 4×4 or 8×8 and then each block is treated as a single entity by the compression operations. Usually, compression in vector fashion utilizes quantization.

Vector quantization (VQ) has emerged as a promising approach to signal compression and attracted extensive research efforts since the early 1980's. It is inherently superior to scalar compression methods owing to its better capability and greater flexibility of exploiting extensive inter-sample dependencies present in a signal. While VQ holds many advantages as a technique for signal compression, its primary limitation is the high implementation complexity, especially when high-fidelity reconstruction is desired. Therefore, vector quantization is more often found in low-rate applications while scalar compression methods are more preferable for applications that require high reconstruction quality.

1.2.4 Compression for progressive transmission

While lossless compression is always preferable for some categories of images, such as medical images, it may require unreasonable transmission and storage capabilities compared to lossy compression. Progressive transmission systems provide a compromise [109, 113]. In progressive transmission applications, an image is compressed in stages such that the first stage provides a low-fidelity reconstruction, and the succeeding stages in combination with the earlier stages provide higher-fidelity reconstructions. The final stage in combination with all previous stages may provide overall lossless compression. In this manner, an image transmitted over a low capacity link can be viewed at the receiving end as soon as part of the compressed sequence arrives rather than waiting until the entire sequence is completely received, and a de-

cision can be made during the course of transmission whether the reconstruction at the receiving end is good enough and therefore further transmission could be aborted, which in turn allows the communication resources to be utilized more efficiently.

Clearly, lossy compression and possibly lossless compression as well are involved in a progressive image transmission application, and methods [109, 61, 16, 104, 81, 118, 113] for such purposes all employ one kind or another of hierarchical data structure. The objective of lossy image compression for progressive transmission is to provide the best possible reconstruction with a limited amount of data and the best possible improvement in reconstruction with any additional amount of data, while the lossless compression should be able to utilize the reconstruction results obtained from the lossy compression as well as to minimize the length of the final compression sequence.

1.2.5 Image compression standards

After extensive research efforts, a few general-purpose image compression standards based on state-of-the-art techniques have emerged to facilitate interoperability of communication and storage equipments from different manufacturers. Among these standards, the *Joint Photographics Experts Group* (JPEG) standard [114] for still image compression, *i.e.*, intraframe compression, and the *Moving Picture Experts Group* (MPEG) standard [58] for motion picture compression, *i.e.*, interframe compression, have been widely adopted in industries. The baseline compression method in the JPEG standard is based on the *discrete cosine transform* (DCT) [23] whereby 8×8 blocks of an image are first transformed using DCT and then the resulting DCT coefficients are uniformly quantized and entropy coded to form the compressed data

sequence. Using this standard method, reconstructed images with indistinguishable to excellent quality can be obtained for compression factors roughly in the range between 4 and 12 [114]. In some experiments conducted during the course of this study, the JPEG standard has been used to assess the compression performances of some proposed algorithms.

1.3 Focus of This Study

The study reported in this dissertation is mainly concerned with the intraframe compression of digitized images and is divided into two parts. The first part involves the investigation of near-lossless, *i.e.*, lossy but high-fidelity, compression of digitized images in a scalar fashion and is an extension of the work done by the author previously [19] where the problem of lossless compression is considered; the second part is devoted to the problem of lossy image compression for progressive transmission using vector quantization.

1.3.1 First part of the study

In this investigation, the use of entropy-coded *differential pulse code modulation* (DPCM) method for image compression is considered. DPCM is a well-known and easy-to-implement method commonly found in speech coding and image compression applications [67, 5, 76]. It uses linear prediction to decorrelate the signal to be compressed and then quantizes and encodes the decorrelated signal to form the compressed sequence. Typically, fixed-length coding of the quantized output is used because of its simplicity. The incorporation of *entropy coding*, a special type of variable-length coding, can clearly lead to improved compression performance. Since the early

1950's, the DPCM method for signal compression has been researched extensively and the design of linear predictors [51] and quantizers [119, 111, 79, 108, 12, 84, 54] for the method has been well developed. In the investigation reported here, such known results are used and a new scheme that combines both lossy and lossless DPCM methods into a common framework is developed. The new scheme is different from previously known schemes in that it uses a *context-based source model* in the coding of the quantized outputs. Context-based source modeling represents one of the main advances made in recent years in the area of data compression. It helps to reduce the uncertainty of a given data sequence through the use of contexts (or conditioning events) and thereby helps to improve compression performance. Similar to the lossless DPCM method that has been considered before [19, 88], it is shown that the new scheme can significantly enhance its compression performance through the use of context-based source modeling techniques.

1.3.2 Second part of the study

This part of the study is mainly focused on the use of vector quantization for lossy image compression. As mentioned earlier, vector quantization represents a major class of signal compression techniques that offer many advantages. However, the conventional VQ algorithms that lead to the implementations of *fixed-rate* compression systems are not optimal in terms of the relationship between compression performance and reconstruction quality [21]. To achieve the optimal performance, in general, variable-rate compression is required. Through the investigation in this part of the study, a new method that uses an unbalanced tree data structure is developed together with four different algorithms for designing variable-rate VQ compression

systems. From experimental results, it is found that the method can achieve very good compression performance and reconstruction quality while being computationally efficient compared to conventional VQ algorithms. Also, due to the use of the tree data structure, the new method is found to be amenable to progressive transmission applications.

It should be pointed out that although the two parts of the study reported here appear to be very different, one dealing with scalar coding and the other using vector quantization, they are closely related to each other, as will be seen from later discussions. In fact, the starting point of investigation in the second part of the study originated from the work done in the first part.

1.3.3 Major contributions

The major contributions of this study are summarized in the following.

1. A context-based source model is developed that can be used for both lossy and lossless compression of digitized images and can lead to significant performance improvement.

2. A new variable-rate vector quantization technique for image compression together with several implementation algorithms is developed. This technique can achieve high rate-distortion performance while being computationally efficient. Also, it is suitable for progressive transmission applications.

3. A novel approach of statistical source modeling is proposed. This approach employs a new technique of selecting contexts for the source model of a data

sequence to be compressed, and provides a mechanism to the design of compression systems for a wide range of code rates.

1.3.4 Organization of this dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, the elements of signal compression using variable-length coding are briefly described. The background theory needed for understanding the operations of the entropy-coded DPCM method and context-based source modeling are also provided. Important implementation details of the new entropy-coded DPCM scheme that uses context-based source modeling are then presented in Chapter 3, along with the experimental results indicative of the compression performance and reconstructed image quality of the scheme. From Chapter 4, we turn to the second part of the study by first introducing the concepts of vector quantization and the general techniques for designing VQ compression systems. Then, in Chapter 5, some fundamental results from rate-distortion theory that deals with the optimal relationship between compression performance and reconstruction quality are discussed. In the same chapter, an important variable-rate VQ technique, called *entropy-constrained vector quantization* [21], that was developed by closely following the guidelines of rate-distortion theory is described. Detailed descriptions of the new variable-rate tree-structured VQ method and the four pertaining algorithms are provided in Chapter 6. Experimental results obtained from implementations of the algorithms as well as some relevant discussions are also presented in this chapter. Finally, in Chapter 7, a summary of the study and some recommendations for future research are given.

2. SOURCE MODELING AND CODING

2.1 Introduction

As mentioned earlier, the compression of a data sequence through variable-length coding (with or without quantization) is usually realized by the assignments of longer codewords to data blocks with lower probabilities and shorter codewords to the more probable blocks. Clearly, before using any optimal or nearly optimal coding technique to perform such a coding operation, one needs to model the data blocks as possible outcomes of some information source so as to be able to estimate their probabilities of occurrence. Therefore, the compression of a data sequence can be in general regarded as consisting of two distinct parts: *source modeling* and *coding* [117]. Source modeling deals with how to view the data sequence based on some *a priori* knowledge and subsequently how to capture the intersymbol dependencies presented in the data; whereas coding transforms the original sequence into a sequence of codewords called the code sequence. As a result, the compressed sequence is composed of two parts: 1) a set of model parameters that identifies the source model associated with the original sequence, and 2) the code sequence that identifies the original sequence among all the data sequences having the same source model. The objective of compression is then to minimize the length of this compressed sequence.

A source model for a data sequence, in general, determines how a specific coding

technique is to be applied to the sequence and also supplies parameters to the coding process. Therefore, source modeling has played a central role in the modern data compression paradigm [117] and attracted increasing research efforts. Through these efforts, advances have been made in building sophisticated source models that can describe a sequence to be compressed more accurately, *i.e.*, predict the sequence with higher probability, than trivial models and thereby result in shorter code sequences. Typically, such sophisticated models employ a number of *contexts* or *conditioning events* to exploit the dependencies between symbols at different positions within a data sequence. For image data sequences where most intersymbol dependencies appear in linear form, *i.e.*, in the form of interpixel correlation, *decorrelation* has been proven to be a very successful modeling method to achieve efficient compression performance. In this section, a general review of decorrelation methods and the context-based source modeling technique is presented. After that, an outline of coding techniques is also given.

2.1.1 Decorrelation methods

As mentioned earlier, in order to achieve good compression performance, multiple contexts are employed in the source models for data sequences to be compressed. In building such a source model for a data sequence, one of the most straightforward approaches is to choose the immediately preceding symbols of any given symbol within the sequence to form the contexts. The complexity of such a model is, however, staggering even when the number of the selected preceding symbols is modest. In the case of compressing a digitized image, for example, one may build a third-order Markov model by using the west, north, and northwest neighbors of any given pixel to

define contexts, prompted by the conjecture that the three neighbors would provide a significant amount of mutual information about the gray-level value of the given pixel. For such a third-order model, supposing that the gray-level values range from 0 to 255, corresponding to 8-bit quantization, there will be a total number of $256^3 \approx 17 \times 10^6$ contexts. Clearly, this large number of contexts has brought the complexity of the model to a level that hinders the modeling algorithm from practical implementation. Yet, notice here that any given pixel of the image is assumed to depend only on its three neighbors and sometimes a more extensive correlation might have to be taken into consideration in order to achieve satisfactory compression performance.

By taking into account the fact that image data are highly correlated, the general strategy to solve or alleviate the above problem is to apply some linear or nonlinear transform operations to the original image such that the transformed values are less correlated or even uncorrelated and consequently can be modeled as outcomes of an information source with smaller number of contexts. Such a transformation is referred to as decorrelation. If it is invertible, then lossless compression of the image can be accomplished; otherwise, such as in the case where the transformed values are quantized, some distortion in the reconstructed image is inevitable.

Decorrelation can be accomplished in either the spatial or the frequency domain. A well-known and important technique for spatial domain decorrelation is *linear prediction* [67, 99, 98]. In linear predictive decorrelation, the value of any pixel in a given image is predicted using a linear combination of the values of its neighboring pixels. The coefficients used in the linear prediction are generally chosen to minimize the mean squared value of the prediction errors and, in turn, can be shown as functions of the autocorrelation values of the pixels in the image. As a result, the prediction

errors will be, usually, small and less correlated than the original image data. In order to enhance the decorrelation performance, the determination of prediction coefficients can also be made adaptively according to the local changes in image data statistics [70, 98].

Another typical set of spatial domain decorrelation techniques is *multiresolution* [113], including *hierarchical interpolation* [98] and *reduced difference pyramid* [39]. An alternative to the decorrelation in the spatial domain is that in the the frequency domain [3, 23] which includes prominent techniques such as *Karhunen-Loeve transform* (KLT), *discrete cosine transform* (DCT), *Walsh-Hadamard transform* (WHT), and *wavelet transform* (WT) [72, 28, 74].

2.1.2 Statistical source modeling

Substantial amount of intersymbol dependencies, especially nonlinear dependencies, may still remain in the sequences of decorrelated pixels. It is therefore possible to enhance the compression performance by using source models with multiple contexts for such data sequences [88].

The fundamentals of statistical source modeling using multiple contexts were put together for the first time by Rissanen and Langdon in their landmark paper [97]. Since then, several algorithms have been proposed for selecting contexts [87, 117] based on different heuristics or trial-and-error procedures. Most of these algorithms center on generating new contexts by splitting the existing contexts in the hope that the new contexts would bring about more mutual information about the probabilities of different symbols. Among the typical ones, more specifically, *prediction by partial matching* [25] and *dynamic history prediction* [117] split contexts based on the

frequencies of occurrences of the contexts; while the one proposed by Ramabadran and Cohn [87], guided by the notion that the contexts should be selected to ensure significant difference of conditional data statistics under each context, splits contexts based on the frequencies as well as the conditional data entropies under the contexts.

Another important aspect of source modeling is the estimation of source statistics under different contexts, *i.e.*, the determination of model parameters, which not only supplies parameters to the coding process but also feeds back references to the context generation process. Detailed discussions on this issue have been given in [97, 24, 9, 117].

2.1.3 Coding techniques

Many efforts have been documented in the literature to develop efficient coding techniques. Through these efforts, three typical coding techniques, *viz.*, the *Huffman coding* [49], *arithmetic coding*, and the *Lempel-Ziv coding* [120, 121, 122], have been developed and used in many applications. All the three coding techniques encode data blocks of fixed length into bit strings of variable length and, therefore, are called *variable-length coding* techniques. In addition, Huffman coding and arithmetic coding are also referred to as *entropy coding* techniques since the average length of the encoded bit strings is close to the entropy of the data blocks being encoded.

Among the three coding techniques, arithmetic coding has been shown to possess many advantages. Its compression efficiency is high even when the entropy of the source model is low, in which case the use of Huffman coding is quite inefficient. Also, arithmetic coding is computationally efficient and can easily accommodate source models with multiple contexts and adaptive statistics. These, in turn, make it most

suitable for applications where source modeling and coding are clearly separated.

Although the concept behind arithmetic coding was known in the late 40's [1], practical implementation of the technique was possible only in the late 70's primarily through the efforts of Rissanen and Langdon [95, 96]. Several other researchers were also involved in this development [83, 101, 47, 56]. Moreover, excellent tutorial articles on arithmetic coding are now available [64, 115].

Based on the above overview of decorrelation, source modeling, and coding techniques, in the next section, a practical image compression method called *differential pulse code modulation* (DPCM) that uses linear predictive decorrelation is described. The similarities and differences between the applications of the method to lossy and lossless image compression are also described. Then, in Section 2.3, a new context-based source modeling technique pertaining to image compression applications will be presented.

2.2 Lossy and Lossless DPCM Methods

Suppose a digitized (*i.e.*, spatially sampled and amplitude quantized) image is represented mathematically as an $M \times N$ matrix $\mathbf{g} = [g(m,n)]$ with the origin at the top-left corner. In this representation, M and N denote the numbers of rows and columns, respectively, and $g(m,n)$ denotes the gray-level of the image pixel at the mth row and nth column. The gray-level of a pixel usually assumes an integer value, and the range of such an integer is typically between 0 and $2^B - 1$ where B is the number of bits used to represent each pixel. For example, when $B = 8$, $g(m,n)$ ranges between 0 and 255. Since the pixels in an image are highly correlated, one may predict the value of each pixel in terms of the neighboring pixel values with a fair

degree of accuracy. Linear combinations of the preceding pixel values (with respect to a scan direction) are typically used for this purpose. For example, by assuming a left-to-right, top-to-bottom scan, the predicted value of $g(m,n)$ in terms of the west, north, and northwest neighbors can be expressed as

$$\hat{g}(m,n) = a_1\ g(m,n-1) + a_2\ g(m-1,n) + a_3\ g(m-1,n-1), \qquad (2.1)$$

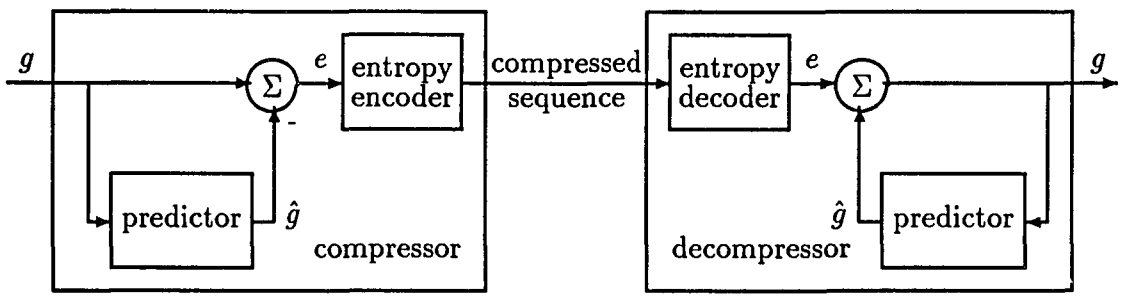
where a_1, a_2, and a_3 are constants referred to as the *prediction coefficients.* Typical values of these coefficients are 0.95 for a_1 and a_2 and -0.9 for a_3 [98]. The prediction error is given by

$$e(m,n) = g(m,n) - \hat{g}(m,n). \qquad (2.2)$$

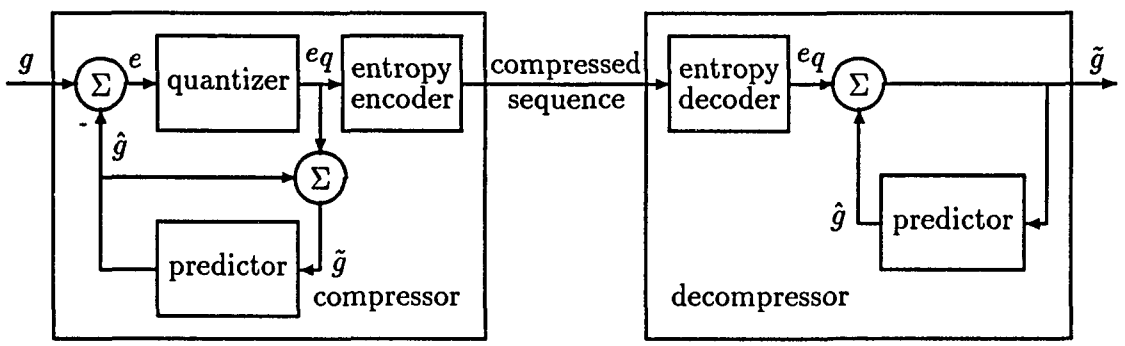
In the lossless DPCM compression system shown in Figure 2.1a, the predicted value $\hat{g}(m,n)$ is clipped at both ends and rounded[1] so that it assumes an integer value in the same range as $g(m,n)$, *i.e.*, from 0 to $2^B - 1$. The prediction error $e(m,n)$ in this case is also an integer ranging from $-(2^B - 1)$ to $(2^B - 1)$, *e.g.*, -255 to $+255$ for 8-bit quantization. Even though the range of $e(m,n)$ is nearly twice that of $g(m,n)$, the distribution of $e(m,n)$ is highly peaked near 0, and entropy coding of the errors results in a compressed representation compared to the original. At the decompressor, the decoded error value is added to the predicted value to recover $g(m,n)$ exactly, assuming that there are no channel errors in the compressed data.

A lossy DPCM compression system is illustrated in Figure 2.1b. Compared to Figure 2.1a, the compressor in Figure 2.1b differs in two respects: 1) it uses the

[1] A rounding operation maps a real number argument to the nearest integer. Two types of such an operation will be identified in the following discussion. They are respectively $\Re^+[\cdot]$ that rounds up a number with a decimal fraction of 0.5 and $\Re^-[\cdot]$ that rounds down a number with a decimal fraction of 0.5



(a)



(b)

Figure 2.1: DPCM systems for (a) lossless and (b) lossy compression

reconstructed pixel values $\tilde{g}(m,n)$ in forming the predicted values, and 2) it quantizes the prediction errors. Also, the predicted value $\hat{g}(m,n)$ is not rounded so that this value, the prediction error $e(m,n)$, and the quantized prediction error $e_q(m,n)$, in general, are real. $\hat{g}(m,n)$ is, however, clipped into between 0 and $2^B - 1$ such that the range of $e(m,n)$ is from $-(2^B - 1)$ to $(2^B - 1)$ as in the lossless case. The quantization operation is irreversible and is the source of loss (or distortion) in the lossy DPCM method. Let the quantization error $q(m,n)$ be defined as

$$q(m,n) = e(m,n) - e_q(m,n). \tag{2.3}$$

In general, $q(m,n)$ is real and depends on the prediction error $e(m,n)$ and the design of the quantizer. Assuming that the reconstructed pixel value $\tilde{g}(m,n)$ is also rounded into an integer[2] using the $\Re^+[\cdot]$ operator, one can easily show that

$$r(m,n) = g(m,n) - \tilde{g}(m,n) = \Re^-[q(m,n)]. \tag{2.4}$$

That is, the reconstruction error $r(m,n)$ is also an integer. The structures of the decompressors are essentially the same in both Figures 2.1a and 2.1b. At the decompressor in Figure 2.1b, the predicted value $\hat{g}(m,n)$ is once again clipped and the reconstructed value $\tilde{g}(m,n)$ is rounded.

Notice that the lossy DPCM system in Figure 2.1b includes an entropy encoder in the compressor to encode the quantized prediction errors as well as a corresponding entropy decoder in the decompressor to decode the compressed sequence. Due to this usage of entropy coder/decoder, the lossy DPCM system is also called an *entropy-*

[2]In order to keep this integer value between 0 and $2^B - 1$, a clipping operation is also needed. However, for the sake of simplicity, the clipping operation will be ignored in the discussion hereafter.

coded DPCM system to distinguish it from other more traditional lossy DPCM systems which do not use entropy coding. The entropy-coded lossy DPCM system can be related to the lossless system in Figure 2.1a as follows. Suppose the quantizer used in Figure 2.1b is a uniform, mid-tread quantizer with quantization interval $\delta = 1$ and integer quantization levels as shown in Figure 2.2a. Then, one can easily verify that the DPCM system in Figure 2.1b is lossless. Actually, in this case, the quantizer performs the rounding operation $\Re^-[\cdot]$ and, since $g(m,n)$ is an integer, the rounding operation can be moved to $\hat{g}(m,n)$ (now becoming $\Re^+[\cdot]$). As a result, the quantizer can be replaced by an identity map, *i.e.*, no quantization at all. The equivalence between the two systems in Figure 2.1 is now obvious.

Typically, the prediction error $e(m,n)$ is modeled as a *Laplacian* distributed random variable [53]. An assumed probability density function (PDF) $f(e)$ of the prediction errors is shown in Figure 2.3. Given the PDF $f(e)$, the discrete probability distribution of the quantization levels can be easily found as

$$P_i = \int_{e_{qi}-\delta/2}^{e_{qi}+\delta/2} f(e)de \tag{2.5}$$

for $i = -\Gamma, \cdots, 0, \cdots, \Gamma$; where Γ is the index of the largest quantization level. Consequently, the *zeroth-order entropy* of the quantized prediction errors $e_q(m,n)$ can be expressed as [36]

$$H(e_q) = -\sum_{i=-\Gamma}^{\Gamma} P_i \log_2 P_i. \tag{2.6}$$

Using a memoryless model for the quantized prediction errors, *i.e.*, assuming that $e_q(m,n)$ can be modeled as independent and identically distributed (iid) random numbers, and an efficient entropy coding technique such as arithmetic coding, the image can be compressed close to $H(e_q)$ bits per pixel with this method.

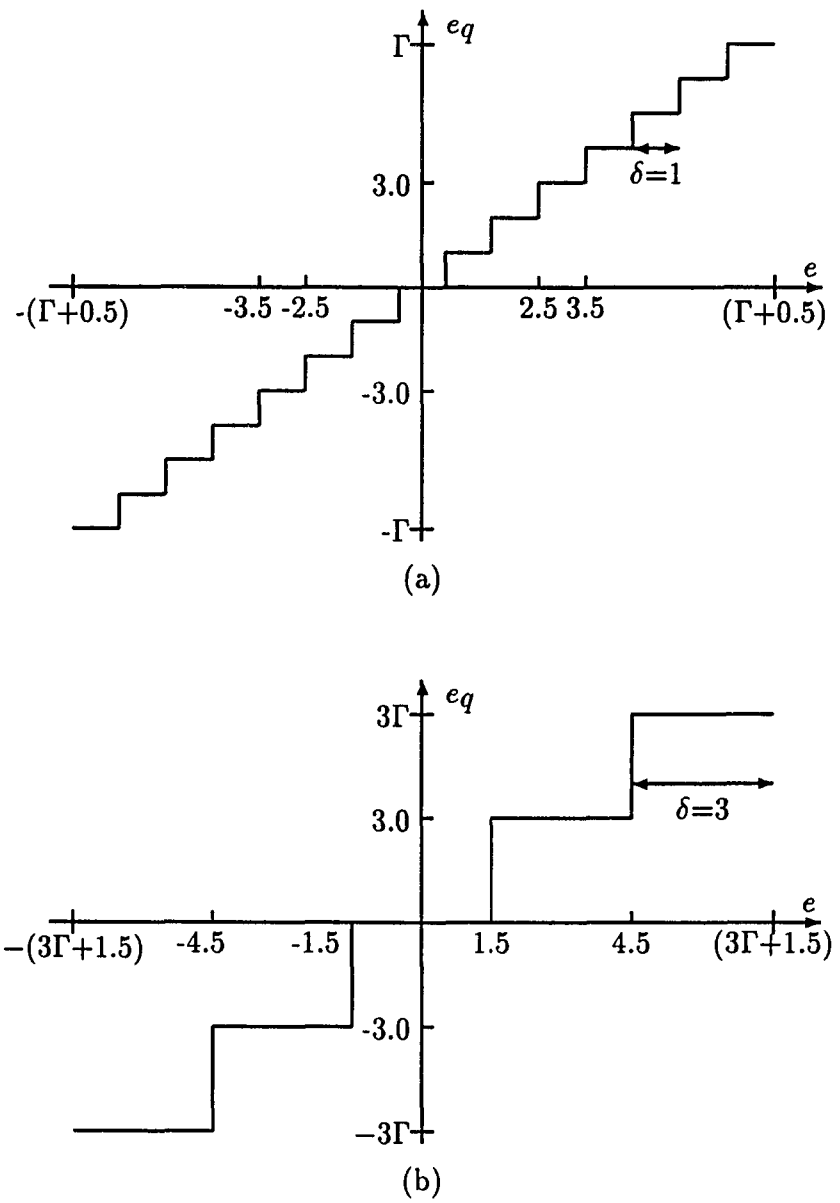


Figure 2.2: Quantizer characteristics for which the DPCM method is (a) lossless and (b) lossy with maximum error limited to ± 1

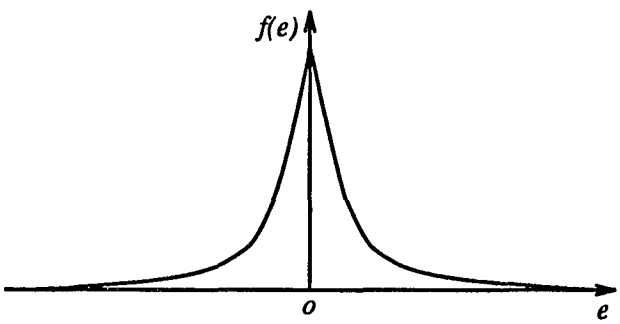


Figure 2.3: Typical PDF of prediction error

To obtain higher compression, some distortion must be allowed in the reconstructed image and this happens when the size of the quantization interval δ is increased beyond one. For example, in Figure 2.2b, the characteristic of a uniform, mid-tread quantizer with $\delta = 3$ and integer quantization levels is shown. The number of quantization levels in this case is only about a third of the number of levels in the quantizer of Figure 2.2a. If we assume that the prediction error PDF in this case remains the same as that when $\delta = 1$ and the standard deviation of the errors is large compared to δ, then the entropy of quantized prediction errors $H(e_q)$ should be reduced by about $\log_2 3 = 1.585$ bits per pixel. If the lossless method, for example, achieves a compression factor[3] of 3 for a particular image with $B = 8$, then this lossy method should achieve a compression factor of about 7.4. It should be pointed out that here the distortion introduced in the reconstructed image is also quite small. In fact, the reconstruction error $r(m,n)$ is either 0 or ± 1 for this quantizer. In practice, however, the improvement obtained in compression performance is somewhat smaller. This is mainly because the predicted values, now based on the lossy recon-

[3]A compression factor is defined as the ratio of the original image size to the length of compressed sequence.

structions, are less accurate, which in turn causes the prediction error PDF to be less peaked. Nevertheless, reasonable improvements in compression performance should be obtained through this method by allowing some small amount of distortion in the reconstructed image.

For the quantizer shown in Figure 2.2b, the lossy DPCM system of Figure 2.1b can be modified as follows without affecting its operation:[4] the predicted value $\hat{g}(m,n)$ can be rounded to an integer using the $\Re^+[\cdot]$ operator so that the prediction error $e(m,n)$ is also an integer. With such a rounding operation, the input to the quantizer becomes discrete and, subsequently, the operation of the quantizer can be interpreted as the mapping of the set $\{-1,0,1\}$ to 0, the set $\{2,3,4\}$ to 3, the set $\{5,6,7\}$ to 6, and so on. The difference between the lossless and lossy DPCM systems is now evident: the lossless method maps each integer prediction error value into a unique level whereas the lossy method, in this case, maps three different error values to the same level.

2.3 Context-Based Source Modeling Technique

In the entropy coding of the quantized prediction error sequence $e_q(m,n)$, a memoryless source model is typically used [98], *i.e.*, the quantized prediction errors are assumed to be independent and identically distributed. As mentioned in the last section, higher compression can be achieved through statistical conditioning of the error sequence. Suppose a set of conditioning events or contexts $C = \{c_k : 1 \leq k \leq K\}$ is defined such that the context for each quantized error sample $e_q(m,n)$ can be

[4]This modification works whenever each of the quantization intervals of the quantizer is of the form $(i+0.5, i+j+0.5]$ where i and $j \geq 1$ are integers and the corresponding quantization level is also an integer.

determined uniquely. Then, the conditional probability distribution of the quantized prediction errors under each context can be estimated, and these distributions can be subsequently used in the encoding of the sequence. In this case, the compression that can be achieved is lower bounded by the conditional entropy

$$H(e_q|C) = -\sum_{k=1}^{K} P_k \sum_{i=-\Gamma}^{\Gamma} P_{i|k} \log_2 P_{i|k}, \tag{2.7}$$

where P_k is the probability of the context c_k, and $P_{i|k}$ is the conditional probability that a prediction error $e(m,n)$ is quantized into the quantization level e_{qi} under the context c_k. It can be shown that the conditional entropy $H(e_q|C)$ is less than the (marginal) entropy $H(e_q)$ by an amount equal to the average mutual information provided by the contexts about the quantized prediction errors [36]. Since the average mutual information is non-negative, improved compression is possible with this technique.

In the following, one approach to defining contexts for entropy-coded DPCM is described. This approach uses the estimated gradients and the predicted value of each pixel to form the context for the quantized prediction error of the pixel and has been shown to be quite successful in improving compression performance for lossless compression applications [19, 88, 20]. First, let the horizontal and vertical gradients at the mth row and nth column of the (reconstructed) image be defined respectively as

$$d_h(m,n) = \tilde{g}(m-1,n) - \tilde{g}(m-1,n-1) \tag{2.8a}$$

and

$$d_v(m,n) = \tilde{g}(m,n-1) - \tilde{g}(m-1,n-1). \tag{2.8b}$$

Clearly, these two gradients assume integer values and range between $-(2^B-1)$ and

$(2^B - 1)$. The d_h-d_v plane is then partitioned into several regions as shown in Figure 2.4 and each of the regions is identified as a primary context. Under each primary context, secondary contexts are defined based on the predicted value $\hat{g}(m,n)$. More specifically, the value range of $\hat{g}(m,n)$ is partitioned into several intervals and each of the intervals is identified as a secondary context. Thus, the context of any quantized prediction error value $e_q(m,n)$ can be uniquely determined using the previously reconstructed pixel values at both the compressor and decompressor. The rationale behind this definition of contexts can be explained as follows. The PDF $f(e)$ of the prediction error sequence can be considered as a weighted average of different conditional error distributions (PDF's) corresponding to different sets of values of $\{\tilde{g}(m,n-1), \tilde{g}(m-1,n), \tilde{g}(m-1,n-1)\}$ used in the prediction of $g(m,n)$ [29]. In general, these distributions are dissimilar and averaging them to obtain the marginal prediction error PDF would cause the uncertainty, *i.e.*, the entropy, to increase. The context-based source model provides a way to keep the dissimilar distributions separate and thereby leads to higher compression. In Figure 2.4, for example, the region (or primary context) with the smallest values of $|d_h|$ and $|d_v|$ can be considered as representing uniform sections within the image. Similarly, regions with high $|d_h|$ and low $|d_v|$, regions with high $|d_v|$ and low $|d_h|$, and regions with high $|d_h|$ and high $|d_v|$ can be regarded as representing respectively vertical edges, horizontal edges, and diagonal edges within the image. It is reasonable to expect that the error distributions corresponding to these different regions (or contexts) will be dissimilar and should therefore be kept separate. Likewise, the distributions corresponding to different ranges of the predicted values are likely to be different and should also be kept separate. For example, when $\hat{g}$ is close to 0 (255), the correpsonding distribution of

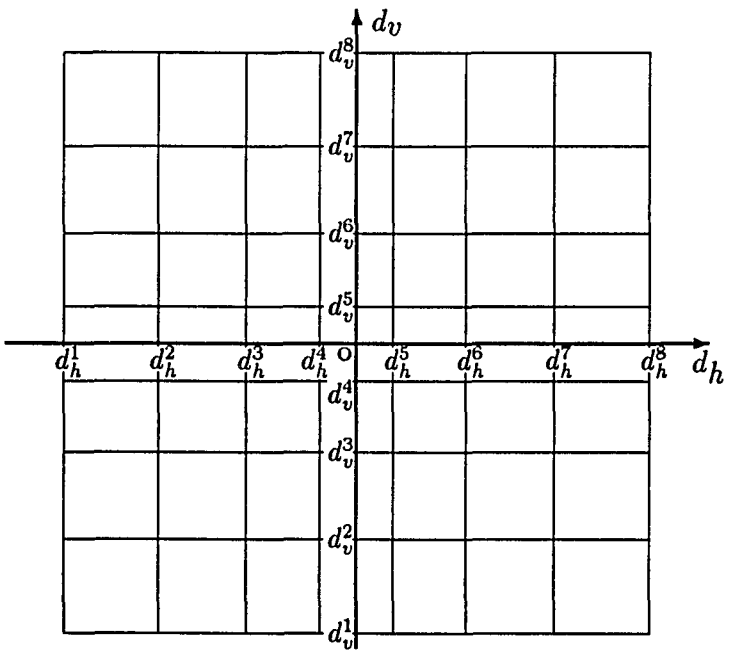


Figure 2.4: Partitioning of the d_h-d_v plane to define the primary contexts

prediction errors e, and consequently that of e_q, is essentially positive (negative).

When using a multiple-context source model as described in the above, traditional coding techniques such as Huffman coding are rather cumbersome to use and can be inefficient. Arithmetic coding, on the other hand, can handle multiple-context as well as adaptive statistics quite easily and can provide nearly optimal compression efficiency with respect to any context-based source model. In arithmetic coding, the entire sequence is coded as a single entity without dividing it into blocks. The probability of the sequence with respect to the source model is computed recursively using finite precision arithmetic operations; the sequence is then encoded by means of a unique binary fraction between 0 and 1, the length of which is approximately equal to the negative logarithm (to the base 2) of the probability. For more detailed

information about the implementation of arithmetic coding, readers are referred to the papers by Pasco [83], Jones [56], and Witten *et al.* [115].

3. ENTROPY-CODED DPCM FOR IMAGE COMPRESSION

Based on the descriptions of the entropy-coded DPCM method and context-based source modeling technique presented in the last chapter, a new scheme for image compression, especially for near-lossless compression, has been developed. This new scheme, from the source modeling point of view, can be regarded as an enhanced version of the entropy-coded lossy DPCM method and will be referred to as ECDPCM hereafter. In this chapter, some implementation details of the ECDPCM scheme are described. Compression and reconstruction performances of the scheme versus other schemes are also presented.

3.1 Predictor and Quantizer

3.1.1 Predictor

The well-known *planar predictor* [82] is used in the implementation of the ECDPCM scheme. This predictor provides consistently good performance for the class of images under investigation and is quite easy to implement. The coefficients of the planar predictor are given by $a_1 = a_2 = 1$ and $a_3 = -1$. Since all these coefficients are integers, the predicted values $\hat{g}(m,n)$ and the prediction errors $e(m,n)$ are also integers. This means that the quantization levels, *i.e.*, the values of the quantized prediction errors $e_q(m,n)$, can be integers without any loss of performance.

3.1.2 Quantizer

The design of scalar quantizers for input signals with common distributions under different optimization criteria has received considerable attention in the literature [53]. In the case of entropy-constrained quantization, optimal designs (*e.g.*, designs that minimize the mean squared quantization error) for different input distributions such as Gaussian and Laplacian and for different output rates have been developed [119, 12, 84]. However, it is also known that, for common distributions and a wide range of output rates, the simpler uniform quantizers can provide performances quite close to those of optimal quantizers [84] which usually appear to be nonuniform. In the implementation of ECDPCM, therefore, the basic quantizer design is that of a uniform quantizer, *i.e.*, a quantizer with constant-width quantization intervals and the quantization level within each interval having been selected to minimize the mean squared quantization error.

Although the mean squared error is typically used as a measure of distortion introduced by quantization, its usefulness in image compression is questionable since it does not take into account the properties of human visual perception. Several studies to determine the visibility of quantization errors have been made in an attempt to develop subjectively more meaningful distortion measures for the design of quantizers [111, 79, 108, 54]. An important result arising out of these studies is the *visibility threshold* for quantization errors. This threshold, typically measured as a function of the prediction error, indicates the maximum allowable error in quantization for different prediction errors which is not noticeable in the reconstructed image. For example, it has been reported that the capability of the human eye to perceive small changes in luminance near an edge decreases as the edge becomes sharper [108]. This

in turn implies that higher quantization errors can be allowed for quantizing large prediction errors without the distortion in the reconstructed image being visible to human observers or, in other words, the visibility threshold increases with the prediction error value. To a certain extent, the visibility threshold function depends on the predictor and the image material used for its measurement. However, its applicability has been found to be fairly general [108]. For the design of quantizers in the implementation of ECDPCM, the visibility threshold function given in [76] obtained by using a two-dimensional predictor ($a_1 = 1$, $a_2 = 0.5$, and $a_3 = -0.5$) and critical television pictures has been used.

Two different quantizers have been designed for the ECDPCM scheme to compress 8-bit digitized images. These quantizers incorporate the visibility threshold criterion, *i.e.*, the quantization errors do not exceed the threshold for any prediction error, and have characteristics as shown in Figure 3.1. From the figure, it can be seen that the first quantizer is a uniform, mid-tread quantizer with $\delta = 3$ and has 171 quantization levels. The second quantizer, *i.e.*, the one in Figure 3.1b, is also a mid-tread quantizer and, except for the quantization interval around zero, is uniform with $\delta = 2$ and 255 quantization levels. Both quantizers introduce a maximum error of ± 1 in the reconstructed images. Henceforth, the two ECDPCM implementations using the quantizers shown in Figure 3.1 will be referred to as ECDPCM-I and ECDPCM-II, respectively.

3.2 Generation of Contexts and Estimation of Source Model Statistics

In the ECDPCM scheme, contexts of the source model for the quantized prediction errors are generated adaptively [20] using a technique referred to as *context*

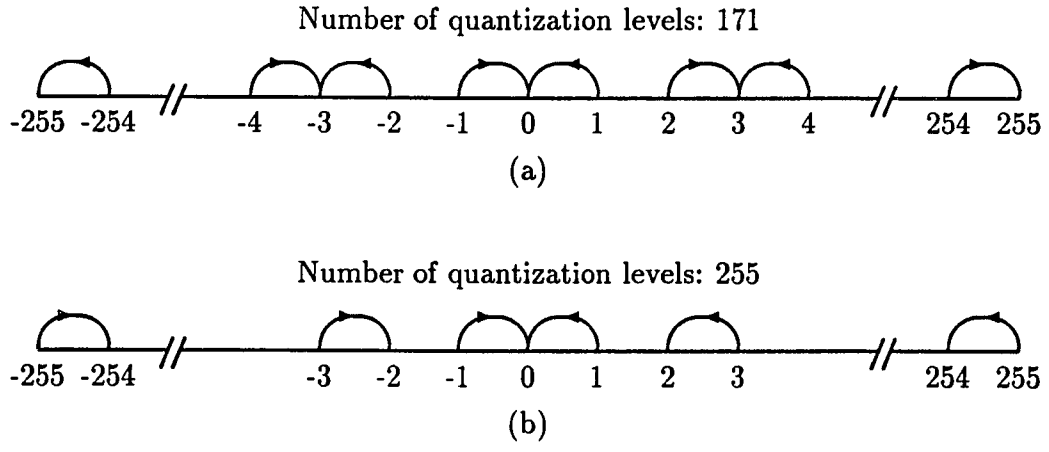


Figure 3.1: Characteristics of the quantizer used in (a) ECDPCM-I and (b) ECDPCM-II

splitting [88, 87]. This technique is described as follows. Suppose c_k represents a context of the source model satisfying the following two conditions. First, the probability of c_k is fairly significant. This significance can be determined by keeping a count of the number of times that c_k occurs and by checking if the count exceeds a preset threshold N_c. Second, the entropy of the conditional distribution under c_k is significantly high. This can be determined by verifying that the probability of the most probable quantized prediction error value under c_k is below a preset threshold $\hat{P}_c$. If both conditions are satisfied, then the context c_k is replaced by two new contexts c_{k0} and c_{k1} which represent a partition of c_k. It can be argued [87, 88] that such a replacement leads to a reduction in the entropy of the quantized prediction error sequence and thereby to improved compression performance. To generate the contexts adaptively, one can then start with a single (null) context and successively split selected contexts as described above until the required number of contexts are generated.

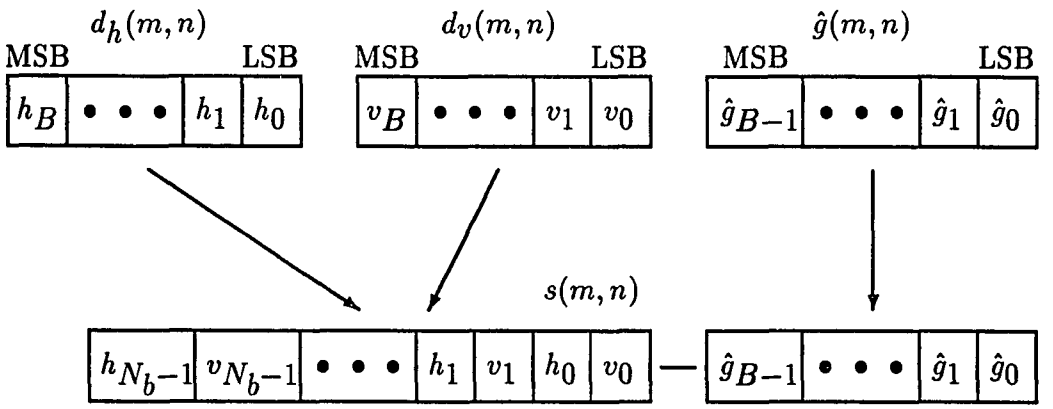


Figure 3.2: Combining bits from the binary representations of $d_h(m,n)$, $d_v(m,n)$, and $\hat{g}(m,n)$ to form the bit string $s(m,n)$

To explain the context generating approach in the ECDPCM scheme further, let us consider the 2's complement binary representations of the gradients $d_h(m,n)$ and $d_v(m,n)$ as well as that of the predicted value $\hat{g}(m,n)$. As shown in Figure 3.2, bits from these representations can be combined together to form the bit string $s(m,n)$. The string $s(m,n)$ has two parts: 1) the left-hand (*i.e.*, the most significant) part consists of N_b least significant bits of $d_h(m,n)$ and $d_v(m,n)$ interlaced, and 2) the right-hand (*i.e.*, the least significant) part consists of the B bit representation of $\hat{g}(m,n)$. The string $s(m,n)$ of length $2N_b+B$ bits can be used to identify the context: the left-hand part identifies the primary context, *i.e.*, a region in the d_h-d_v plane, whereas the right-hand part identifies the secondary context, *i.e.*, a range of values of $\hat{g}(m,n)$. For example, suppose $s(m,n)$ = 101010110011-10****** where * indicates a "don't-care" entry and N_b is taken to be 6. Then, the primary context is specified by $d_h(m,n) = -3$ and $d_v(m,n) = 5$, and the secondary context is specified by $128 \leq \hat{g}(m,n) \leq 192$. In the formation of the left-hand part of $s(m,n)$, only

the N_b least significant bits of $d_h(m,n)$ and $d_v(m,n)$ are used since most of the gradient estimates assume small values. Also, the bits from $d_h(m,n)$ and $d_v(m,n)$ are interlaced to assign equal importance to the two gradients.

Given the bit string representation of a context as described above, the contexts of a source model can be conveniently represented by means of a rooted variable-depth binary tree [87]. In such a tree, the terminal nodes, or more accurately, the bit patterns corresponding to the paths from the root to the different terminal nodes represent the contexts. In creating the contexts, one starts with just the root node (representing the *null* context). If a terminal node satisfies the conditions for splitting, then one extends this node by one more level thereby replacing it with two new terminal nodes. This process continues until the total number of terminal nodes, *i.e.*, contexts, reaches a predetermined maximum K_C. In our implementation, the number of primary contexts and the number of secondary contexts are limited respectively to K_P and K_S so that $K_C = K_P + K_S$. Notice that the primary contexts represented by nodes closer to the root are typically generated before the generation of the secondary contexts.

The multiple-context source model statistics, *i.e.*, the conditional distribution of the quantized prediction errors under different contexts, are estimated using the *cumulative adaptive technique* [97]. In this technique, the relative frequency counts of different quantized prediction error values are maintained from which the conditional probabilities can be easily estimated. Initially, the relative frequency counts are set to 1 or some other values reflecting our *a priori* knowledge of the distribution. Afterwards, each time a particular quantized prediction error value is coded, the corresponding frequency count is incremented by one. The updated frequency counts

are then used in the coding of the next quantized prediction error as well as in the context generating procedure, and so on. In this manner, both source modeling and coding can be accomplished simultaneously such that a single scan of the original image is sufficient for compressing it.

In the implementation of the ECDPCM scheme, separate counts are not maintained for all possible quantized prediction error values in order to reduce the memory requirement and to speed up the encoding and decoding operations. Instead, since the distributions are typically peaked near zero, error values equal to or larger than a certain value ξ_S can be combined and assigned a single count. Similarly, error values equal to or smaller than $-\xi_S$ can be combined and assigned another single count. The total number of counts is thus only $2\xi_S + 1$ for each context. When encoding a quantized prediction error value above ξ_S, for example, one first encodes ξ_S using its count (or probability) and then encodes the specific error value assuming that all the values forming the ξ_S symbol are equally probable.

3.3 Experimental Results and Discussions

Both of the ECDPCM-I and ECDPCM-II algorithms have been implemented in software and several experiments have been conducted to evaluate the compression performance and the reconstructed image quality. The results of these experiments are presented in this section. Two groups of medical images were used in the experiments. The first group consists of 15 magnetic resonance (MR) images. Nine of these images are of size $512 \times 696 \times 8$ (512 rows, 696 columns, and 8 bit quantization) while the other six are of size $480 \times 648 \times 8$. The second group has 5 ultrasound (US) images of size $512 \times 640 \times 8$.

The compression results obtained by applying the two ECDPCM algorithms to the MR and US images are given in Table 3.1, together with the results of the conventional lossless DPCM scheme illustrated in Figure 2.1a. These results are the sizes of the compressed sequences in number of bits per pixel (*bpp*). The corresponding compression factors, of course, can be calculated as $8/bpp$.

In evaluating the performance of the conventional DPCM scheme, the same planar predictor as that in ECDPCM-I and ECDPCM-II was used to decorrelate any given image. A memoryless source model was used in encoding the decorrelated pixels. That is, neither partitioning of the d_h-d_v plane nor the $\hat{g}$ value range occurred in this case. The statistical parameters of the memoryless model, *i.e.*, the probabilities of different prediction error values, were estimated using the cumulative adaptive technique with $\xi_S = 255$. In other words, a full relative frequency count table with 511 entries was used. Moreover, arithmetic coding was used to encode the prediction errors. In the two single-context lossy ECDPCM implementations, on the other hand, a memoryless model, the cumulative adaptive technique for probability estimation with a full frequency count table, *i.e.*, with 171 entries for ECDPCM-I and 255 entries for ECDPCM-II, and arithmetic coding were used. Therefore, a comparison between the second column and the third column of Table 3.1, as well as between the second column and the fifth column, shows the improvement in compression that can be achieved by allowing a small amount of distortion in the reconstructed images. Similarly, comparisons between the third column and the fourth column, and between the fifth column and the sixth column demonstrate the improvement in compression due to the use of a context-based source model.

In the implementations of the ECDPCM-I and ECDPCM-II algorithms with

Table 3.1: Compression performances[a] of ECDPCM-I and ECDPCM-II

Image	DPCM	ECDPCM-I		ECDPCM-II	
	lossless	single context	multiple contexts	single context	multiple contexts
MR01	1.75	1.19	0.74	0.97	0.70
MR02	2.24	1.48	1.04	1.32	1.04
MR03	2.21	1.49	1.04	1.32	1.02
MR04	1.93	1.28	0.84	1.03	0.76
MR05	1.96	1.30	0.84	1.06	0.78
MR06	2.04	1.37	0.91	1.16	0.86
MR07	2.53	1.66	1.18	1.59	1.25
MR08	2.52	1.66	1.19	1.59	1.25
MR09	2.51	1.64	1.19	1.57	1.24
MR10	3.54	2.03	1.43	2.02	1.55
MR11	3.45	1.97	1.37	1.95	1.49
MR12	3.44	1.99	1.37	1.96	1.50
MR13	2.61	1.59	1.05	1.41	1.09
MR14	4.39	2.62	1.78	2.65	2.07
MR15	2.55	1.55	1.01	1.35	1.04
UT01	3.46	1.85	1.13	1.99	1.36
UT02	3.31	1.70	1.02	1.86	1.23
UT03	2.75	1.27	0.85	1.32	0.91
UT04	3.88	2.26	1.45	2.48	1.76
UT05	3.86	2.25	1.44	2.45	1.74

[a]Measured in number of bits per pixel.

Table 3.2: Parameter values used in ECDPCM-I and ECDPCM-II

Algorithm	Image Type	$K_C = K_p + K_s$	$(N_C, \hat{P}_C)^a$	$(N_C, \hat{P}_C)^b$	ξ_s	N_b
ECDPCM-I	MR	300=250+50	(90, 1)	(130, 0.8)	4	7
	US	200=150+50	(5, 1)	(5, 1)	11	8
ECDPCM-II	MR	250=200+50	(20, 1)	(130, 0.8)	5	7
	US	250=200+50	(10, 1)	(5, 0.95)	10	7

[a]For splitting primary contexts.

[b]For splitting secondary contexts.

multiple contexts, the following parameters are of interest: the total number of contexts K_C, the threshold N_C and $\hat{P}_C$ used in context splitting for the primary and secondary contexts, the ξ_s value used in the relative frequency count table, and the number of bits N_b used in the formation of the bit string $s(m,n)$. The values of these parameters were selected after extensive experimentation to achieve maximum compression performance without excessive complexity and are listed in Table 3.2.

The reconstruction performances of the ECDPCM-I and ECDPCM-II algorithms in terms of SNR, PSNR, and PAE are shown in Tables 3.3 and 3.4, respectively. The *signal-to-noise ratio* (SNR) of a reconstructed image in decibals (dB) is defined as

$$\text{SNR} = 10\log_{10}\frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}[g(m,n)]^2}{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}[r(m,n)]^2}, \tag{3.9}$$

where $g(m,n)$ is the gray-level of the pixel at the mth row and nth column in the original image, and $r(m,n)$ is the error at $g(m,n)$ between the reconstructed and the original images. The *peak SNR* (PSNR) in dB is defined similarly by replacing $g(m,n)$ in the above formula with the peak gray-level value 255, and the *peak absolute error* (PAE) is simply the maximum value $|r(m,n)|$. In order to assess the reconstruction performances of the ECDPCM algorithms, the compression results and reconstructed

image qualities corresponding to the JPEG standard (entropy-coded DCT) are also included in Tables 3.3 and 3.4. For the purpose of comparison, these results were obtained by using the JPEG option of the "xv" utility with the quality levels adjusted such that the compression results are close to those of the ECDPCM algorithms.

Given the above descriptions of experimental results, we now can assess the ECDPCM algorithms and compare their performances with those of other compression schemes. From Tables 3.1, 3.3, and 3.4, several observations can be made. These observations are presented in the following together with some discussions.

By examining the fourth and sixth columns of Table 3.1 and the fifth column of both Tables 3.3 and 3.4, it can be seen that the ECDPCM algorithms achieve compression factors in the range from 4 to 11 with a PSNR of about 50 dB for 8-bit images. Compared to the conventional lossless DPCM algorithm for which the compression results are given in the second column of Table 3.1, the ECDPCM algorithms have reduced the compressed data volume by an amount of 1 to 2.5 bits per pixel. If the percentage improvement in compression of an ECDPCM algorithm over another algorithm is measured as $[(\chi_2 - \chi_1)/\chi_2] \times 100$ where χ_1 and χ_2 are the compression results corresponding to the ECDPCM algorithm and the other algorithm, respectively, then it can be seen that the ECDPCM-I algorithm achieves an average improvement of about 56.8% for MR images and 66.0% for US images over the conventional lossless DPCM algorithm while ECDPCM-II achieves improvement of 55.8% and 62.5%. Since the conventional DPCM algorithm is lossless, the reconstructed images produced by it are perfect, *i.e.*, the SNRs are infinite and the PAEs are zero. However, it should be pointed out that the subjective quality of a reconstructed 8-bit image with a PSNR of 50 dB is extremely good with the reconstruction

Table 3.3: Compression and reconstruction performances of ECDPCM-I versus the JPEG standard

Image	Compression Performance (bits per pixel)		Reconstruction Performance					
	ECDPCM-I multiple contexts	JPEG	ECDPCM-I			JPEG		
			SNR (dB)	PSNR (dB)	PAE	SNR (dB)	PSNR (dB)	PAE
MR01	0.74	0.77	38.4	51.3	1	31.4	44.3	24
MR02	1.04	1.04	38.7	50.7	1	31.3	43.3	22
MR03	1.04	1.07	38.7	50.6	1	31.9	43.8	23
MR04	0.84	0.86	37.4	50.6	1	30.4	43.6	24
MR05	0.84	0.87	38.1	50.7	1	31.0	43.6	24
MR06	0.91	0.93	37.9	50.6	1	31.0	43.8	25
MR07	1.18	1.18	38.2	50.6	1	30.9	43.3	24
MR08	1.19	1.22	38.4	50.5	1	31.5	43.7	22
MR09	1.19	1.19	38.7	50.6	1	31.3	43.2	25
MR10	1.43	1.44	42.1	49.8	1	34.0	41.7	21
MR11	1.37	1.41	41.7	49.8	1	33.4	41.5	24
MR12	1.37	1.37	42.9	49.9	1	34.4	41.4	22
MR13	1.05	1.07	43.1	49.8	1	39.8	46.4	13
MR14	1.78	1.80	42.9	49.9	1	34.4	41.4	17
MR15	1.01	1.04	42.8	49.8	1	39.6	46.6	15
UT01	1.13	1.16	34.8	50.7	1	27.2	43.0	17
UT02	1.02	1.03	34.3	50.7	1	26.3	42.8	22
UT03	0.85	0.88	35.3	50.6	1	27.3	42.6	26
UT04	1.45	1.45	35.9	50.6	1	27.3	42.0	19
UT05	1.44	1.46	36.0	50.6	1	27.5	42.0	17

Table 3.4: Compression and reconstruction performances of ECDPCM-II versus the JPEG standard

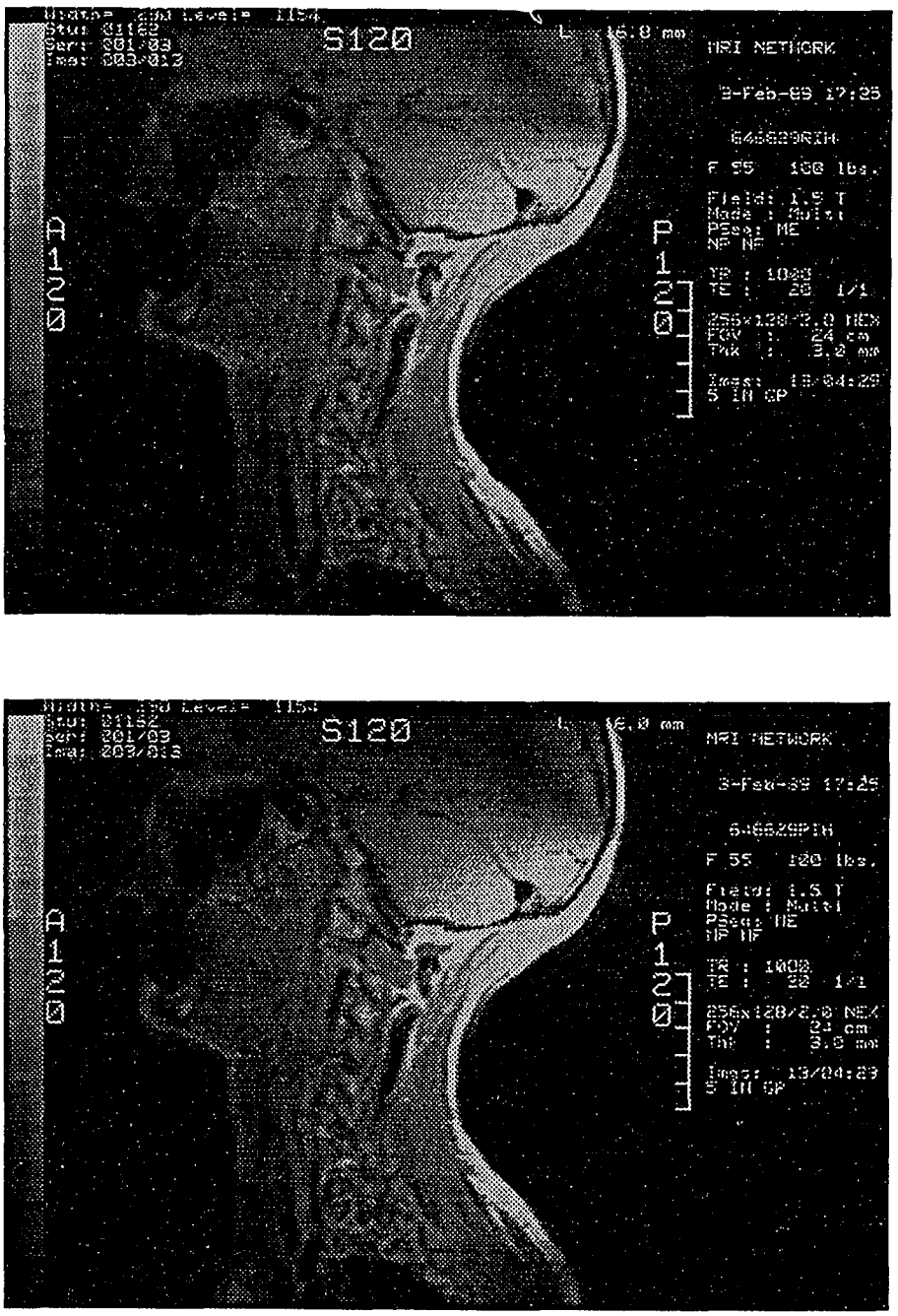
Image	Compression Performance (bits per pixel)		Reconstruction Performance					
	ECDPCM-II multiple contexts	JPEG	ECDPCM-II			JPEG		
			SNR (dB)	PSNR (dB)	PAE	SNR (dB)	PSNR (dB)	PAE
MR01	0.70	0.70	39.2	52.2	1	30.3	43.2	26
MR02	1.04	1.04	39.6	51.6	1	31.3	43.3	22
MR03	1.02	1.03	39.7	51.6	1	31.5	43.3	24
MR04	0.76	0.78	38.4	51.6	1	29.3	42.5	28
MR05	0.78	0.79	39.1	51.7	1	29.9	42.5	33
MR06	0.86	0.88	38.9	51.6	1	30.3	43.1	30
MR07	1.25	1.27	39.3	51.7	1	31.8	44.2	21
MR08	1.25	1.26	39.5	51.7	1	32.1	44.2	21
MR09	1.24	1.27	39.7	51.6	1	32.3	44.2	22
MR10	1.55	1.58	43.4	51.2	1	34.8	42.5	20
MR11	1.49	1.53	43.0	51.1	1	34.1	42.2	18
MR12	1.50	1.50	44.1	51.1	1	35.1	42.1	19
MR13	1.09	1.17	44.6	51.2	1	40.1	46.7	11
MR14	2.07	2.10	44.2	51.2	1	35.9	42.9	15
MR15	1.04	1.04	44.2	51.2	1	39.6	46.6	15
UT01	1.36	1.37	35.3	51.2	1	28.3	44.2	15
UT02	1.23	1.23	34.7	51.1	1	27.6	44.0	15
UT03	0.91	0.93	35.8	51.1	1	27.8	43.1	22
UT04	1.76	1.78	36.4	51.1	1	28.8	43.5	13
UT05	1.74	1.75	36.6	51.1	1	29.0	43.6	15

being practically indistinguishable from the original. Therefore, we can conclude that by allowing a small amount of distortion in the reconstructed images, the compressed data volume can be greatly reduced through application of the ECDPCM algorithms.

A comparison between the third column and the fourth column of Table 3.1, as well as between the fifth column and the sixth column, shows that the compression performance improves significantly with the use of context-based source model. More specifically, the average improvement in compression obtained for the ECDPCM-I algorithm is 31.8% for the MR images and 36.8% for the US images. For ECDPCM-II, the corresponding values are 23.8% for the MR images and 30.9% for the US images. Notice that no additional distortion is introduced in this case.

Since the compression results in columns 2 and 3 of both Tables 3.3 and 3.4 are roughly the same, the ECDPCM algorithms can be compared with the JPEG standard in terms of the qualities of their reconstructed images using the values given in columns 4 through 9 of the two tables. It can be seen that the ECDPCM algorithms provide about 7 to 8 dB higher SNR than the JPEG standard. More striking is the difference in the values of the peak absolute error. The implementation complexities of the ECDPCM algorithms are of the same order as the JPEG standard with much of the complexity arising from the entropy coding part. To illustrate the subjective quality of the reconstructions obtained using the ECDPCM algorithms, the original and reconstructed versions of MR08 and MR11 are shown in Figures 3.3 and 3.4, respectively. From these figures, it can be clearly seen that the reconstructions are subjectively indistinguishable from the originals.

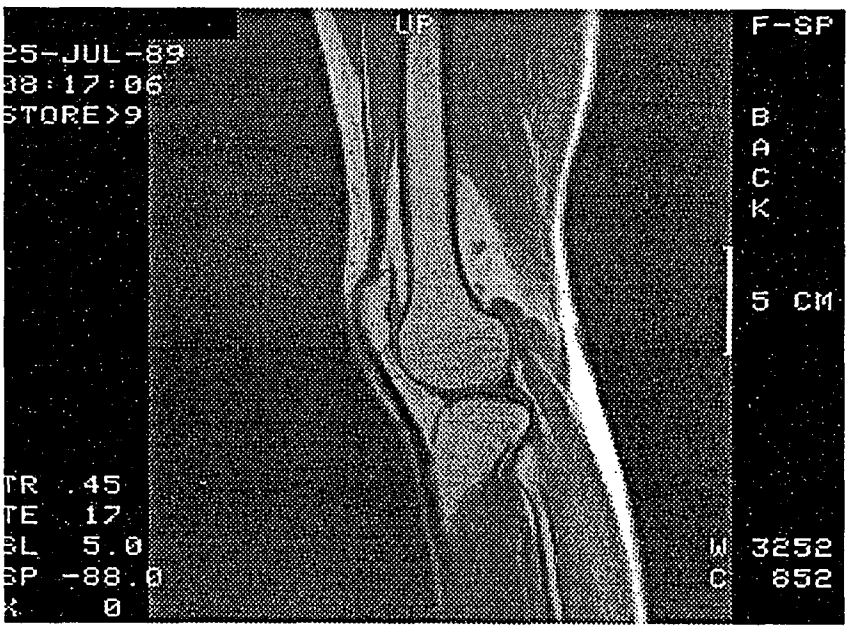
A comparison between the two ECDPCM algorithms with multiple contexts reveals that the ECDPCM-I algorithm provides a slightly higher compression whereas



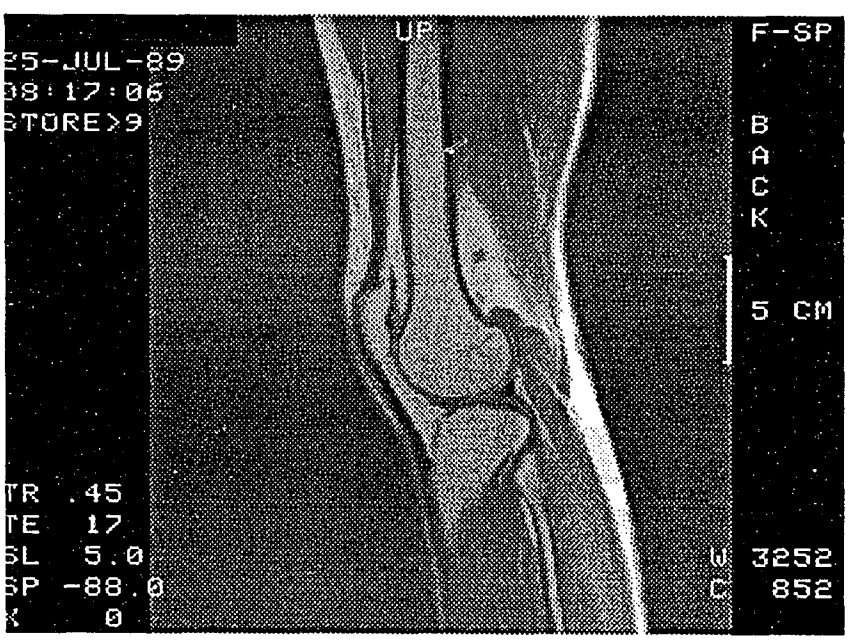
(a)

(b)

Figure 3.3: MR08 image compressed using ECDPCM-I: (a) original version and (b) reconstructed version



(a)



(b)

Figure 3.4: MR11 image compressed using ECDPCM-II: (a) original version and (b) reconstructed version

ECDPCM-II provides a slightly higher SNR. Comparing the third column and the fifth column of Table 3.1, it can be seen that the ECDPCM-II algorithm with a single context provides higher compression than the ECDPCM-I algorithm with a single context for the MR images. This is counter-intuitive since ECDPCM-II uses a quantizer with $\delta = 2$ while ECDPCM-I uses a quantizer with $\delta = 3$. The reason is that the prediction error PDF for these MR images are highly peaked near zero and most of the reduction in entropy is achieved by combining together the error values near zero, *viz.*, -1, 0, and +1, which is done by both quantizers. For other error values, the quantizer of ECDPCM-II combines only two values together, but it also introduces less quantization error thereby affecting the peakedness of the error PDF to a less extent compared to the quantizer of ECDPCM-I. Clearly, higher compression can be achieved by increasing the quantization interval δ, but this increase would also cause the visibility threshold criterion to be violated, especially for small prediction errors. Keeping the quantization interval small for small prediction errors and increasing the interval width for larger prediction errors, *i.e.*, using a nonuniform quantization characteristic, in general, does not result in improved compression performance. In fact, on some occasions, the compression performance is reduced due to the increased quantization errors.

3.4 Summary

The near-lossless, *i.e.*, lossy but high-fidelity, compression of digitized images using the entropy-coded DPCM method with a large number of quantization levels was investigated. Through the investigation, a new scheme called ECDPCM has been developed. This new scheme combines both the lossy and lossless entropy-coded

DPCM methods into a common framework with the nature of reconstruction depending only on the widths of quantization intervals for quantizing the prediction errors. Two different quantizers were considered in the implementation of the ECDPCM scheme. In order to enhance the compression performance, a context-based source model and the arithmetic coding technique were used. The contexts of the source model are generated adaptively through the use of the context splitting technique, and the source model statistics are also estimated adaptively using the cumulative adaptive technique. The new ECDPCM scheme was evaluated in terms of compression performance and reconstructed image quality using several medical MR and US images. It has been found that the scheme can provide compression factors in the range from 4 and 11 with a peak SNR of about 50 dB for 8-bit images. The use of multiple contexts has been found to have improved the compression performance by 25% to 30% for MR images and 30% to 35% for US images. Also, a comparison with the JPEG standard revealed that the new entropy-coded DPCM scheme can provide about 7 to 8 dB improvement in quality for the same compression performance.

The quantizers used in ECDPCM were designed based on the visibility threshold criterion derived from television pictures. While the results obtained were very good, similar criterion derived for medical images from the viewpoint of diagnostic quality may be helpful in improving the performance of the scheme. In previous work [19, 88], the lossless compression of digitized images has been investigated and the lossless DPCM method with a context-based source model has been found quite effective. Combining this with the results of the present investigation, it appears that the ECDPCM scheme, as a common entropy-coded DPCM method for both lossy and lossless applications, with controllable level of distortion (from zero distortion to some

small amount of distortion) would be quite useful for the high-fidelity compression of digitized images.

4. VECTOR QUANTIZATION

4.1 Introduction

From the discussions and experimental results presented in the last two chapters, it is seen that the compression of a data sequence, such as a sequence of image pixels, in scalar fashion is quite sophisticated even when only a small extent of inter-sample dependencies is exploited. To exploit a greater extent of the dependencies and achieve better compression performance, compression in vector fashion, *viz.*, *vector quantization* (VQ), provides a powerful yet viable alternative approach.

Vector quantization is a generalization of scalar quantization to a vector, *i.e.*, an ordered set of samples. The jump from one dimension to multiple dimensions is a major step and allows more extensive inter-sample dependencies present in a signal to be exploited. This, in turn, allows a wealth of new ideas, concepts, techniques, and applications to arise that often have no counterpart in the case of scalar quantization. More importantly, a fundamental result of Shannon's information theory [105, 106, 107] has indicated that better performance can always be achieved by compressing vectors instead of scalars, no matter whether or not the data source is memoryless. While some traditional compression schemes, such as the transform coding used in the JPEG standard, operate on vectors and achieve quite satisfactory compression performances, the quantization in these schemes is still accomplished on

scalars and hence the corresponding compression systems are inherently suboptimal: better performance is always achievable in theory by quantizing vectors instead of scalars, even if the scalars have been preprocessed such that they are uncorrelated or independent.

In this chapter, the basic definition of vector quantization, its properties, and a general algorithm for designing VQ systems are described. Several variations of VQ algorithms are also described or summarized.

4.2 Definition and Properties of Vector Quantization

4.2.1 Definitions

Mathematically, a vector quantizer Q of dimension K and size N is a many-to-one mapping from the K-dimensional vector space $\mathcal{R}^K$ to a finite set $\mathcal{Y}$ containing N output or reconstruction vectors, called *code vectors*. Thus,

$$Q : \mathcal{R}^K \rightarrow \mathcal{Y}.$$

The set $\mathcal{Y} = \{\mathrm{y}_i : i \in \mathcal{I}\}$ is called the *vector codebook* and has size N, meaning that it has N distinct elements, each being a vector in $\mathcal{R}^K$. That is, $\mathrm{y}_i \in \mathcal{R}^K$ for each $i \in \mathcal{I} = \{0, 1, 2, \cdots, N-1\}$. The *code rate* or, simply, the *rate* of a vector quantizer is $R = (\log_2 N)/K$, which measures the number of bits per vector component used to represent an input vector in $\mathcal{R}^K$ and gives an indication of the accuracy or precision that is achievable with a vector quantizer if the codebook is well-designed.

Associated with every vector quantizer of size N is a partition of $\mathcal{R}^K$ into N regions or *cells*, ω_i for $i \in \mathcal{I}$. The i-th cell is defined by

$$\omega_i = \{\mathrm{x} \in \mathcal{R}^K \mid Q(\mathrm{x}) = \mathrm{y}_i\}. \tag{4.1}$$

From the definition of the cells, it follows that

$$\bigcup_{i \in \mathcal{I}} \omega_i = \mathcal{R}^K \quad \text{and} \quad \omega_i \bigcap \omega_j = \emptyset \text{ for } i \neq j \tag{4.2}$$

so that the cells form a partition of $\mathcal{R}^K$.

A vector quantizer Q can be decomposed into two components, the vector *encoder* α and the vector *decoder* β, and expressed as a tuple (α, β). The encoder α is a many-to-one mapping from $\mathcal{R}^K$ to the index set $\mathcal{I}$, and the decoder β maps the index set $\mathcal{I}$ one-to-one onto the reconstruction set $\mathcal{Y}$, *i.e.*,

$$\alpha : \mathcal{R}^K \rightarrow \mathcal{I} \quad \text{and} \quad \beta : \mathcal{I} \rightarrow \mathcal{Y}.$$

It is important to note that a given partition $\Omega = \{\omega_i : i \in \mathcal{I}\}$ of the input vector space fully determines how the encoder will assign an index to a given input vector and, on the other hand, a given codebook $\mathcal{Y} = \{\mathbf{y}_i : i \in \mathcal{I}\}$ fully determines how the decoder will generate an output reconstruction vector from a given index. Therefore, the task of the encoder is to identify which of the N geometrically specified cells an input vector lies in, and that of the decoder is simply a table lookup procedure which is fully determined by specifying the codebook. Fundamentally, the encoder does not need to know the geometry of the partition to perform its operation. This is because for most vector quantizers of practical interest, the codebook provides sufficient information to characterize the partition. In such cases, the encoding operation can be performed by using the codebook as the data set which implicitly specifies the partition to reduce the computational requirement. More about this will be said in later subsections.

Based on the above description, the overall operation of VQ can be regarded as

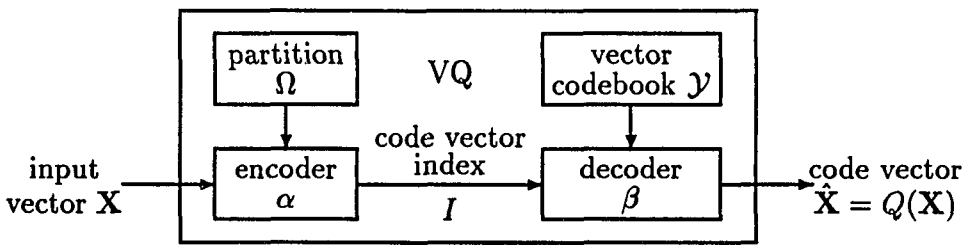


Figure 4.1: A vector quantizer as the cascade of an encoder and a decoder

the cascade of two operations:

$$Q(\mathbf{x}) = \beta \circ \alpha(\mathbf{x}) = \beta(\alpha(\mathbf{x})), \tag{4.3}$$

which is also illustrated in Figure 4.1. In the context of a digital communication system, the encoder of a vector quantizer selects an appropriately matching code vector $\mathbf{y}_i \in \mathcal{Y}$ to approximate an input vector $\mathbf{x}$. The index i of the selected code vector or, the *codeword*, is then transmitted (usually as a binary word) to the receiver where the decoder performs a table lookup procedure and generates the reconstruction $\mathbf{y}_i$, the quantized approximation of the original input vector. The objective of such a VQ system is, therefore, to produce the best possible reconstruction for a given codebook size N. To quantify this idea and to define the performance of a quantizer, the idea of a distortion measure is required.

4.2.2 Distortion measures

A distortion measure d is an assignment of a nonnegative cost $d(\mathbf{x}, \hat{\mathbf{x}})$ associated with quantizing any input vector $\mathbf{x}$ with a reconstruction vector $\hat{\mathbf{x}}$. Given such a measure, the performance of a K-dimensional VQ system can then be quantified by

the average distortion

$$D = \frac{1}{K} E\{d(\mathbf{X}, \hat{\mathbf{X}})\} = \frac{1}{K} \int_{\mathcal{R}^K} d(\mathbf{x}, \hat{\mathbf{x}}) f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} \tag{4.4}$$

between each input vector component and the corresponding reconstructed output. In the above equation, the integration is over the K-dimensional space, and $f_{\mathbf{X}}(\mathbf{x})$ is the joint probability density function (pdf) of the inputs $\mathbf{X}$. In practice, the average distortion D is usually evaluated by calculating the long term sample average

$$\hat{D} = \lim_{M \to \infty} \frac{1}{MK} \sum_{m=0}^{M-1} d(\mathbf{x}_m, \hat{\mathbf{x}}_m). \tag{4.5}$$

Generally, the performance of a VQ system is said to be good if the average distortion is small.

Ideally, for vector quantization, a distortion measure should be tractable to permit analysis and design, and computable so as to guide the actual encoding process for encoders which select the minimum distortion reconstruction. It should also be meaningful in real-life applications so that large or small average distortion values correspond respectively to bad and good performance.

The most convenient and widely used measure of distortion between an input vector $\mathbf{x}$ and a reconstruction vector $\hat{\mathbf{x}} = Q(\mathbf{x})$ is the *squared error* defined as

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \| \mathbf{x} - \hat{\mathbf{x}} \|^2 = (\mathbf{x} - \hat{\mathbf{x}})^t (\mathbf{x} - \hat{\mathbf{x}}) = \sum_{k=0}^{K-1} (x_k - \hat{x}_k)^2, \tag{4.6}$$

where t is the transpose operator and x_k denotes the k-th component of the vector $\mathbf{x}$. Based on this error measure, the *average squared-error distortion* is defined as

$$D = \frac{1}{K} E\{d(\mathbf{X}, \hat{\mathbf{X}})\} = \frac{1}{K} E\{\| \mathbf{X} - \hat{\mathbf{X}} \|^2\}. \tag{4.7}$$

The squared-error distortion measure is frequently interpreted as the power or energy of an error signal and, therefore, has some intuitive appeal in addition to being an analytically tractable measure for many purposes. Due to this interpretation, for the squared-error distortion, it is common practice to measure the performance of a VQ system using the *signal-to-quantization-noise ratio* (SQNR)

$$\text{SQNR} = 10\log_{10}\frac{E\{\| \mathbf{X} \|^2\}}{E\{d(\mathbf{X},\hat{\mathbf{X}})\}} \text{ dB}. \tag{4.8}$$

This corresponds to normalizing the average distortion by the average energy and plotting it on a logarithmic scale: large (small) SQNR corresponds to small (large) average distortion.

Numerous alternative distortion measures may also be defined for assessing the dissimilarity between the input and reconstruction vectors (see, for example, [42, 46]). Many of the measures of interest for VQ have the form

$$d(\mathbf{x},\hat{\mathbf{x}}) = \sum_{k=0}^{K-1} d_s(x_k,\hat{x}_k), \tag{4.9}$$

where $d_s(x_k,\hat{x}_k)$ is a scalar distortion measure as in one-dimensional quantization. A distortion measure having this additivity property with the same scalar distortion measure used for each component is particularly appropriate for signal compression where each vector component has the same physical meaning. Of particular interest is the case where the scalar distortion measure is given by $d_s(x,\hat{x}) = |x-\hat{x}|^n$ for some positive integer value n. In this case, the n-th root of $d(\mathbf{x},\hat{\mathbf{x}})$ is nothing but the l_n norm of the error vector $\mathbf{x}-\hat{\mathbf{x}}$. When $n = 2$, we obtain the squared-error distortion measure already discussed.

Another commonly used distortion measure is the *weighted squared error* defined

as

$$d(\mathbf{x}, \hat{\mathbf{x}}) = (\mathbf{x} - \hat{\mathbf{x}})^t \mathbf{W} (\mathbf{x} - \hat{\mathbf{x}}), \tag{4.10}$$

where $\mathbf{W}$ is a symmetric and positive definite weighting matrix. This measure allows unequal weights be introduced to different vector components to render certain contributions to the distortion more important than others. Moreover, if $\mathbf{W}$ is factorized as $\mathbf{P}^t\mathbf{P}$, then

$$d(\mathbf{x}, \hat{\mathbf{x}}) = (\mathbf{x} - \hat{\mathbf{x}})^t \mathbf{P}^t \mathbf{P} (\mathbf{x} - \hat{\mathbf{x}}) = (\mathbf{w} - \hat{\mathbf{w}})^t (\mathbf{w} - \hat{\mathbf{w}}), \tag{4.11}$$

where $\mathbf{w} = \mathbf{P}\mathbf{x}$ and $\hat{\mathbf{w}} = \mathbf{P}\hat{\mathbf{x}}$ are the transformations of $\mathbf{x}$ and $\hat{\mathbf{x}}$, respectively. Thus, the weighted squared error between the original vectors is equal to the squared error between the transformed vectors. This also indicates that, in general, vector quantization in a transform domain, which may be advantageous in some applications, can be regarded as one in the original domain with a different distortion measure.

A number of perceptually based distortion measures have also been used in speech and image compression [53, 46, 40]. These measures correlate well with subjective judgements and are particularly useful in low code rate applications [68]. Due to its popularity and computational simplicity, however, the squared-error distortion measure will be used throughout this dissertation unless specified otherwise. In most cases, generalizations of the analyses and designs presented hereafter to other distortion measures are straight forward, but may be computationally more involved.

4.2.3 Properties of optimal vector quantizers

Given a distortion measure d, for a fixed codebook size N and a fixed dimension K, a vector quantizer Q is said to be *optimal* if it minimizes the average distortion

$J(Q) = E\{d(\mathbf{X}, Q(\mathbf{X}))\}$ between an input vector and the corresponding quantized output. Thus, from the discussion in Subsection 4.2.1, it is clear that the principal goal in the design of a vector quantizer is to find a partition Ω or encoding rule, specifying the encoder α, and a codebook $\mathcal{Y}$, specifying the decoder β, that minimizes the average distortion $J(Q) = J(\alpha, \beta) = E\{d(\mathbf{X}, \beta(\alpha(\mathbf{X})))\}$ over all the vectors to be quantized. In the following, two necessary conditions for a vector quantizer to be optimal [66, 73] are described pertaining to the encoder and the decoder, respectively. Their proofs can be found in [40].

- *Nearest Neighbor Condition.* For a given codebook $\mathcal{Y} = \{\mathbf{y}_i : i \in \mathcal{I}\}$, the optimal partition cells satisfy

$$\omega_i = \{\mathbf{x} \mid d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j) \; \forall \, j \in \mathcal{I}\} \tag{4.12}$$

  and, for every input vector $\mathbf{x}$, the quantizer selects the code vector $\mathbf{y}$ as reconstruction yielding the minimum distortion, *i.e.*,

$$Q(\mathbf{x}) = \arg \min_{\mathbf{y} \in \mathcal{Y}} d(\mathbf{x}, \mathbf{y}). \tag{4.13}$$

In other words, given a codebook which fully specifies the decoder β, the optimal encoder is the nearest neighbor mapping

$$\alpha(\mathbf{x}) = \arg \min_{i \in \mathcal{I}} d(\mathbf{x}, \mathbf{y}_i) \tag{4.14}$$

that results in the minimum distortion

$$d(\mathbf{x}, Q(\mathbf{x})) = \min_{\mathbf{y} \in \mathcal{Y}} d(\mathbf{x}, \mathbf{y}) = \min_{i \in \mathcal{I}} d(\mathbf{x}, \mathbf{y}_i). \tag{4.15}$$

From the nearest neighbor condition, it becomes clear that, by using the given codebook and the nearest neighbor mapping rule, the encoder of a VQ system can

perform its operation without explicit knowledge about the geometry of the partition cells which may be quite complex, especially when the vector dimension K is large. Just as the nearest neighbor mapping rule optimizes the encoder of a VQ system for a given decoder, the decoder can also be optimized for a given encoder with the following condition.

- *Centroid Condition.* For a given partition $\Omega = \{\omega_i : i \in \mathcal{I}\}$, the optimal code vectors satisfy

$$\mathbf{y}_i = \text{centroid}(\omega_i) = \arg\min_{\mathbf{y}} E\{d(\mathbf{X}, \mathbf{y}) | \mathbf{X} \in \omega_i\}. \tag{4.16}$$

In other words, given an encoder α that maps input vectors $\mathbf{X}$ into different codewords in $\mathcal{I}$, the optimal decoder β is the mapping that assigns to each codeword i the centroid of all input vectors encoded into i:

$$\beta(i) = \arg\min_{\mathbf{y}} E\{d(\mathbf{X}, \mathbf{y}) | \alpha(\mathbf{X}) = i\}, \tag{4.17}$$

i.e., $\beta(i)$ is the vector minimizing the conditional average distortion over all the input vectors mapped into i.

In general, for an arbitrary random sequence of input vectors and distortion measure, it is quite difficult to minimize the conditional average distortion expressed in Equation (4.16) or (4.17). For the squared-error distortion measure defined in Subsection 4.2.2, however, it can be easily derived that the centroid of a set ω is simply the ordinary Euclidean centroid, *i.e.*,

$$\text{centroid}(\omega) = E\{\mathbf{X} | \mathbf{X} \in \omega\}. \tag{4.18}$$

In practice, this centroid is usually calculated by taking the arithmetic average of the vectors lying in ω

$$\frac{1}{|\omega|}\sum_{\mathbf{x}_m\in\omega}\mathbf{x}_m, \tag{4.19}$$

where $|\omega|$ stands for the cardinality of the set ω.

No general sufficient conditions for a VQ system to be optimal have been developed. Both the nearest neighbor condition and the centroid condition are only necessary conditions for optimality in the sense that they do not sufficiently guarantee that the globally minimum average distortion will be achieved. It can be shown that, though, under some mild restrictions, a VQ system satisfying the two necessary conditions can indeed achieve local optimality [41]. Readers are referred to [43] for examples of the insufficiency of the two conditions.

From Equation (4.4), it can be derived that for a vector quantizer $Q = (\alpha, \beta)$ with partition $\Omega = \{\omega_i : i \in \mathcal{I}\}$ of the input vector space $\mathcal{R}^K$ and codebook $\mathcal{Y} = \{\mathbf{y}_i : i \in \mathcal{I}\}$, the average distortion has the following expression:

$$D = \sum_{i\in\mathcal{I}} P_i D_i, \tag{4.20a}$$

where, P_i is the probability that an input vector lies in the partition cell ω_i, *i.e.*,

$$P_i = P[\mathbf{X}\in\omega_i] = P[\mathbf{X} \mid \alpha(\mathbf{X}) = i] = P[\mathbf{X} \mid Q(\mathbf{X}) = \mathbf{y}_i], \tag{4.20b}$$

and D_i is the conditional average distortion over all the inputs falling in ω_i, expressed as

$$D_i = \frac{1}{K}E\{d(\mathbf{X},\mathbf{y}_i)|\mathbf{X}\in\omega_i\} = \frac{1}{K}\int_{\omega_i} d(\mathbf{x},\mathbf{y}_i) f_{\mathbf{X}|\mathbf{X}\in\omega_i}(\mathbf{x})d\mathbf{x}, \tag{4.20c}$$

with $f_{\mathbf{X}|\mathbf{X}\in\omega_i}(\mathbf{x})$ being the conditional joint pdf of $\mathbf{X}$ given that $\mathbf{X}\in\omega_i$. Clearly, the average distortion is a function of both the partition Ω and the codebook $\mathcal{Y}$. The

significance of the above two necessary conditions for optimality lies in that, based on these conditions, the joint optimization problem of minimizing the functional $J(Q)$ with respect to both Ω and $\mathcal{Y}$ can be decomposed into two optimization subproblems: one minimizing $J(\alpha, \beta)$ with respect to Ω (specifying the encoder α) and the other to $\mathcal{Y}$ (specifying the decoder β). The two subproblems are interdependent: the optimization of the encoder α depends on that of the decoder β, and vice versa. Therefore, an iterative procedure can be used to exploit the two conditions in the attempt to minimize the average distortion. In such a procedure, the two coding units α and β are alternatively improved based on the last improved version of the other until some termination criterion is met. The design of vector quantizers based on this idea will be described in the next section.

4.3 Design of Vector Quantizers

It was pointed out earlier that, in the design of a VQ system, one would like to minimize the mathematical expectation expressed in Equation (4.4). This mathematical expectation is useful for developing information theoretical performance bounds, but is often not useful to calculate in practice since the required probability density function $f_{\mathbf{X}}(\mathbf{x})$ is usually unknown. Hence a pragmatic approach to the design is to take a long sequence of training data, estimate the "true" but unknown expected distortion by calculating the sample average in the expression (4.5), and attempt to design a system that minimizes the sample average distortion for the training sequence. If the training sequence is sufficiently long and the *source* producing the training sequence is *asymptotically mean stationary* [45], then when the system is used to quantize future data from the same source, the performance of the system

on the new data will be close to that achieved on the training data [59, 103]. Many processes of interests, such as speech signals and image data sequences, can be well modeled as outcomes of an asymptotically mean stationary source. Therefore, in practice, a VQ system is usually designed to minimize the sample average distortion for a long training sequence, and then used to quantize other sequences produced from the same source or, in other words, those sequences that are well represented by the training sequence.

4.3.1 The generalized Lloyd algorithm

As mentioned in the last section, the nearest neighbor condition and the centroid condition for optimality provide a basis for designing vector quantizers through an iterative descent procedure. Starting with a pre-determined codebook, in the procedure, each iteration finds the optimal (nearest neighbor) partition corresponding to the old codebook, and then constructs a new codebook which is optimal for that partition. That is, we begin with an initial quantizer $(\alpha^{(0)}, \beta^{(0)})$ and repeatedly apply a transformation

$$(\alpha^{(l+1)}, \beta^{(l+1)}) = T(\alpha^{(l)}, \beta^{(l)}),$$

where $\alpha^{(l+1)}$ is first chosen with $\beta^{(l)}$ being fixed to minimize $J(\alpha^{(l+1)}, \beta^{(l)})$, and then, for fixed $\alpha^{(l+1)}$, $\beta^{(l+1)}$ is chosen to minimize $J(\alpha^{(l+1)}, \beta^{(l+1)})$. This procedure guarantees that the average distortion $J^{(l)} = J(\alpha^{(l)}, \beta^{(l)})$ is reduced or at least left unchanged at each iteration, and thereby the VQ system performance is improved continually from one iteration to the next. Since $J(\alpha, \beta)$ is bound below by zero, the sequence of real numbers $J^{(l)}$ is guaranteed to converge; and it has been shown that

the convergence of the quantizers $(\alpha^{(l)}, \beta^{(l)})$ themselves is also guaranteed [103]. The iterative algorithm can be described in more detail as follows:

Step 1: *Initialization.* Set $l = 0$ and $\hat{D}^{(0)}$ to a sufficiently large value; and determine an initial codebook $\mathcal{Y}^{(0)} = \{\mathbf{y}_i^{(0)} : i \in \mathcal{I}\}$ using an appropriate method.

Step 2: *Classification.* Classify the vectors in the training sequence $\mathcal{X} = \{\mathbf{x}_m : 0 \leq m \leq M-1\}$ into clusters $\Omega = \{\omega_i : i \in \mathcal{I}\}$ using the nearest neighbor condition:

$$\omega_i = \{\mathbf{x}_m \in \mathcal{X} \mid d(\mathbf{x}, \mathbf{y}_i^{(l)}) \leq d(\mathbf{x}, \mathbf{y}_j^{(l)}) \; \forall \, j \in \mathcal{I}\}.$$

Step 3: *Construction of New Codebook.* Calculate the centroids of the clusters Ω just found to obtain a new codebook, *i.e.*, $\mathcal{Y}^{(l+1)} = \{\text{centroid}(\omega_i) : i \in \mathcal{I}\}$.

Step 4: *Termination Test.* Calculate the average distortion

$$\hat{D}^{(l+1)} = \frac{1}{MK} \sum_{i \in \mathcal{I}} \sum_{\mathbf{x}_m \in \omega_i} d(\mathbf{x}_m, \mathbf{y}_i^{(l+1)}).$$

If the decrease in the overall distortion $\hat{D}^{(l+1)}$ relative to $\hat{D}^{(l)}$ is below a certain threshold, stop and output the new codebook $\mathcal{Y}^{(l+1)}$; otherwise set $l + 1 \rightarrow l$ and go to Step 2.

It should be mentioned that in the execution of Step 2 of the above algorithm, some clusters may turn out to be empty or nearly empty. In such a case, the calculation of centroids in Step 3 for those empty clusters is meaningless. Also, those clusters that are nearly empty do not make much sense in the classification of the training vectors. For both the cases, therefore, some alternate rule to assign new code vectors is needed in order to optimize the system performance. A variety of

heuristic solutions have been proposed for handling the problem of empty and nearly empty clusters. Readers are referred to [40] for more details about these heuristic techniques.

The algorithm described above, called the *generalized Lloyd algorithm* or simply GL algorithm, was originally developed by Lloyd for the scalar quantization problem [66]. It was then generalized to vector quantizer designs by Linde, Buzo, and Gray [65], and so is sometimes referred to as the *LBG algorithm* in the vector quantization literature. It is also known as the *k-means algorithm* after MacGueen [69] in the pattern recognition literature. There is no guarantee that the iterative algorithm will result in a globally optimized quantizer. However, it can be shown that subject to some mathematical conditions the algorithm does converge to a local optimum [65, 40]. In an attempt to achieve global optimality, therefore, it is often usual to repeat the above algorithm for several different initial codebooks and then choose the codebook that results in the minimum average distortion. Different methods for codebook initialization are discussed in the next subsection.

A number of other iterative algorithms have also been proposed for designing vector quantizers. These algorithms include the *Kohonen self-organization feature maps* [62], which employs a neural network clustering technique, and *simulated annealing* [60] where some amount of noise is introduced into each iteration and a stochastic optimization technique is used in an attempt to avoid local optimum.

4.3.2 Determination of initial codebook

The problem of obtaining an initial codebook plays an important role in the design of VQ systems. It affects the speed of convergence of the GL algorithm as well

as the final performance of the resulting system. In fact, if the initial codebook is good enough, it may not be even worth the effort to improve it further through the iterative algorithm. A variety of techniques for codebook initialization has been proposed [40]. Most of these techniques are themselves vector quantizer design algorithms involving some kind of iterative procedure. Basically, the techniques follow one of two general approaches: start with some simple codebook of the desired size or start with a simple codebook of small size and recursively construct larger ones. In the following, a summary of several commonly used techniques is presented.

Random initialization. This is the simplest approach towards codebook initialization. In this type of schemes, an initial codebook is formed by using random numbers generated based on the estimated statistics of the data to be quantized [69]. Alternatively, the codebook can formed by using the first N vectors in the training sequence. If the data to be quantized is highly correlated, it is likely that a better initial codebook can be constructed by selecting some widely spaced or distant vectors from the training sequence [112].

Clustering initialization [30, 31]. This scheme begins with the entire training sequence of vectors and then recursively merges the vectors into clusters until the number of clusters reduces to the desired value. A codebook will then be formed by the centroids of the clusters. As results, we will have a partition of the training sequence into the correct number of clusters and have the optimal codebook for this partition. The partition, however, might not be a nearest neighbor partition and, therefore, the resulting codebook is in general not optimal. This scheme involves more computation than the random initialization schemes due to its recursive nature, but is faster than the full-search GL algorithm.

Product code initialization [44]. In this technique, a scalar quantizer is first designed for each of the K components of the vectors to be quantized. The vector quantization codebook is then formed by the Cartesian product of the K scalar quantization codebooks. For example, suppose a K-dimensional VQ codebook of size $N = 2^{KR}$ is to be designed. Then, one can first design K scalar quantization codebooks, $Y_k = \{y_{k,i} : 0 \leq i \leq 2^R - 1\}$, $0 \leq k \leq K-1$, with each pertaining to a specific vector component, and subsequently construct the VQ codebook as follows

$$\mathcal{Y} = \{(y_0, y_1, \cdots, y_{K-1}) : y_k \in Y_k, 0 \leq k \leq K-1\}.$$

Splitting initialzation. Instead of constructing a higher dimensional codebook from lower dimensional codebooks like in the product code scheme, one can construct a fixed dimensional codebook with larger size from a codebook with smaller size using a splitting technique [65, 17]. In the technique, the optimal rate 0 code vector, *i.e.*, the centroid of the entire training sequence, is first calculated. This single code vector is then split into two vectors by slight perturbation on the one already found. The original code vector is usually adopted as a member of the new pair to ensure that the average distortion will not increase. The new pair is then used as the initial code vectors for an iterative improvement algorithm, *e.g.*, the GL algorithm, to construct a good codebook with size 2. The improvement continues in this manner in stages, using the final code vectors of one stage to form an initial codebook for the next by splitting, until the codebook with desired size is obtained. It should be mentioned that this algorithm provides a complete design procedure from scratch on a training sequence and will later be seen to suggest a vector analog to successive approximation quantizers which, in turn, is one of the main topics of this dissertation.

4.3.3 Computational and memory requirements

After running the GL algorithm on an initial codebook, the resulting codebook can then be used to quantize each input vector. The quantization is performed by computing the distortion between the input and each of the code vectors, and then choosing the code vector with the minimum distortion as the reconstruction of the input vector. This type of quantization is referred to as *full search* since all code vectors are tested for quantizing each input vector. For a K-dimensional quantizer with codebook size N, the number of distortion computations needed to quantize a single input vector is N. In the cases where the squared-error distortion measure is used, a rough total of K multiply-and-add operations are required to perform each distortion computation. Therefore, the computational requirement for quantizing each input vector can be approximated as $KN = K2^{RK}$, which grows exponentially with the number of dimensions K and the code rate R. By assuming one memory location is required to store a vector component, it can be easily seen that the requirement to store a codebook is also $KN = K2^{RK}$.

While the computational and memory requirements associated with the quantization process itself should be emphasized in the characterization of a VQ system, it is also important to examine the requirements associated with the design of the system. To design a K-dimensional vector quantizer with codebook size N using the GL algorithm based on a training sequence of M vectors, it can be shown that the computational requirement for training is $KNML = K2^{RK}ML$, where L is the number of iterations, and the requirement to store the codebook as well as the training vectors is $K(N + M)$.

From the above discussion, it can be clearly seen that the direct use of the

full search vector quantization suffers from a serious complexity barrier, especially the computational requirement. This barrier severely limits the application of the full search VQ as a complete and self-contained compression technique to rather modest vector dimensions and code rates for practical problems. While the code rate needs to be maintained at some reasonable level in order for such a quantizer to achieve a desired compression performance, reducing the vector dimension often sacrifices the possibility of effectively exploiting the statistical dependency existing in the samples or parameters of the signal to be compressed. In response to the above limitation, a number of promising techniques have been proposed over the years which substantially reduce the computational requirement at the cost of a relatively small loss in performance and/or increase in memory requirement. These techniques will be described in the next section.

4.4 Variations of Vector Quantization Techniques

Unlike the full search VQ, most other VQ techniques apply some constraints to the structure of the codebook such that the number of code vector searches can be greatly reduced for quantizing each input vector. These techniques generally compromise the performance achievable with the full search VQ, but often provide very favorable tradeoffs between performance and complexity. As a result, they can be used to design quantizers for larger dimensions and higher rates and, thereby, achieve quality that is simply impossible for the full search VQ; in this sense, no performance is lost at all. In the following, one of these techniques, the *tree-structured vector quantization* (TSVQ), which forms the basis for the work reported in this part of my dissertation, is described in detail, and several others that are widely used in practice

are summarized.

4.4.1 Tree-structured vector quantization

The tree-structured vector quantization technique was first proposed in [17] and is closely related to the splitting algorithm for codebook initialization described in the last section. In this technique, the code vector search is performed in stages and, in each stage, a substantial subset of candidate code vectors is eliminated from consideration by a relatively small number of operations, which thereby significantly reduces the computational requirement incurred in quantizing each input vector. To describe the technique, let us first look at the design procedure of a binary TSVQ system.

Suppose a good codebook $\mathcal{Y}^*$ consisting of two code vectors, $\mathbf{y}_0$ and $\mathbf{y}_1$, has been constructed based on a training sequence of vectors using, for example, the GL algorithm. From $\mathcal{Y}^*$, an initial codebook of size 4 can be obtained by splitting $\mathbf{y}_0$ and $\mathbf{y}_1$. Instead of running a full search VQ design on the 4 member codebook, however, we divide the training vectors into two clusters, ω_0 and ω_1, one being the nearest neighbor cluster of $\mathbf{y}_0$ and the other of $\mathbf{y}_1$. For each of these clusters, we then find a good codebook of size 2 using the GL algorithm. As results, four code vectors, two in the codebook $\mathcal{Y}_0$ for the cluster ω_0 and the other two in $\mathcal{Y}_1$ for ω_1, are obtained. Each of these 4 vectors is then split again into two initial vectors and the above process is repeated, and so on, until some desired number $N = 2^\mu$ of code vectors are obtained.

The above described design procedure of a VQ system can be depicted as a binary tree of depth μ shown in Figure 4.2 where $\mu = 3$. The tree consists of a total

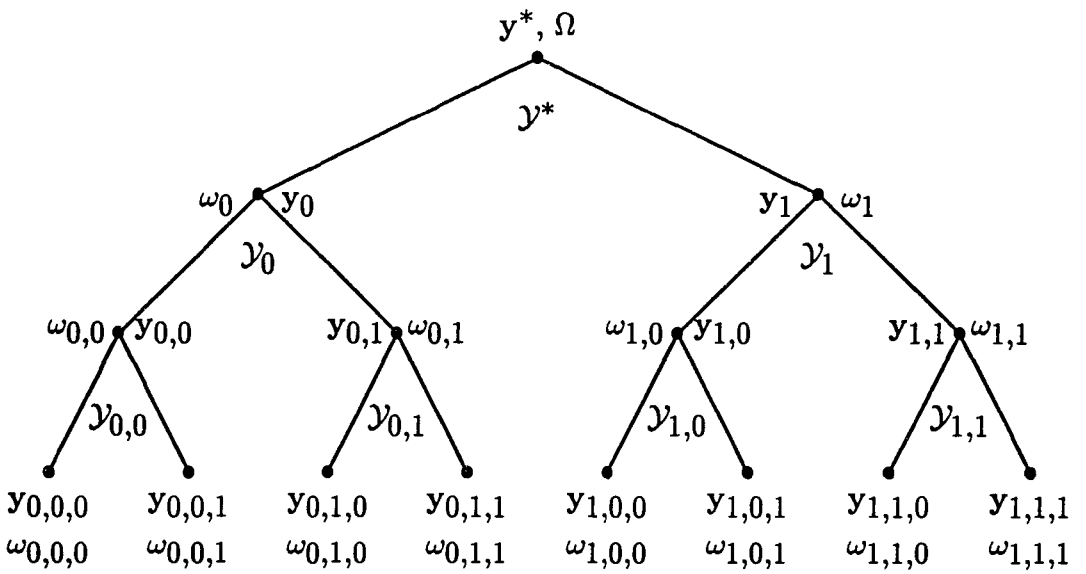


Figure 4.2: A binary tree-structured vector quantization system

of $2^{\mu+1} - 1$ nodes; and associated with each node is a vector or, equivalently, the nearest neighbor (sub)cluster of the vector. This tree is *uniform* since all of its internal nodes have the same number of immediate descendent nodes. It is also *balanced* in the sense that the depths of all terminal nodes are identical. For a system so designed, the codebook will be made up of the N vectors associated with the terminal nodes, while the vectors associated with the internal nodes will serve as *intermediate code vectors* for inputs classified into their corresponding nearest neighbor (sub)clusters. In Figure 4.2, for example, the codebook of the system is formed by the 8 vectors $\mathbf{y}_{i,j,k}$, $0 \leq i,j,k \leq 1$; meanwhile, $\mathbf{y}_{0,0}$ and $\mathbf{y}_{0,1}$ are the intermediate code vectors for inputs classified into $\omega_{0,0}$ and $\omega_{0,1}$, respectively, and they together form the binary intermediate codebook $\mathcal{Y}_0$ for the cluster ω_0, of which $\omega_{0,0}$ and $\omega_{0,1}$ are the two subclusters.

When an input vector $\mathbf{x}$ is quantized by the TSVQ system in Figure 4.2, its code vector is not selected by an ordinary full search. Instead, the stage 1 intermediate codebook $\mathcal{Y}^* = \{\mathbf{y}_0, \mathbf{y}_1\}$ is first searched, and the vector between $\mathbf{y}_0$ and $\mathbf{y}_1$ that yields smaller distortion is found. If the binary index of this vector is i, then i will be transmitted or stored as the first bit of codeword for $\mathbf{x}$. Given that $\mathbf{y}_i$ has been selected as the first stage intermediate code vector for $\mathbf{x}$, the corresponding stage 2 intermediate codebook $\mathcal{Y}_i = \{\mathbf{y}_{i,0}, \mathbf{y}_{i,1}\}$ is then searched, from which the next intermediate code vector and hence the second bit of codeword for $\mathbf{x}$ can be determined. This successive binary search process proceeds until, at stage μ, one of the system code vectors has been selected. In short, $\mathbf{x}$ is quantized by traversing the tree associated with the TSVQ system from the root node to one of the terminal nodes along a path that gives the minimum distortion at each node in the path. The code vector for $\mathbf{x}$ is the vector associated with the terminal node and the codeword is the concatenation of indices of the nodes along the path.

From the above description, it can be seen that a total number of $\mu = \log_2 N$ binary searches are needed for a K-dimensional binary TSVQ system of codebook size N to quantize an input vector. The corresponding computational requirement can therefore be approximated as $2K\mu = 2K \log_2 N = 2K^2 R$, which is only linearly proportional to the code rate R and the square of vector dimension K. Moreover, the memory requirement for the TSVQ system can be shown as $2K(N-1)$, about double that of a full search VQ system. This is because in addition to storing the N code vectors, the intermediate code vectors associated with the internal nodes of the tree must also be stored. While the computational requirement of a TSVQ is reduced by a factor of $2^{\mu-1}/\mu$ compared to a full search VQ of the same dimension and code

rate, its performance is also degraded since the partition of the input vector space no longer satisfies the nearest neighbor condition and the code vector for each input is not selected through an exhaustive search of the codebook. The degradation, however, is usually quite modest, but often not negligible. A comparison of quantization performance between TSVQ and full search VQ in terms of SQNR[1] for fixed vector dimension but different code rates, *i.e.*, different vector codebook sizes, is given in Figure 4.3. The SQNR values shown in the figure were measured on a sequence of 200,000 4-dimensional training vectors containing samples of a *first-order Gauss-Markov source*[2] with regression coefficient $\rho = 0.9$. Clearly, from the figure, it can be seen that the performance gap between TSVQ and full search VQ is small, but increases with code rate.

Since the tree-structured codebook only affects the search strategy of the encoding operation, a TSVQ system, in general, does not need the intermediate code vectors for decoding and is, in this respect, identical to a full search VQ system. It should be, however, pointed out that, due to the successive approximation nature,

[1]The SQNR here is evaluated a little differently from that expressed in Equation (4.8) by using the formula

$$\text{SQNR} = 10\log_{10}(E\{d(\mathbf{X},\mathbf{y}^*)\}/E\{d(\mathbf{X},\hat{\mathbf{X}})\}) \text{ dB},$$

where $\mathbf{y}^*$ is the centroid of the entire input vector space. This definition of SQNR can also be interpreted as the ratio of zero-rate and nonzero-rate quantization noises and, unless otherwise specified, will be used in the remainder of this dissertation to present experimental results of various VQ techniques.

[2]A first-order Gauss-Markov source $\{X_m\}$ is defined by

$$X_{m+1} = \rho X_m + W_m$$

where the regression coefficient ρ has magnitude less than 1 and $\{W_m\}$ are zero mean, independent, and identically distributed Gaussian random variables.

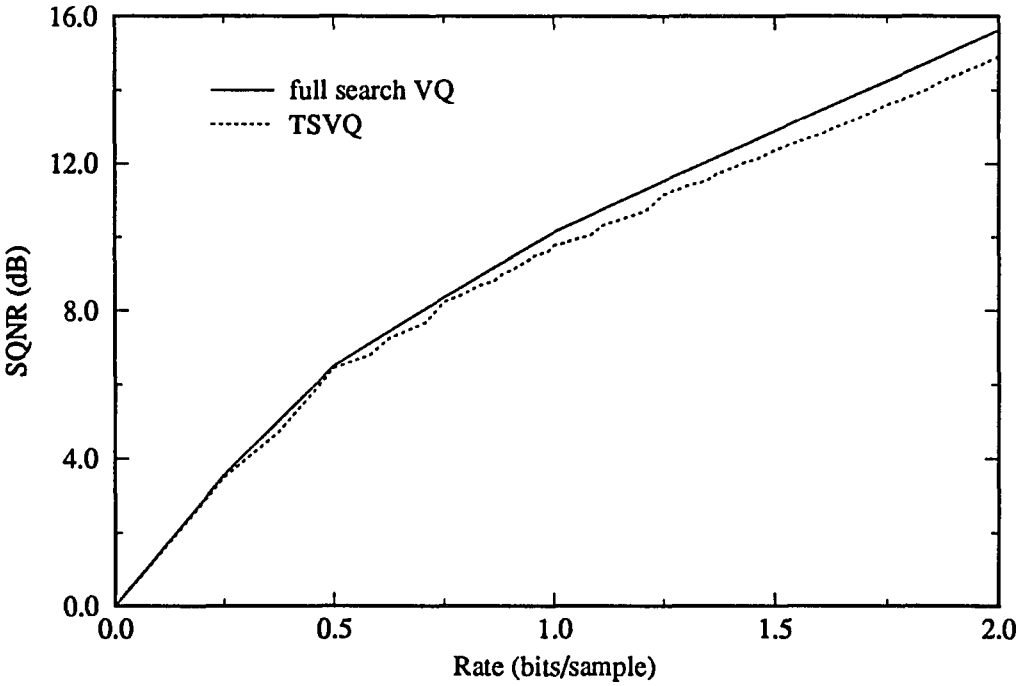


Figure 4.3: SQNR versus code rate of TSVQ and full search VQ for a Gauss-Markov source with $\rho = 0.9$

TSVQ technique is particularly suitable for progressive transmission applications. In such applications, a TSVQ decoder does use the intermediate code vectors. There, when an input vector is quantized, the encoder (*i.e.*, the transmitter) sends a bit of codeword to the decoder (*i.e.*, the receiver) as soon as the search of code vector passes through each stage of the tree. The decoder uses the indicated intermediate code vector as a temporary reconstruction and, as each new codeword bit arrives, the reconstruction is replaced with a better approximation using an intermediate code vector from a deeper level of the tree. Finally, when the last bit is received, a particular code vector is identified and then used as the final reconstruction.

Besides the design procedure described above, TSVQ systems can also be de-

signed using other techniques [44, 2, 37]. For example, one approach is to begin with a full search codebook and to design a tree structure into the codebook [44]. To accomplish this, the code vectors are first grouped into close disjoint pairs and the centroid of each pair is then associated to the immediate ancestor node of the pair. One then works backwards through the tree, always grouping close pairs, until the root node is reached.

Binary TSVQ is a special and the simplest case of TSVQ. In general, one can design a TSVQ system by dividing the cluster of training vectors associated with each tree node into more than two subclusters. Such a system requires more computation and less memory than a binary system, with some increase in performance. An example of this type of systems can be found in [116]. Moreover, unbalanced and/or nonuniform TSVQ systems can also be designed to improve the performance-complexity tradeoff. Such design techniques and improvements are the topics of next two chapters.

4.4.2 Other vector quantization techniques

Classified vector quantization. Classified VQ is similar to a one stage TSVQ where the criterion for selecting which of n branches to descend from the root node is based on an heuristic characteristic that the designer adopts to identify the mode of each input vector. Thus, instead of an intermediate codebook, an arbitrary classifier may be used to select a particular subset of the codebook to be searched. Many possibilities exist for the choice of classifier and each leads to a special VQ technique. For example, in image compression, the classifier can be an edge detector that identifies the presence or absence of edges in an image block and the direction and location of

edges if present [90].

Cascaded vector quantization [57, 100, 8, 35]. The basic idea of this technique is to divide the encoding task into successive stages. The first stage performs a relatively crude quantization of an input vector using a small codebook. Then, a second stage quantizer operates on the error vector between the original input and the quantized first stage output. The quantized error thus provides a second approximation to the original input, thereby leading to a more accurate reconstruction of the input. A third stage quantizer may also be used to quantize the second stage error to provide a further refinement, and so on. Clearly, a cascaded VQ system can be regarded as a special TSVQ system where only a single small codebook is used for each stage of the tree instead of a different codebook for each node of the stage. Such a system realizes the same computation reduction as a TSVQ system while reducing the memory requirement even below that of a full search VQ system. Moreover, cascaded VQ is also suitable for progressive transmission applications.

Product code techniques. Besides tree structure, a product code is another very useful structure for vector quantization. In such a VQ system, the codebook is a Cartesian product of several smaller codebooks, which may permit one to achieve good performance-complexity tradeoffs by encoding different aspects of an input vector separately for their different effects on performance and implementation. One example of product code techniques for VQ is the *mean-removed VQ* (MRVQ) [6, 7] where the sample mean and the mean-removed residual of a vector are quantized separately. This technique has been used for image compression in an attempt to improve the subjective reconstruction quality. Another example is the *gain-shape VQ* (GSVQ) [17, 102] where two different, but interdependent, codebooks are used

to quantize the gain and the shape of a vector separately, with the gain being defined as the energy of the vector while the shape as the original vector normalized by the gain. It can be shown that such a gain-shape separation of vectors is an optimal separation in the context of product codebook construction [40]. Most *transform VQ* techniques (see, for example, [3, 23, 4]) that take advantage of the energy compaction property of an orthogonal transformation are also special cases of product code techniques. In such a technique, an input vector first undergoes, for example, the Fourier transform, discrete cosine transform, or wavelet transform. The transform coefficients are then divided into several groups and each group is quantized using a different codebook. Moreover, two or more product code techniques can be combined to obtain a hybrid product code VQ method. One such example is given in [75] where MRVQ and GSVQ are combined together in the design of codebooks that are more robust against uncertainties in the statistical model of the signal to be compressed.

Recursive techniques. Recursion is yet another commonly used structure for vector quantization [59]. Such VQ techniques attempt to reduce the computational requirement by using short vector dimensions and incorporating memory into a VQ system. In these techniques, a codebook is selected from several alternatives for quantizing each input vector based on past inputs. The decoder therefore must be somehow informed which codebook is being used by the encoder in order to decode a channel codeword. Based on different schemes of informing the decoder, recursive VQ techniques can be subdivided into three classes: *predictive VQ* [27, 33], *adaptive VQ* [2, 27], and *finite-state VQ* [34].

Other vector quantization techniques include *constrained storage VQ* [18],

hierarchical and multiresolution VQ [38, 48, 16], *delayed decision VQ* [55, 110], *etc.* A state-of-the-art survey and thorough description of most VQ techniques can be found in [40]. Other surveys and discussions can also be found in [44, 68, 77, 26].

5. RATE-DISTORTION THEORY AND VARIABLE-RATE VECTOR QUANTIZATION

5.1 Introduction

5.1.1 Rate-distortion theory

Since the main objective in the design of a signal compression system is to minimize the average distortion for a desired code rate, it is natural and also important to ask whether there is a lower bound on the distortion and what the bound is, if any. By knowing such information, one can compare the compression performance of different systems and decide whether to search for other possibly more complex systems that might offer better performance. Around 1950, Shannon [105, 106, 107] and Kolmogorov [63] showed that if a distortion measure is defined appropriately, the lower bound does exist. The bound $D(R)$, known as the *distortion-rate function*, specifies the minimum possible distortion D attainable when a source is encoded at a code rate not exceeding R bits per sample; and the performance limit provided by $D(R)$ applies to all compression systems, including those using vector quantization. As a branch of information theory, *rate-distortion theory* deals with the determination of $D(R)$ without requiring the design of actual compression systems and has been well developed [36, 10]. In Section 5.2, some fundamental results from this theory

will be discussed. Later, we will see that these results can be used to guide the design of certain compression systems, as well as to compare their performance.

5.1.2 Variable-rate vector quantization

In most of the vector quantization techniques described so far, it is assumed that the code vector index is transmitted or stored as the codeword for an input vector, and the codewords for different inputs all have the same number of bits. Under this assumption, the problem of designing a VQ system is to find a set of reconstruction vectors, known as the vector codebook, that minimizes the average distortion between the vectors being quantized and their reconstructions, subject to a constraint on the *codebook size*, *i.e.*,

$$\min_{Q} \frac{1}{K} E\{d(\mathbf{X}, Q(\mathbf{X}))\} \quad \text{with the codebook size of } Q \leq N. \tag{5.21}$$

Using a system so designed, signals can be compressed at a fixed ratio and the compressed data can be transmitted over a fixed-rate channel without buffering. The fixed-rate VQ system, however, is not optimal in the rate-distortion sense since the minimization of the average distortion is for a fixed codebook size, instead of a desired code rate. However, it has been shown [36, 10, 85] that if the vector dimension K is sufficiently large, there exists a finite set of N reconstruction vectors with code rate $R = (\log_2 N)/K$, such that the average distortion D is arbitrarily close to the distortion-rate function $D(R)$, over all possible compression schemes with average code rate less than or equal to R. This provides a theoretical basis for the use of the fixed-rate or codebook size constrained vector quantizers: for sufficiently large vector dimensions, they are theoretically optimal. Unfortunately, practical schemes can not use arbitrarily large vector dimensions.

To achieve optimal performance in the rate-distortion sense, in general, variable-rate quantization is required. In image compression applications, for example, fixed-rate VQ methods often suffer from blockiness distortion, particularly at the edges, due to a limited number of code vectors [77]. By using a variable-rate method, one can devote more code vectors or, equivalently, more codeword bits, to the edge regions of an image while allocating fewer bits to the low activity regions and the background. With such a design, it is reasonable to expect that a better reconstruction quality can be achieved without increase of the average code rate. Systems so designed, on the other hand, are usually more complex to implement than fixed-rate systems. This additional complexity is even more significant if the compressed data is to be transmitted over a fixed-rate channel due to the necessity of buffering (and the inherent problems of buffer overflow and underflow) as well as that of channel error propagation. In general, however, the performance gains attained by a variable-rate system outweigh the additional complexity. Also, there are applications, such as storage and packet switched communication networks, for which variable-rate transmission is naturally suited. Consequently, several research works have been devoted to developing variable-rate VQ techniques. In the following, we present a precise definition of variable-rate vector quantization.

Typically, in a K-dimensional variable-rate VQ system, each input vector $\mathbf{x} \in \mathcal{R}^K$ to be quantized is first mapped into a codeword c in the *channel codebook* $\mathcal{C} = \{c_i : i \in \mathcal{I}\}$. The codeword c is then sent through a channel or stored in a medium, which is assumed to be noiseless, and later mapped into a reconstruction vector $\mathbf{y}$ in the vector codebook $\mathcal{Y} = \{\mathbf{y}_i : i \in \mathcal{I}\}$. In general, the vector codebook $\mathcal{Y}$ does not need to be finite or even countable. The channel codebook $\mathcal{C}$, however,

is finite or countable and contains codewords of different lengths in terms of number of bits. A vector quantizer Q with such vector and channel codebooks can be in general decomposed into a cascade of four components: the vector encoder α, the variable-length encoder γ, the corresponding decoder γ^{-1}, and the vector decoder β, *i.e.*,

$$Q = \beta \circ \alpha = \beta \circ \gamma^{-1} \circ \gamma \circ \alpha, \tag{5.22}$$

as shown in Figure 5.1. The first component α, the same as that in a fixed-rate system, is a many-to-one mapping from the input vector space $\mathcal{R}^K$ to the code vector index set $\mathcal{I}$ and is usually information lossy. The second part $\gamma : \mathcal{I} \rightarrow \mathcal{C}$ maps a code vector index into a channel codeword, and is one-to-one and information lossless. The mapping γ^{-1} is the inverse of γ, and maps each channel codeword back into its index. Finally, the mapping β outputs a reconstruction vector in $\mathcal{Y}$ for an index. Notice that in order to be information lossless, the variable-length mapping γ needs not only to be invertible, but also *uniquely decodable.* Consequently, the lengths of codewords in $\mathcal{C}$ must satisfy the *Kraft Inequality* [1]:

$$\sum_{i \in \mathcal{I}} 2^{-|c_i|} \leq 1, \tag{5.23}$$

where $|c_i|$ denotes the length of the codeword c_i in number of bits. As will be seen later, this places a constraint on the minimum achievable average codeword length or, equivalently, the code rate.

With the above definition, the problem of designing a variable-rate VQ system is then to find a vector codebook that minimizes the average distortion between the vectors being quantized and their reconstructions, subject to a constraint on the

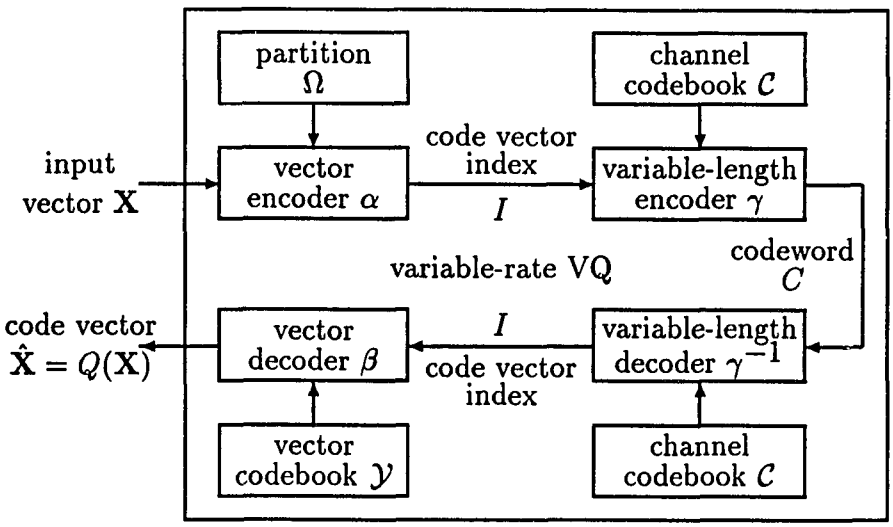


Figure 5.1: A variable-rate vector quantizer as the cascade of four components

average code rate, *i.e.*,

$$\min_{Q} \frac{1}{K} E\{d(\mathbf{X}, Q(\mathbf{X}))\} \text{ with } \frac{1}{K} E\{|\gamma(\alpha(\mathbf{X}))|\} \leq R. \tag{5.24}$$

In Sections 5.3 and 5.4, several important variable-rate VQ techniques will be described; and in the next chapter, a new method for directly designing a variable-rate VQ system will be presented.

5.2 Fundamentals of Rate-Distortion Theory

Most results of rate-distortion theory were developed in a scalar formulation. However, the problem under consideration here is vector quantization and, as mentioned earlier, even in the extreme case of a memoryless source, vector quantization can still achieve better performance than scalar quantization. Therefore, it is more appropriate to investigate rate-distortion limits in a vector formulation. In such a

case, samples of a source are usually grouped into vectors to determine the conditions based on which those limits can be achieved.

5.2.1 Definition of $D(R)$

Suppose a sequence of vectors $\mathbf{X}$ is quantized by a K-dimensional vector quantizer $Q_K = \beta_K \circ \alpha_K$ with codebook $\mathcal{Y} = \{\mathbf{y}_i : i \in \mathcal{I}\}$. For the purpose of transmission or storage, the code vector indices $I = \alpha_K(\mathbf{X})$ can be encoded, by using an entropy code γ such as the Huffman code or arithmetic code, into another uniquely decodable bit stream of possibly lower rate than the stream of their original binary representations. If the indices are encoded independently, then the average code rate achieved by the entropy code is given by the zeroth-order entropy of the indices, defined as

$$H_K(I) = E\{|\gamma(\alpha_K(\mathbf{X}))|\} = -\sum_{i \in \mathcal{I}} P_i \log_2 P_i \text{ bits/vector}, \tag{5.25}$$

where $0 \log_2 0 \stackrel{\Delta}{=} 0$ and $P_i = P[\alpha_K(\mathbf{X}) = i]$, the same as in Equation (4.20), is the probability that an input vector is quantized into code vector $\mathbf{y}_i$. The zeroth-order entropy $H_K(I)$ is the minimum average code rate achievable by any lossless variable-length code if the indices are encoded independently [36]. Therefore, given a distortion measure d, the minimum average distortion between the vectors $\mathbf{X}$ and their reconstructions over all possible K-dimensional VQ for a given code rate R is

$$D_K(R) = \min_{Q_K} \frac{1}{K} E\{d(\mathbf{X}, Q_K(\mathbf{X}))\} \text{ with } \frac{1}{K} H_K(I) \leq R. \tag{5.26a}$$

$D_K(R)$ is referred to as the *Kth-order operational distortion-rate function.* Based on $D_K(R)$, the distortion-rate function $D(R)$ is then defined as

$$D(R) = \lim_{K \to \infty} D_K(R). \tag{5.26b}$$

According to this definition, the distortion-rate function $D(R)$ is not merely a lower bound for any quantizer, but can also be approached arbitrarily close, in principle, by using a vector quantizer of sufficiently high dimension, which may not be true for a scalar quantizer.

5.2.2 Properties of $D(R)$

The distortion-rate function $D(R)$ has a number of interesting properties [36, 10]. In the following, some of these properties are described.

- *Property 1.* $D(R)$ is a nonnegative, nonincreasing function of R, *i.e.*, $D(R_1) \geq D(R_2) \geq 0$ if $R_1 \leq R_2$.
- *Property 2.* $D(R)$ is a convex $\cup$ function, *i.e.*, $D(\eta R_1 + (1-\eta)R_2) \leq \eta D(R_1) + (1-\eta)D(R_2)$ for any $\eta \in [0,1]$.
- *Property 3.* If two quantizers Q' and Q'' both achieve the point $(R, D(R))$, then so does the quantizer $Q = \eta Q' + (1-\eta)Q''$ for any $\eta \in [0,1]$.

It should be pointed out that the Kth-order operational distortion-rate function $D_K(R)$ is also a nonincreasing function of R and possesses the third property described above. However, it is not necessarily convex or even continuous, particularly if the source under consideration is discrete-amplitude, such as a digital signal. An illustration of $D(R)$ and $D_K(R)$ is given in Figure 5.2. Also shown in the figure is the convex hull of $D_K(R)$ in dashed line.

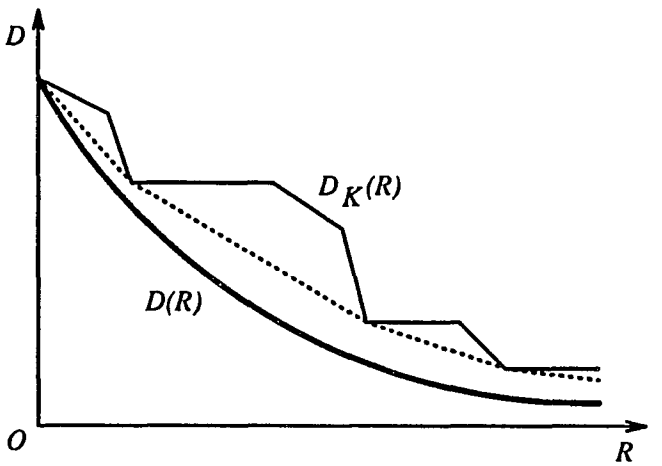


Figure 5.2: Illustration of $D(R)$ and $D_K(R)$ together with the convex hull of $D_K(R)$ in dashed line

5.2.3 Evaluation of $D(R)$

While defining the distortion-rate function $D(R)$ is straightforward, the analytical evaluation of such a function is often very difficult or even impossible. For the squared-error distortion measure and a first-order Gauss-Markov source with variance σ^2 and regression coefficient ρ, it can be derived that for $R \geq \log_2(1+\rho)$ [10],

$$D(R) = (1-\rho^2)2^{-2R}\sigma^2 \tag{5.27}$$

and the corresponding signal-to-quantization-noise ratio

$$\text{SQNR} = 20R\log_{10}2 - 10\log_{10}(1-\rho^2) \cong 6.02R - 10\log_{10}(1-\rho^2) \text{ dB}. \tag{5.28}$$

In most other cases, however, computational methods may have to be used to calculate $D(R)$.

Since $D(R)$ is a convex function, Blahut proposed an elegant convex programming algorithm to compute $D(R)$ [13]. In the Blahut algorithm and other more

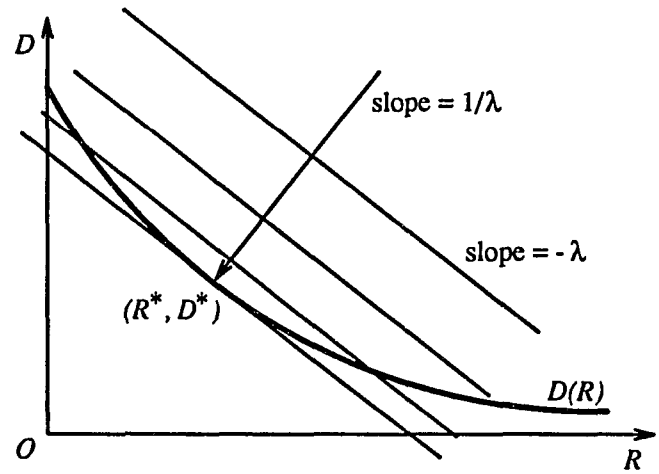


Figure 5.3: Convergence of a sequence of pairs (R, D) in the calculation of $D(R)$ using the Blahut algorithm

general convex programming algorithms, the constrained minimization problem of (5.26) is reformulated into an unconstrained minimization of the Lagrangian functional

$$J_\lambda(Q) = E\{d(\mathbf{X}, Q(\mathbf{X}))\} + \lambda E\{|\gamma(\alpha(\mathbf{X}))|\}. \tag{5.29}$$

With this reformulation, the rate R is no longer an input to the minimization problem. Instead, for each value of the Lagrange multiplier λ, a sequence of rate-distortion pairs (R, D) is calculated from the minimization process which converges to a point (R^*, D^*) on the $D(R)$ curve. The convergence is monotonic in the (R, D) plane in the direction specified by λ that is perpendicular to the slope of $D(R)$ at the limiting point (R^*, D^*), as shown in Figure 5.3. Clearly, for a number of different choices of λ, different points on the $D(R)$ curve can be calculated by using the algorithms. In the next section, a variable-rate VQ technique will be described where the Blahut algorithm is utilized to design VQ systems for different desired code rates.

5.3 Entropy-Constrained Vector Quantization

5.3.1 Background

As indicated in Subsection 5.2.1, for VQ systems of a fixed vector dimension K and vector codebook size N, a lower average code rate can be achieved with the same average distortion by encoding the code vector index sequence using an entropy code rather than simply transmitting or storing the indices in their original binary representations. Entropy coding reduces the average code rate from $\log_2 N$ bits per vector to the index entropy $H_K(I)$ by exploiting the nonuniform probability distribution of the indices. With additional effort, the average code rate can be further reduced by taking into account the higher-order inter-dependencies of the indices. However, the code rate here is an average since schemes for achieving such lower rate generally have a variable-rate nature, and hence, an additional amount of complexity. If we are willing to deal with this additional complexity, then the natural VQ design problem is to find a set of code vectors that minimizes the average distortion between the vectors being quantized and their reconstructions, subject to a constraint on the *index entropy*, *i.e.*, achieves the distortion bound specified by the Kth-order operational distortion-rate function $D_K(R)$.

Expressions for the index entropy and distortion performance of the above defined *entropy-constrained vector quantization* (ECVQ) (including its special case, *entropy-constrained scalar quantization* (ECSQ)) typically also include the number of reconstructions N as a parameter. Ideally, we would like to find the optimum performance over all N. In many analytical approaches and all practical implementations, however, it is in fact necessary to limit N to some maximum value.

A number of investigations have been reported on determining the optimal ECSQ and ECVQ (for example, see [119, 11, 78, 12, 32, 123, 84, 21]). Among these efforts, Berger [11] described the necessary conditions for an optimal ECSQ under the squared-error distortion measure (and later for other measures [12]), and first presented the Lagrangian formulation, described in the last section, of the ECSQ system design problem. Based on the formulation, an iterative algorithm was also developed for computing the quantization thresholds of an ECSQ system. Farvardin and Modestino [32] then extended the Berger's necessary conditions for optimality to general distortion measures, and outlined two methods for ECSQ system design. As a generalization of one of these two methods, the first ECVQ system design algorithm was proposed by Chou *et al.* [21]. The algorithm also begins with a Lagrangian formulation and follows the guideline of the Blahut's algorithm, but in implementation is quite similar to the generalized Lloyd algorithm for codebook size constrained VQ system design. As results of running the algorithm, a set of variable-rate vector quantizers can be obtained of which the rate-distortion pairs (R, D) converge in the direction specified by the Lagrange multiplier λ. In this section, a detailed description of the algorithm is presented as it will be used as an important constituent part in one of our new algorithms discussed in the next chapter.

5.3.2 The ECVQ algorithm

Recall that the Kth-order operational distortion-rate function $D_K(R)$ defined in Equation (5.26a) is not necessarily convex or even continuous. Hence, the Blahut's algorithm and other convex programming algorithms can not be used to find $D_K(R)$. These algorithms, however, can be used to find the convex hull of $D_K(R)$ by mini-

mizing the functional

$$J_\lambda(Q_K) = E\{d(\mathbf{X}, Q_K(\mathbf{X}))\} + \lambda E\{|\gamma(\alpha(\mathbf{X}))|\} \tag{5.30}$$

with respect to $Q_K = \beta \circ \gamma^{-1} \circ \gamma \circ \alpha$. In practice, it is often possible to find a variable-rate quantizer Q_K from the minimization of $J_\lambda(Q_K)$ that achieves a point on $D_K(R)$ and also on its convex hull with average code rate (or distortion) sufficiently close to the desired average rate (or distortion).

Rewriting the quantizer Q_K as (α, γ, β) and reexpressing the functional $J_\lambda(Q_K)$ as

$$J_\lambda(\alpha, \gamma, \beta) = E\{d(\mathbf{X}, \beta(\alpha(\mathbf{X}))) + \lambda|\gamma(\alpha(\mathbf{X}))|\}, \tag{5.31}$$

then an iterative descent algorithm similar to the GL algorithm can be employed to find a quantizer (α, γ, β) that minimizes the functional $J_\lambda(\alpha, \gamma, \beta)$. Starting with an initial quantizer $(\alpha^{(0)}, \gamma^{(0)}, \beta^{(0)})$, a transformation

$$(\alpha^{(l+1)}, \gamma^{(l+1)}, \beta^{(l+1)}) = T(\alpha^{(l)}, \gamma^{(l)}, \beta^{(l)})$$

is repeatedly performed such that $J_\lambda(\alpha^{(l)}, \gamma^{(l)}, \beta^{(l)})$ is decreasing with l. The transformation T operates as follows.

Step 1: For fixed $\gamma^{(l)}$ and $\beta^{(l)}$, $\alpha^{(l+1)}$ is chosen to minimize $J_\lambda(\alpha^{(l+1)}, \gamma^{(l)}, \beta^{(l)})$. The determination of $\alpha^{(l+1)}$ is the same as the nearest neighbor classification in the GL algorithm except that the criterion for classifying nearest neighbors is changed from $d(\mathbf{x}, \beta(\alpha(\mathbf{x})))$ to $d(\mathbf{x}, \beta(\alpha(\mathbf{x}))) + \lambda|\gamma(\alpha(\mathbf{x}))|$.

Step 2: For fixed $\alpha^{(l+1)}$ and $\beta^{(l)}$, $\gamma^{(l+1)}$ is chosen to minimize $J_\lambda(\alpha^{(l+1)}, \gamma^{(l+1)}, \beta^{(l)})$. In this case,

$$J_\lambda(\alpha^{(l+1)}, \gamma^{(l+1)}, \beta^{(l)}) =$$

$$\sum_{i\in\mathcal{I}} P_i^{(l+1)} E\left\{d(\mathbf{X},\beta^{(l)}(i)) + \lambda|\gamma^{(l+1)}(i)| \;\middle|\; \alpha^{(l+1)}(\mathbf{X}) = i\right\},$$

where $P_i^{(l+1)} = P[\alpha^{(l+1)}(\mathbf{X}) = i]$. Clearly, a lossless variable-length code $\gamma^{(l+1)}$ that minimizes $J_\lambda(\alpha^{(l+1)},\gamma^{(l+1)},\beta^{(l)})$ is one that minimizes the expected codeword length

$$R = \sum_{i\in\mathcal{I}} P_i^{(l+1)}|\gamma^{(l+1)}(i)|.$$

Such a code has codeword lengths

$$|\gamma^{(l+1)}(i)| = -\log_2 P_i^{(l+1)} \;\forall\, i \in \mathcal{I}$$

with average code rate exactly equal to the index entropy [36], and can be realized by arithmetic coding [95, 56] with resulting code rate very close to the entropy.

Step 3: For fixed $\alpha^{(l+1)}$ and $\gamma^{(l+1)}$, $\beta^{(l+1)}$ is chosen to minimize $J_\lambda(\alpha^{(l+1)}, \gamma^{(l+1)},\beta^{(l+1)})$. This step is exactly the same as the vector codebook construction step of the GL algorithm.

The above described procedure guarantees that $J_\lambda^{(l)} = J_\lambda(\alpha^{(l)},\gamma^{(l)},\beta^{(l)})$ is nonincreasing with l. Following the same lines of argument for the GL algorithm, it appears that the convergences of $J_\lambda^{(l)}$ and the sequence of quantizers $(\alpha^{(l)},\gamma^{(l)},\beta^{(l)})$ are also guaranteed. Like the GL algorithm, there is no guarantee that this algorithm will result in a globally optimized variable-rate quantizer. As the result of running the algorithm, a quantizer $Q_K = (\alpha,\gamma,\beta)$ is obtained which locally minimizes the functional

$$(1/K)J_\lambda(Q_K) = D(Q_K) + \lambda R(Q_K),$$

with

$$D(Q_K) = (1/K)E\{d(\mathbf{X}, \beta(\alpha(\mathbf{X})))\} \quad \text{and} \quad R(Q_K) = (1/K)E\{|\gamma(\alpha(\mathbf{X}))|\}.$$

If the minimization were global, with reference to Figures 5.2 and 5.3, it can be easily inferred that $-\lambda$ is the slope of the line in the (R, D) plane that passes through the point $(R(Q_K), D(Q_K))$ and supports the convex hull of the Kth-order operational distortion-rate function $D_K(R)$. Therefore, by repeating the minimization of $J_\lambda(\alpha, \gamma, \beta)$ for various λ, the entire convex hull of $D_K(R)$ can be found.

Two points on the convex hull are easy to find. The first point (R_∞, D_∞) is obtained by minimizing $J_\lambda(\alpha, \gamma, \beta)$ in the limit of large λ, hence, the notation R_∞ and D_∞. This point corresponds to the rate 0 channel codebook, in which there is only one codeword, the null string. In this case, $|\gamma(\alpha(\mathbf{x}))| = 0$, and $\beta(\alpha(\mathbf{x})) = \mathbf{y}^* = \arg\min_{\mathbf{y}} E\{d(\mathbf{X}, \mathbf{y})\}$ with $\mathbf{y}^*$ being the centroid of the entire input vector space. Therefore, $R_\infty = 0$, and $D_\infty = (1/K)E\{d(\mathbf{X}, \mathbf{y}^*)\}$. The second point (R_0, D_0), obtained at $\lambda = 0$, corresponds to the fixed-rate codebook designed by the GL algorithm. Ideally, the codebook size N should be arbitrarily large or, equivalently, the average code rate R_0 should not be constrained. In practical design, however, N has to be limited to some finite value, which, in turn, constrains R_0 to be at most $(\log_2 N)/K$. From this point of view, clearly, the quantizers obtained from running the above described algorithm can be regarded as both *codebook size* and *entropy* constrained. One of many ways to obtain the remainder of the points on the convex hull of $D_K(R)$ is to "walk up" the curve starting from the point (R_0, D_0), which corresponds to $\lambda = \lambda_0 = 0$, by running the algorithm for increasing values λ_1, λ_2, $\cdots$, as shown in Figure 5.4.

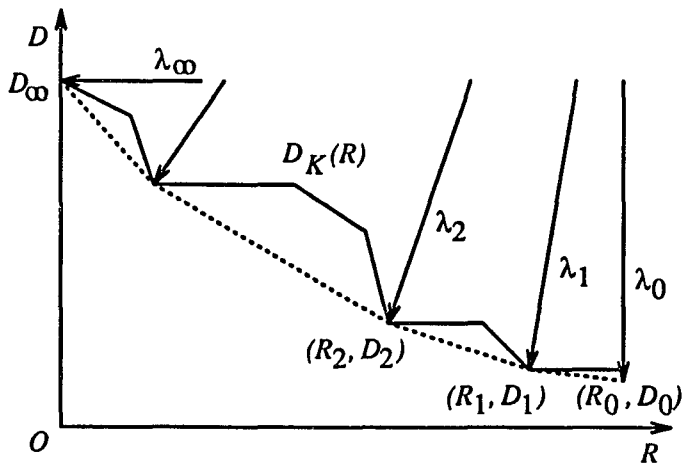


Figure 5.4: Running the ECVQ algorithm for different values of λ to obtain the convex hull of $D_K(R)$

5.3.3 Performance and complexity

It has been shown that ECVQ systems outperform many other scalar and vector quantizers with their reconstruction level and code vector indices also entropy-coded [21]. Performance gains are especially significant for sources with memory, such as speech signals and image pixel sequences. In Figure 5.5, a comparison of rate-distortion performance in terms of SQNR between ECVQ and full search VQ (with and without entropy coding of the code vector indices) for the same Gauss-Markov source ($\rho = 0.9$) that was used in the experiments of Figure 4.3 is given. For this comparison, the vector dimension is once again fixed at 4. The ECVQ has a vector codebook size of 512, and its SQNR curve was obtained by varying the value of the Lagrange multiplier λ, whereas the SQNR values corresponding to full search VQ were measured at different codebook sizes. From the figure, it can be seen that the ECVQ outperforms the full search VQ. The performance improvement is however not

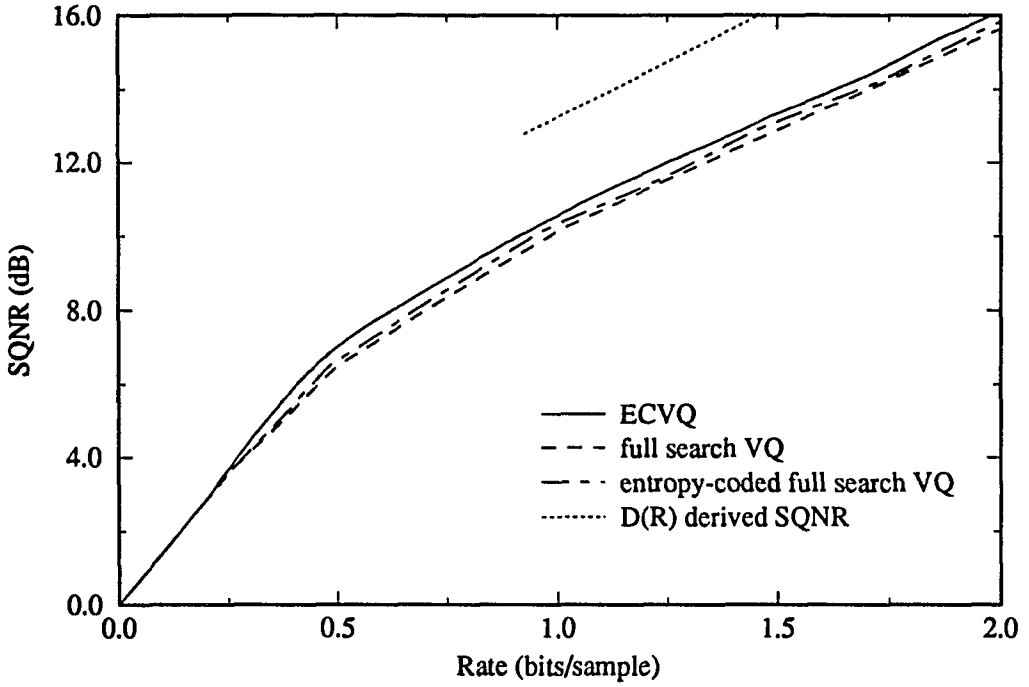


Figure 5.5: SQNR versus code rate of ECVQ and full search VQ for a Gauss-Markov source with $\rho = 0.9$

very significant, mainly because the full search VQ has already achieved quite good performance for the source under consideration.

The SQNR curve derived from the distortion-rate function for the Gauss-Markov source used in the above comparison is also shown in Figure 5.5. This curve is given by (see Equation (5.28))

$$\text{SQNR} \cong 6.02R + 7.21 \text{ dB}$$

for $R > 0.926$ bits/sample, and is the upper bound on the SQNR curve that can be achieved by any quantizer. From the figure, it is evident that a large gap exists between the bounding SQNR curve and the SQNR curves that are achieved by the ECVQ and full search VQ algorithms. By increasing the vector dimension, this

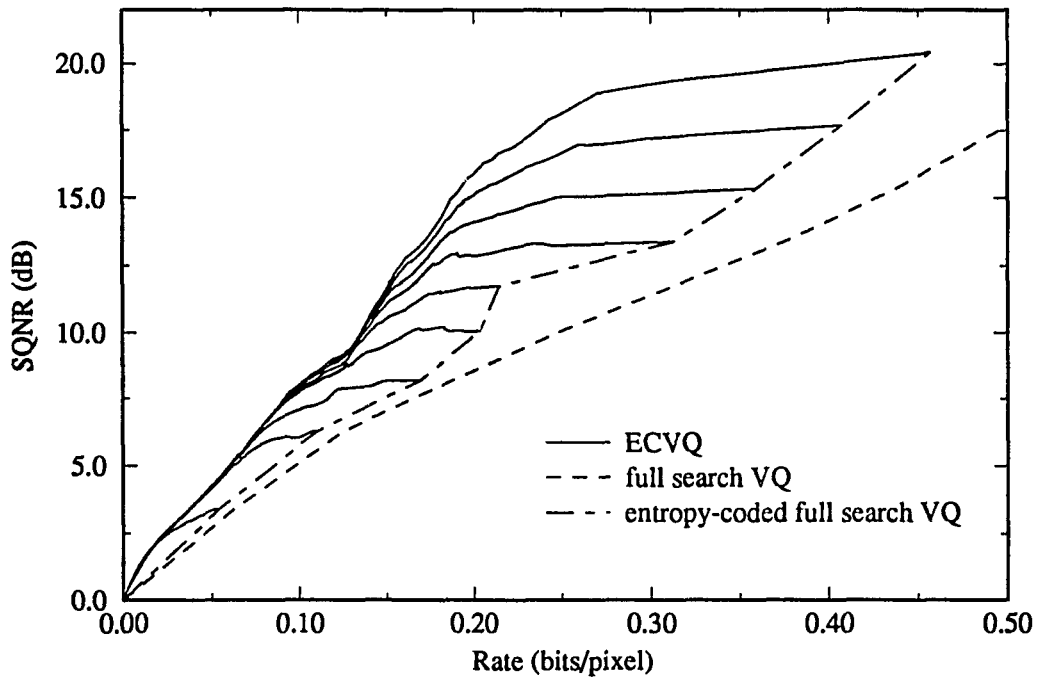


Figure 5.6: SQNR versus code rate of ECVQ and full search VQ for a set of training images

performance gap can be narrowed.

Another performance comparison between ECVQ and full search VQ is given in Figure 5.6. In this comparison, the vector dimension is fixed at 16, and the SQNR curves were obtained by applying different VQ algorithms to a sequence of 89,088 training vectors. The training vector sequence was formed by 4 medical MR images that were used in the first part of this study, and each of the training vectors contained a contiguous block of 4 × 4 pixels of one of the images. The vector codebook size for ECVQ was taken to be $N = 2^n$ where $n = 1, 2, \cdots, 9$. Clearly, from Figure 5.6, it can be seen that the ECVQ algorithm improves the rate-distortion performance substantially over full search VQ and, more importantly, the improvement increases

with the codebook size N.

The ECVQ algorithm itself generally reduces the vector codebook size as λ increases. This is because during the course of the iterative design procedure, a particular cluster, say the ith cluster, may become empty, *i.e.*, $\alpha(\mathbf{x})$ never equals i in Step 1 of the procedure. In this case, Step 2 will set $|\gamma(i)| = \infty$ so that the cluster will never become populated again as the iterative procedure proceeds. Unlike the GL algorithm, there is no need to repopulate empty clusters in the ECVQ algorithm: for example, splitting a highly populated cluster may indeed reduce distortion, but it may also increase the entropy of the code vector indices. Therefore, code vectors corresponding to the empty clusters can be removed from the vector codebook and the iterative procedure can be continued with the smaller codebook size. In Figure 5.7, the effective codebook size of ECVQ versus average code rate for the experiments of Figure 5.6 is shown for different initial codebook sizes N. From Figures 5.6 and 5.7, it is evident that the performance gain of ECVQ is achieved at the cost of higher complexity, in terms of both computation and memory for training and actual operation as well.

5.4 Other Variable-Rate Vector Quantization Techniques

Besides ECVQ, another important class of variable-rate vector quantization techniques is classified vector quantization, described briefly in Subsection 4.4.2. Recall that in classified VQ, each input vector $\mathbf{x}$ is first classified into one of several categories and then quantized with the vector codebook pertaining to the particular category that $\mathbf{x}$ falls in. Apparently, the sizes of codebooks for different categories can be made different such that vectors with different characteristics are quantized with different

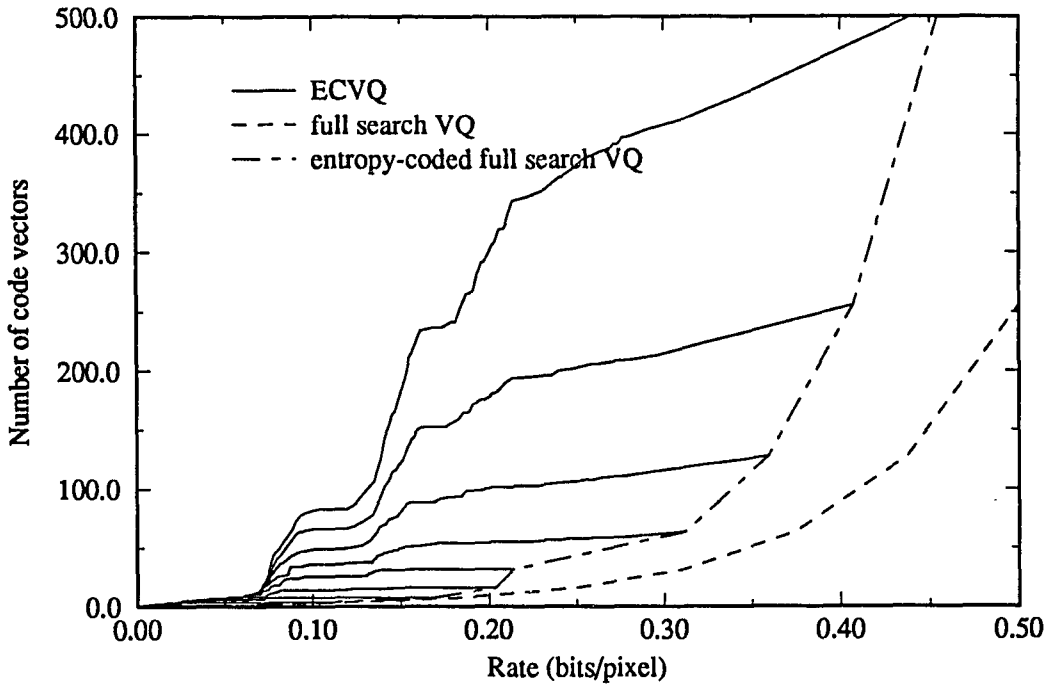


Figure 5.7: Effective vector codebook sizes of ECVQ for different initial codebook sizes

resolutions. For example, image blocks containing edges may be quantized using a codebook with more code vectors while the more frequent nonedge blocks may be quantized with a codebook of smaller size [89]. In this manner, it is possible to improve the distortion performance, especially to preserve the perceptually important edge locations and angular orientations, while maintaining the average code rate at the desired level. Of course, a lossless variable-rate coding can always be applied to the resulting code vector indices such that the average code rate be further reduced [71].

A third important class of variable-rate VQ techniques is the unbalanced and/or nonuniform tree-structured VQ. Recall that in TSVQ a code vector index, *i.e.*, the

codeword, is the concatenation of indices of the tree nodes along the path from the root to a terminal node. Therefore, an unbalanced and/or nonuniform tree can be naturally used to implement a variable-rate VQ system. This class of variable-rate VQ techniques will be discussed in detail over the next chapter together with several new algorithms.

6. VARIABLE-RATE TREE-STRUCTURED VECTOR QUANTIZATION

6.1 Introduction

Three major vector quantization techniques, *viz.*, full search VQ, tree-structured VQ, and entropy-constrained VQ, were described in the last two chapters. Among these three techniques, full search VQ is neither optimal in terms of rate-distortion performance nor efficient in implementation. Its design procedure, *i.e.*, the generalized Lloyd algorithm, however, provides the basis for designing other variations of vector quantizers. ECVQ signifies a new direction in designing VQ systems for its capability to achieve compression performance close to the operational distortion-rate function, the optimum bound for a given vector dimension. Unfortunately, its implementation complexity is even higher than that of full search VQ due to the larger number of code vector searches required in quantizing each input vector. TSVQ, on the other hand, requires only a sequence of binary (or m-ary) searches instead of one large search as in full search VQ and ECVQ, and hence greatly reduces the computational complexity. As a tradeoff, its performance in general suffers some degradation over that of full search VQ for the same number of code vectors due to the suboptimal structure of the vector codebook and the constraint on the code vector search. Also, the memory requirement of binary TSVQ is nearly doubled. In

this chapter, the problem of combining TSVQ technique with the idea of ECVQ is considered. The objective is to improve the rate-distortion performance of TSVQ systems while retaining the low computational complexity and the progressive transmission capability of such systems as well as maintaining the memory requirement at a reasonable level.

6.1.1 The Makhoul TSVQ algorithm

Traditionally, TSVQ systems have been designed *one stage at a time* by splitting each intermediate code vector of last stage into two vectors and then applying the GL algorithm to each pair of the new vectors. As a result, the tree grown for such a TSVQ system is balanced, implementing a fixed-rate VQ. An alternative TSVQ design algorithm was introduced by Makhoul *et al.* [68], where the tree is designed *one node at a time* rather than one stage at a time by splitting the terminal node that contributes most to the overall distortion of the vector quantizer. This algorithm is "greedy" in the sense that it only considers the short term effects of extending the tree or, in other words, it only considers what happens with the addition of a single new pair of terminal nodes grown from a current terminal node. In general, an application of the algorithm will result in an unbalanced tree because the node that is split at any time can be at any depth. Such a tree, however, can be used to implement either a variable-rate VQ system, as being explicit from its unbalanced nature, or a fixed-rate system, which is actually what Makhoul *et al.* presented in their paper. In the fixed-rate implementation, the tree is first grown until the number of terminal nodes reaches a predetermined power of two. The terminal nodes, each associated with a code vector, are then renumbered and this remapping is used as

the code vector indices. Therefore, although the tree is unbalanced, the quantization will be at fixed rate. Such a fixed-rate mapping, however, renders the tree unusable for progressive transmission since the resulting codeword bits no longer reflect the successive approximation of the vectors being quantized in the encoding process.

One advantage of the above described tree growing scheme is that there will be more code vectors available to quantize inputs with larger variances, such as image edge blocks, since this is where the tree has been split the most. In addition, the low complexity tree-structured code vector search of TSVQ is retained. Makhoul *et al.* reported that their tree resulted in lower average distortion than balanced TSVQ, but higher than full search VQ.

6.1.2 Pruned TSVQ algorithm

Another TSVQ algorithm, referred to as the *pruned tree-structured VQ* (PTSVQ) algorithm, is described in [22, 93]. The design of a PTSVQ system is based on an extension of an algorithm for optimal tree pruning in tree-structured classification and regression due to Breiman *et al.* [15], known as the *generalized BFOS algorithm.* In such a design, a balanced tree is first grown as in the traditional TSVQ design and then pruned back into a sequence of unbalanced subtrees. Each of the subtrees is optimal in that it has the lowest average distortion among all subtrees of the original balanced tree with the same or lower average code rate; and the sequence of these optimal subtrees is obtained by always pruning off the branch of the last pruned subtree that leads to the lowest slope or ratio ϕ of increase in distortion to decrease in rate. Clearly, the subtrees so obtained are nested, which makes the complexity of this pruning algorithm much less than that of exhaustively examining all subtrees of

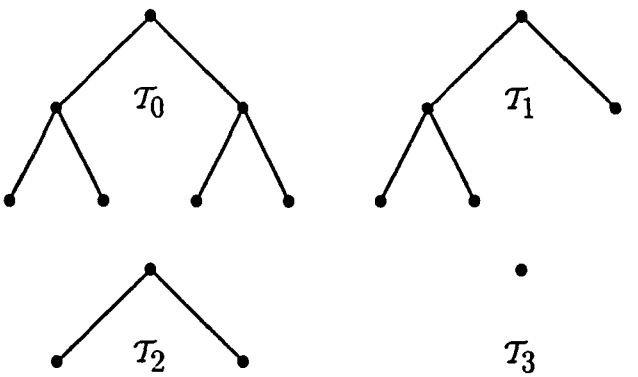


Figure 6.1: A sequence of nested subtrees

the original balanced tree. In Figure 6.1, an example sequence of nested subtrees is shown where, the unbalanced tree $\mathcal{T}_1$ is a subtree of the balanced tree $\mathcal{T}_0$, $\mathcal{T}_2$ is a subtree of $\mathcal{T}_1$, and so on.

Given a desired rate, PTSVQ operates in the same way as the traditional TSVQ, except that the tree is unbalanced, which leads to a natural variable-rate implementation. For different desired rates, different trees are used. Each of these trees is one of the above described nested subtrees, corresponding to a distinct rate-distortion pair (R, D) and being optimal for that particular rate. Because the subtrees are nested, they can all be embedded into one single unbalanced tree, *i.e.*, the subtree corresponding to the highest rate. As a result, the codewords of subtrees corresponding to lower rates are prefixes of the codewords of subtrees corresponding to higher rates. Clearly, this embedded property makes PTSVQ usable to progressive transmission applications. Also, because of the optimality of the subtrees, PTSVQ ensures that the quality of the intermediate reconstructions will be the best possible for the number of received bits, given that the quantizer is obtained from the original balanced tree.

PTSVQ has the ability to outperform the traditional balanced TSVQ in the rate-distortion sense as being able to devote more bits to high distortion events. It has been also shown that PTSVQ even outperforms fixed-rate full search VQ over most rates of interest [91, 93]. In addition, for the same code rate, the average computational requirement of PTSVQ for quantizing an input vector remains the same as that of the traditional TSVQ.

The successive tree pruning process in the design of a PTSVQ system can be regarded as a tree-structured version of the process of "walking up" the $D_K(R)$ curve from the point (R_0, D_0) in ECVQ design and, correspondingly, the ratio ϕ of increase in distortion to decrease in rate resulting from pruning a branch is nothing but the Lagrange multiplier λ. Similar to ECVQ technique, PTSVQ requires that a large initial balanced tree should be grown before pruning back in order to achieve good rate-distortion performance. This, in turn, poses a severe disadvantage to PTSVQ algorithm. Recall that the memory requirement for storing a balanced tree is exponentially proportional to the depth of the tree. In many cases of image compression application, however, it is common to grow some branches of a tree to a depth of more than 20 to allocate fine tuned code vectors for the edge blocks. For those cases, clearly, the memory requirement for the design of a PTSVQ system becomes prohibitively excessive.

6.2 Greedily-Grown Tree-Structured Vector Quantization

In view of the above mentioned disadvantage of the PTSVQ algorithm, a new method, referred to as the *greedily-grown tree-structured VQ* (GTSVQ) method, has been developed together with four different algorithms for designing unbalanced TSVQ

systems. This method resembles a constrained inverse of the pruning algorithm, that is, it grows a tree directly from scratch and, thereby, precludes the requirement of growing a large initial tree. It follows the same approach of the Makhoul TSVQ algorithm to grow a tree in a greedy fashion by splitting one terminal node at a time without considering the effects of later growth of the tree. However, instead of splitting the node that contributes the most distortion as in the Makhoul algorithm, the new method splits the node that provides the best tradeoff between overall average distortion and average code rate, *i.e.*, the node that results in the highest ratio of decrease in distortion to increase in rate. As a result of applying the method, an unbalanced tree or, more precisely, a sequence of nested unbalanced trees, is grown, and these nested trees can then be used to implement a variable-rate quantizer which is amenable to progressive transmission applications. To describe the method and its first two constituent algorithms, GTSVQ-I and GTSVQ-II, let us first define some notations and then examine the effects of splitting a tree node on the overall average distortion and code rate.

Given an arbitrary tree $\mathcal{T}$, let $\mathcal{S}$ denote a subtree of $\mathcal{T}$ and τ denote an internal or terminal node of $\mathcal{T}$. In the remainder of this chapter, the relationship between $\mathcal{S}$ and $\mathcal{T}$, and, τ and $\mathcal{T}$ will be expressed as $\mathcal{S} \sqsubset \mathcal{T}$ and $\tau \in \mathcal{T}$, respectively. If τ is a terminal node of $\mathcal{T}$ and this relationship needs to be specifically identified, then the expression $\tau \dashv \mathcal{T}$ will be used. Without ambiguity, the notation τ for a node will also be used to denote the index of the code vector (or intermediate code vector) associated with the node.

Using the above notations, the overall average distortion of a K-dimensional vector quantizer $Q_{\mathcal{T}} = (\alpha_{\mathcal{T}}, \beta_{\mathcal{T}})$ with a binary tree-structured vector codebook $\mathcal{T}$

can be expressed as

$$D(\mathcal{T}) = \sum_{\tau \dashv \mathcal{T}} P_\tau D_\tau, \tag{6.1a}$$

where $P_\tau = P[\alpha_{\mathcal{T}}(\mathbf{X}) = \tau]$ is the probability that $\mathbf{y}_\tau$ is selected as the code vector for an input through the tree-structured search in $\mathcal{T}$, and D_τ is the conditional average distortion over all the input vectors for which $\mathbf{y}_\tau$ is selected as the code vector, *i.e.*,

$$D_\tau = \frac{1}{K} E\{d(\mathbf{X}, \mathbf{y}_\tau) \mid \alpha_{\mathcal{T}}(\mathbf{X}) = \tau\}.$$

Correspondingly, the average code rate of $Q_{\mathcal{T}}$ can be expressed as[1]

$$R(\mathcal{T}) = \sum_{\tau \dashv \mathcal{T}} P_\tau \mu_\tau \quad \text{bits/vector}, \tag{6.1b}$$

where μ_τ is the length of the path in $\mathcal{T}$ from the root to the node τ, *i.e.*, the depth of τ in $\mathcal{T}$.

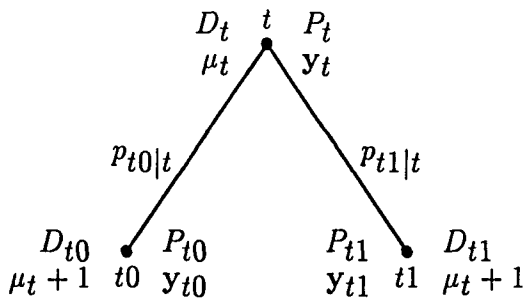
If the terminal node t of $\mathcal{T}$ is split, by applying the GL algorithm, into two new nodes $t0$ and $t1$ as shown in Figure 6.2, a new tree $\mathcal{T}'$ results of which $\mathcal{T}$ is a subtree. In this case, the average distortion becomes

$$D(\mathcal{T}') = P_{t0} D_{t0} + P_{t1} D_{t1} + \sum_{\substack{\tau \dashv \mathcal{T} \\ \tau \neq t}} P_\tau D_\tau, \tag{6.2a}$$

and, since the depth of a node is always one more than the depth of its immediate ancestor, the average rate becomes

$$R(\mathcal{T}') = P_{t0}(\mu_t + 1) + P_{t1}(\mu_t + 1) + \sum_{\substack{\tau \dashv \mathcal{T} \\ \tau \neq t}} P_\tau \mu_\tau \quad \text{bits/vector}. \tag{6.2b}$$

[1] From now on until the end of this chapter, for simplicity of discussions, code rates will be measured in number of bits per vector and number of bits per vector component interchangeably. This causes no confusions when the vector dimension is given.



$$p_{t0|t} + p_{t1|t} = 1$$
$$P_{t0} = p_{t0|t}\, P_t$$
$$P_{t1} = p_{t1|t}\, P_t$$
$$P_{t0} + P_{t1} = P_t$$

Figure 6.2: Splitting the node t into two new nodes $t0$ and $t1$

Clearly, because of the split, the average distortion has been decreased by an amount

$$\Delta D_t = D(\mathcal{T}) - D(\mathcal{T}') = P_t D_t - P_{t0} D_{t0} - P_{t1} D_{t1}, \tag{6.3a}$$

and the average code rate has been increased by

$$\Delta R_t = R(\mathcal{T}') - R(\mathcal{T}) = P_t \quad \text{bits/vector.} \tag{6.3b}$$

The ratio of these changes is therefore given by

$$\phi(t) = \frac{\Delta D_t}{\Delta R_t} = D_t - p_{t0|t} D_{t0} - p_{t1|t} D_{t1} \tag{6.4}$$

where, for $i = 0$ or 1, $p_{ti|t} = P_{ti}/P_t$ is the conditional probability that $\mathbf{y}_{ti}$ is selected as the code vector for an input given that $\mathbf{y}_t$ has already been chosen as an intermediate code vector for the input.

In both of the new algorithms GTSVQ-I and GTSVQ-II that we propose, an unbalanced tree-structured vector quantizer is designed one tree node at a time by always splitting the terminal node that leads to the largest ϕ. These two new algorithms will be described in detail in the next two subsections, respectively, and some related discussions will be given in the following subsection.

6.2.1 The first new algorithm: GTSVQ-I

A looped procedure of the GTSVQ-I algorithm for designing an unbalanced binary tree $\mathcal{T}$ based on a training vector sequence is given in the following. In this procedure, two descendent nodes for each terminal node are designed ahead of time so that the ratio ϕ of the terminal node can be evaluated. The terminal nodes, however, are not split right away. Instead, they are put into a waiting list, from which the one with the largest ϕ will be selected for splitting during each pass of the design loop.

Step 1: *Initialization.*

1. Design the root node t for $\mathcal{T}$ together with the corresponding intermediate code vector $\mathbf{y}^*$ (for the squared-error distortion measure, simply take the mean of the entire training sequence as $\mathbf{y}^*$);
2. Set $R(\mathcal{T}) = 0$ and calculate $D(\mathcal{T}) = (1/K)E\{d(\mathbf{X}, \mathbf{y}^*)\}$;
3. Design two descendent nodes for t using the GL algorithm, but do not split t;
4. Calculate ΔD_t, ΔR_t, and $\phi(t)$ using Equations (6.3) and (6.4);
5. Put t into the waiting list together with ΔD_t, ΔR_t, and $\phi(t)$.

Step 2: *Splitting of a terminal node and evaluation of rate-distortion performance of the new tree.*

1. Search the waiting list for the node t with the largest ϕ;
2. Split t into two descendent nodes $t0$ and $t1$;
3. Calculate $D(\mathcal{T}') = D(\mathcal{T}) - \Delta D_t$ and $R(\mathcal{T}') = R(\mathcal{T}) + \Delta R_t$.

Step 3: *Design of descendent nodes.* For $i = 0$ or 1,

1. Design two descendent nodes for the newly sprouted terminal node ti using the GL algorithm, but do not actually split ti
2. Calculate ΔD_{ti}, ΔR_{ti}, and $\phi(ti)$ using Equations (6.3) and (6.4);
3. Put ti into the waiting list together with ΔD_{ti}, ΔR_{ti}, and $\phi(ti)$.

Step 4: *Termination test.* Set $\mathcal{T}' \rightarrow \mathcal{T}$. If $D(\mathcal{T}')$ or $R(\mathcal{T}')$ reaches a predetermined value, stop and output the new tree $\mathcal{T}$; otherwise go to Step 2.

It should be mentioned that in Step 3 of the above described procedure, the newly sprouted terminal node ti may contain just a few training vectors. Also, it is possible that ti contains a large number of training vectors, but one of its two descendent nodes from the design turns out to be empty or nearly empty. In such cases, ti will not be placed into the waiting list, that is, it will be frozen and not be split any further.

In general, for a given sequence of input vectors, such as a sequence of image blocks, a TSVQ system designed by the above procedure may perform quantizing operations in the same manner as that of other VQ systems, *i.e.*, quantize each input into a final code vector before quantizing the next vector. When used in a progressive transmission application, however, the quantization should proceed in the same order that the tree nodes were split. For example, suppose in the design of the tree the splitting of the node t was followed by that of s. Then, in real applications, the input vectors quantized into $\mathbf{y}_s$ should be further quantized into $\mathbf{y}_{s0}$ or $\mathbf{y}_{s1}$ right after those quantized into $\mathbf{y}_t$ have been further quantized into $\mathbf{y}_{t0}$ or $\mathbf{y}_{t1}$, no matter whether t is the immediate ancestor of s or not. In other words, input vectors to be quantized

are classified into different clusters of varying "importance" levels, each associated with a distinct node in the tree. The clusters that were expected to provide better rate-distortion tradeoffs are regarded as "more important" and will be quantized first, and then followed by those judged to be "less important".

Similar to the traditional TSVQ algorithm, in GTSVQ-I, the code vector index for each individual input vector being quantized is transmitted or stored as the codeword or, equivalently, the indices of successive intermediate code vectors are transmitted or stored as the codeword bits.

6.2.2 The second new algorithm: GTSVQ-II

The tree design procedure of this new algorithm is exactly the same as that of GTSVQ-I. Also, the quantizing operations in both algorithms proceed in the same fashion, *i.e.*, input vectors are quantized successively in the order that the nodes associated with their intermediate code vectors were split. However, instead of simply transmitting or storing the code vector indices as codewords as in GTSVQ-I, the GTSVQ-II algorithm requires that the indices be encoded with an entropy code before being transmitted or stored. When used in a progressive transmission application, this can be realized by encoding the sequence of intermediate code vector indices with a binary arithmetic code. Thus, GTSVQ-II is slightly more complicated than the straightforward GTSVQ-I, but it can achieve better performance in the rate-distortion sense.

Strictly speaking, GTSVQ-II should not be counted as a new algorithm, since it is just a somewhat trivial improvement of the GTSVQ-I algorithm by incorporating a well developed entropy coding technique. It is presented here in this way mainly for

performance comparison purpose. As will be seen in the next section, it also serves as an introduction to two other new algorithms.

6.2.3 Comparison with other TSVQ algorithms

From the descriptions given in the last two subsections, it can be seen that the two new algorithms GTSVQ-I and GTSVQ-II are very similar to the Makhoul TSVQ algorithm except for the splitting criterion. That is, in GTSVQ-I and GTSVQ-II, the terminal node t with the largest ratio $\phi(t)$ of decrease in distortion to increase in rate is split, whereas in the Makhoul algorithm, the node t with the largest distortion D_t is split. In other words, the new algorithms optimize the rate-distortion tradeoff at each split, rather than splitting a node t without considering the resulting increase in rate ΔR_t and decrease in distortion ΔD_t. They are still greedy, however, since they split nodes without considering future tree behavior.

The ratio ϕ in the new GTSVQ algorithms has the same interpretation as that in the PTSVQ algorithm. Recall that in PTSVQ algorithm, the branch of a tree with minimum ϕ is pruned to get the minimum increase in distortion for a decrease in rate. This serves to keep the distortion as low as possible as the rate decreases. Conversely, in the GTSVQ algorithms, the terminal with the largest ϕ is split to procure the maximum decrease in distortion for an increase in rate. This attempts to reduce the distortion as much as possible and as quickly as possible. Therefore, the pruning process in PTSVQ and the growing process in GTSVQ can be regarded as the constrained inverse of each other. The primary difference is that the growing is greedy and suboptimal since the tree is grown only one node at a time, whereas the pruning is optimal since many nodes may be pruned off at once. In other words, PTSVQ has

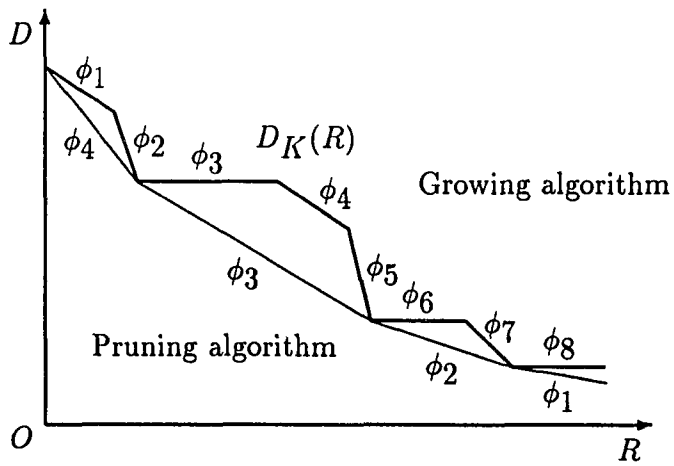


Figure 6.3: Walking up and down the operational distortion-rate function curve in tree pruning and growing algorithms

greater flexibility in trading off a smaller increase in distortion for a large decrease in rate. From this point of view, it can be seen that the tree growing process in GTSVQ closely resembles the process of "walking down" the operational distortion-rate function curve, as shown in Figure 6.3, with each node split corresponding to one step and the ratio ϕ of the split specifying the direction of the walk.

Compared to growing balanced trees in the traditional TSVQ algorithm, there is no guarantee that the unbalanced trees resulting from the new GTSVQ algorithms will always lead to a lower average distortion for a given rate. When growing a tree, it is possible that the splitting of a node t which has a small ϕ results in a descendent node ti with a large ϕ. Should that be the case, then the node ti will be split by the balanced tree growing algorithm, as long as it is at a depth smaller than that of the final tree, but not necessarily by an unbalanced tree growing algorithm. In other words, choosing not to split a node with small ϕ may preclude a descendent

split with large ϕ. Hence, there is no guarantee that the unbalanced tree will even outperform the balanced tree of the same rate. Adding a look-ahead step in the new algorithms would improve the performance and rule this out in some cases, but would also add additional complexity to the algorithms. On the other hand, the balanced tree will miss every split with a high ϕ that occurs at a depth greater than the depth of the final tree and may be forced to have many poor splits in it. Therefore, it is reasonably to expect that, in most cases, growing an unbalanced tree would yield a lower average distortion than growing a balanced tree of the same rate.

It should be pointed out that while the new GTSVQ algorithms are expected to be able to achieve better rate-distortion performance in most cases, their average computational requirements for quantizing an input vector remain roughly the same as that of the traditional TSVQ algorithm, given that the quantizations are at the same code rate. For example, in the GTSVQ-I algorithm, the probability that an input vector $\mathbf{x}$ is quantized into the code vector $\mathbf{y}_t$ is P_t and, in the course of quantizing $\mathbf{x}$ into $\mathbf{y}_t$, a total number of μ_t binary searches are required, where μ_t is the depth of the tree node t. Hence, the average number of binary searches is given by $\sum_t P_t \mu_t$, which is nothing but the average code rate and is equal to the number of binary searches that the traditional TSVQ algorithm requires to quantize each input. Compared to GTSVQ-I, the entropy coding of the code vector indices in GTSVQ-II adds additional computational requirement but helps enhancing the rate-distortion performance. So, it can be also expected that the average computational requirement of GTSVQ-II stays roughly the same as that of the traditional algorithm for the same code rate.

No analytical expression for memory requirements of the new algorithms can

be derived. On one hand, some branches of a tree grown by these algorithms may have depths much larger than that of a balanced tree grown by the traditional TSVQ algorithm, and these branches require much more memory locations to store their internal nodes. On the other hand, the new algorithms avoids many poor splits as well as splitting the empty or nearly empty nodes and, thereby, avoids growing complete balanced trees for which the memory requirement is exponentially proportional to the number of terminal nodes. Therefore, it is expected that the memory requirements of the new algorithms are higher than that of the traditional algorithm for the same code rate, but not at an excessive level. More about this respect of VQ algorithms will be discussed in Section 6.4 where the experimental results are presented.

6.3 Entropy-Constrained Greedily-Grown Tree-Structured Vector Quantization

As presented in the last section, both GTSVQ-I and GTSVQ-II algorithms design an unbalanced tree-structured vector codebook by always splitting the terminal tree node with the largest ratio $\phi = \Delta D/\Delta R$. Also, for a tree so designed, GTSVQ-II employs an entropy code, particularly an arithmetic code, to encode the code vector indices in an attempt to improve the rate-distortion performance over the GTSVQ-I algorithm. Apparently, under the condition that an entropy code is used to encode the code vector indices, the quantity ϕ defined as the ratio of decrease in distortion to increase in entropy can be used as the tree node splitting criterion and naturally incorporated into the unbalanced tree growing procedure described in Subsection 6.2.1 to improve the rate-distortion performance of the tree.

Given a tree $\mathcal{T}$ and with entropy coding, one can achieve an average distortion

given by

$$D(\mathcal{T}) = \sum_{\tau \dashv \mathcal{T}} P_\tau D_\tau, \tag{6.5a}$$

which is the same as that expressed in (6.1a), and at the same time reduce the average code rate to the zeroth-order entropy of the code vector indices, *i.e.*,

$$R(\mathcal{T}) = -\sum_{\tau \dashv \mathcal{T}} P_\tau \log_2 P_\tau \quad \text{bits/vector.} \tag{6.5b}$$

Now, suppose the terminal node t of $\mathcal{T}$ is split into two new nodes $t0$ and $t1$ as depicted in Figure 6.2, resulting in a new, supertree $\mathcal{T}'$ of $\mathcal{T}$. Then, it can be shown that (see Equation 6.1c) the average distortion of the new tree is

$$D(\mathcal{T}') = P_{t0} D_{t0} + P_{t1} D_{t1} + \sum_{\substack{\tau \dashv \mathcal{T} \\ \tau \neq t}} P_\tau D_\tau, \tag{6.6a}$$

and the average code rate changes to

$$\begin{aligned} R(\mathcal{T}') &= -P_{t0} \log_2 P_{t0} - P_{t1} \log_2 P_{t1} - \sum_{\substack{\tau \dashv \mathcal{T} \\ \tau \neq t}} P_\tau \log_2 P_\tau \\ &= P_t H_t + R(\mathcal{T}) \quad \text{bits/vector,} \end{aligned} \tag{6.6b}$$

where H_t is the conditional entropy of code vector indices given that $\mathbf{y}_t$ has been chosen as the intermediate code vector, *i.e.*,

$$H_t = -p_{0|t} \log_2 p_{0|t} - p_{1|t} \log_2 p_{0|t} \quad \text{bits/vector.}$$

Hence, due to splitting t, the average distortion is decreased by an amount of $\Delta D_t = P_t D_t - P_{t0} D_{t0} - P_{t1} D_{t1}$, which is also the same as that expressed in (6.2c), while the average code rate is increased by $\Delta R_t = P_t H_t$, and the ratio of these changes is given by

$$\phi(t) = \frac{\Delta D_t}{\Delta R_t} = \frac{D_t - p_{t0|t} D_{t0} - p_{t1|t} D_{t1}}{H_t}. \tag{6.7}$$

Based on the idea of using the above defined ratio ϕ as the terminal node splitting criterion in a tree growing procedure, two other new unbalanced TSVQ algorithms, GTSVQ-III and GTSVQ-IV, have been developed. Since both of these new algorithms take into account the change in entropy of the code vector indices when selecting terminal tree nodes for splitting, they are also referred to as *entropy-constrained greedily-grown tree-structured VQ* (ECGTSVQ) algorithms. In the following two subsections, these algorithms will be described, respectively.

6.3.1 The third new algorithm: GTSVQ-III

The GTSVQ-III algorithm follows the same looped procedure of the GTSVQ-I and GTSVQ-II algorithm, described in Subsection 6.2.1, to grow unbalanced trees. That is, during each pass of the design loop, only one terminal node is split and this node, compared to others in the waiting list, should have the largest ratio $\phi = \Delta D/\Delta R$ of decrease in distortion to increase in rate. The only difference between this new algorithm and the two predecessors lies in the definition of ΔR: here, ΔR is defined as the increase in zeroth-order entropy of the code vector indices, whereas in GTSVQ-I and GTSVQ-II, ΔR is defined as the increase in average length of binary representations of the indices. Therefore, the splitting of a node in the GTSVQ-III design procedure can be interpreted as one that provides the best tradeoff between distortion and code vector index entropy.

Similar to GTSVQ-II, the quantizing operation of a TSVQ system designed by the GTSVQ-III algorithm requires that the code vector indices be encoded with an entropy code before being transmitted or stored. When engaged in progressive transmission applications, a binary arithmetic code will be used to encode the sequence

of intermediate code vectors.

6.3.2 The fourth new algorithm: GTSVQ-IV

This new algorithm resembles the GTSVQ-III algorithm in all respects, including those related to the tree growing procedure and practical quantizing operations, with only one important exception: the design of descendents for the newly sprouted terminal nodes. Recall that the general schema of the GTSVQ method is to grow an unbalanced tree by always splitting the terminal node that has the largest ratio ϕ. In this manner, one attempts to reduce the distortion as much as possible for an increase in rate at each node split. The ratio ϕ of a terminal node is determined when the splitting of the node is designed, which is before the node is put into the waiting list as a candidate for actual splitting. Clearly, if the ratio ϕ of each terminal node can be somehow designed to attain a larger value, it is possible to reduce the distortion even more at each node split and, thereby, achieve a further improved rate-distortion performance. To realize this idea, GTSVQ-IV employs the ECVQ algorithm described in the last chapter to design the split of each terminal node rather than using the GL algorithm as in the three previous GTSVQ algorithms.

The design of splitting a terminal node into two descendents is nothing but the design of a vector quantizer with two code vectors based on training vectors that have been classified into the cluster associated with the terminal node. When the ECVQ algorithm is used for the design, the rate-distortion performance of the resulting split varies depending on the specification of the Lagrange multiplier λ. That is, a set of rate-distortion pairs can be obtained from such a design procedure for different choices of λ. As mentioned previously, the curve in the rate-distortion plane that connects

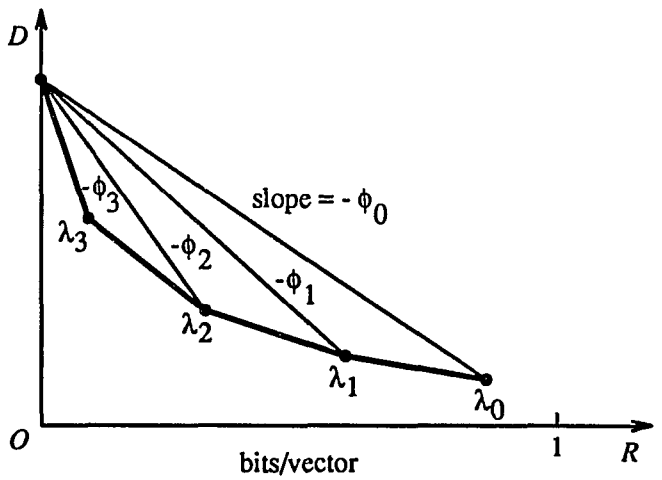


Figure 6.4: Variations in rate and distortion resulting from the splitting of a node into two descendents

these pairs is convex and monotonically decreasing with rate, as shown in Figure 6.4. Also, the point on the curve with the highest rate (and the lowest distortion) corresponds to the rate-distortion performance of the split designed by the ECVQ algorithm with $\lambda = \lambda_0 = 0$ or, equivalently, the GL algorithm. From the convexity of the curve, it can be easily seen that the ratio of decrease in distortion to increase in rate resulting from this split, *i.e.*, the ϕ_0 shown in Figure 6.4, is smaller than that of any other split designed by the ECVQ algorithm with nonzero λ. Therefore, by choosing a nonzero Lagrange multiplier value, a node split can be designed with the ECVQ algorithm to provide a larger ratio ϕ.

Apparently, there are an infinite number of choices of the Lagrange multiplier value for the design of a node split. As can be seen from Figure 6.4, choosing a larger λ would result in a higher ratio ϕ, which is what we desire. However, such a choice would also lead to a smaller decrease in distortion as well as a smaller increase in rate and, consequently, poses two disadvantageous effects on the tree growing process:

first, a larger tree may have to be grown to achieve the desired average distortion level and, second, one of the two descendent nodes from the split may turn out to be nearly empty. With these considerations in mind, one may follow any of the three heuristic approaches given below to choose a reasonable λ value:

1. try several λ values, including $\lambda = 0$, for the splitting and use the one that leads to the largest ϕ as well as a significant decrease in distortion;

2. use larger λ values for splitting popular nodes and smaller values for those underpopulated;

3. design the split with the GL algorithm first and obtain the corresponding ratio of decrease in distortion to increase in rate ϕ_0, and then simply use $\lambda = \phi_0$. From Figures 5.3 and 6.4, it can be seen that this is a valid but very heuristic choice for λ.

A closely related problem to the above considerations is the handling of empty and nearly empty nodes. Similar to other GTSVQ algorithms, in GTSVQ-IV, empty or nearly empty nodes will be frozen and not be split any further. In addition, if one descendent of a terminal node turns out to be nearly empty even when the splitting is designed with the GL algorithm, then the terminal node will be frozen.

6.3.3 Comparison between the ECGTSVQ algorithms

The entropy-constrained GTSVQ algorithms GTSVQ-III and GTSVQ-IV are natural extensions of the GTSVQ-I and GTSVQ-II algorithms given that an entropy code is used to encode the code vector indices. Due to the use of entropy coding in the quantizing operations and the incorporation of code vector index entropy as a

constraint in the tree growing design procedure, these two algorithms are expected to achieve better rate-distortion performances than the two previous GTSVQ algorithms.

Similar to GTSVQ-I and GTSVQ-II, the greedy tree growing process in the GTSVQ-III and GTSVQ-IV algorithms resembles the process of "walking down" the operational rate-distortion function curve step by step with each step corresponding to a terminal node split. The difference between GTSVQ-III and GTSVQ-IV is that the latter uses the ECVQ algorithm to design the walking steps in the attempt to magnify the downward slope ϕ of each step. From the descriptions given in the last subsection, it can be seen that the sizes of walking steps so designed are effectively reduced. As a result, a step with large slope ϕ may force many of the following steps to have only tiny slopes. In light of these arguments, the tree growing process in the GTSVQ-IV algorithm can be said to be more greedy than that of the GTSVQ-III algorithm and, because of this greedy nature, the trees designed by the GTSVQ-IV algorithm are not guaranteed to be able to always provide better rate-distortion performance than those of the GTSVQ-III algorithm.

Due to the use of code vector index entropy as a constraint in the tree design process, no expression for the relationship between the computational or memory requirements of the entropy-constrained GTSVQ algorithms and the average distortion or code rate can be derived analytically. However, it is expected that the memory requirements of these algorithms, especially the GTSVQ-IV algorithm, are higher than those of the GTSVQ-I and GTSVQ-II algorithms for the same level of average code rate. Experimental results demonstrating these requirements will be presented in the next section.

6.4 Experimental Results and Discussions

The four new variable-rate tree-structured vector quantization algorithms described above have been implemented in software and tested on a computer generated Gauss-Markov source and a set of several medical images in order to study their operations and evaluate their performances as well as complexities against several other VQ algorithms, such as the traditional TSVQ algorithm and the full search VQ algorithm. As mentioned in Subsection 6.3.2, any nonnegative value can be chosen as the Lagrange multiplier λ for designing the split of a terminal node in the GTSVQ-IV algorithm. Without loss of generality, only experimental results obtained from the implementation that follows the third heuristic approach described in that subsection are presented.

6.4.1 Quantization of a Gauss-Markov source

In this set of experiments, the rate-distortion performances and the computational and memory requirements of the four GTSVQ algorithms were tested against the ECVQ and traditional TSVQ algorithms for the same Gauss-Markov source that was used in the experiments summarized in Figures 4.3 and 5.5. The results are presented in Figures 6.5, 6.6, and 6.7, respectively. Once again, the vector dimension was fixed at 4 in the experiments and the performance was evaluated in terms of SQNR. The computational requirement was measured by the average number of vector computations that are required to quantize an input vector, while the memory requirement was estimated by the total number of code vectors (including intermediate code vectors) that need to be stored.

From Figure 6.5, it can be seen the performances of the GTSVQ algorithms

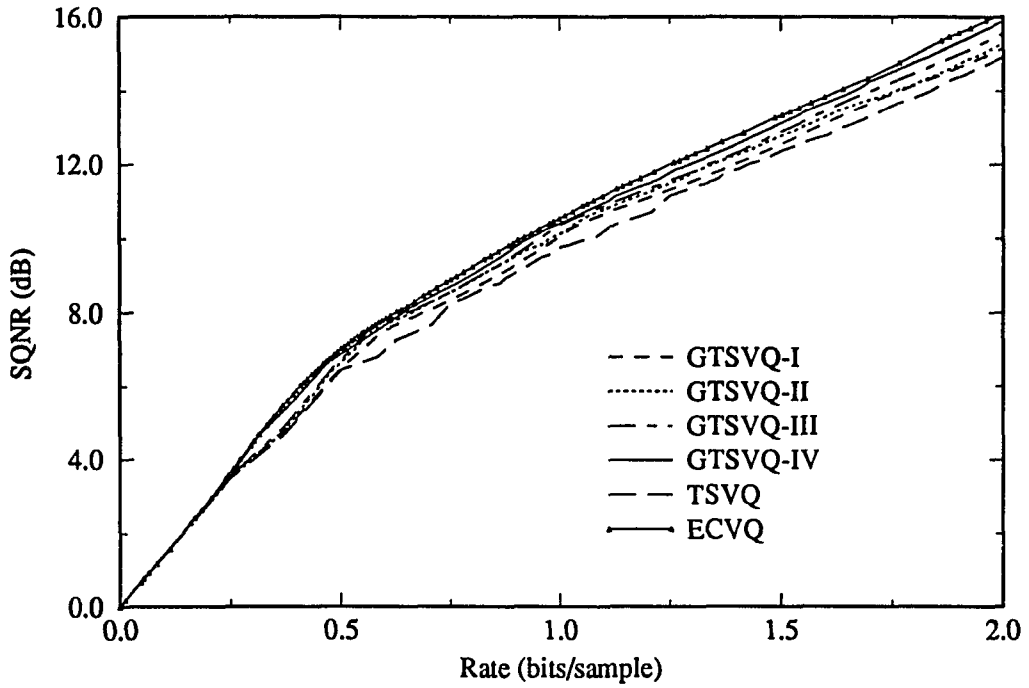


Figure 6.5: SQNR versus code rate of the GTSVQ, traditional TSVQ, and ECVQ algorithms for a Gauss-Markov source with $\rho = 0.9$

lie between those of the ECVQ and traditional TSVQ algorithms, and the four algorithms successively outperform the preceding algorithms. Also, a comparison between Figures 6.5 and 4.3 reveals that the two ECGTSVQ algorithms, *i.e.*, GTSVQ-III and GTSVQ-IV, can achieve roughly the same rate-distortion performance as the full search VQ algorithm for the Gauss-Markov source used in the experiments. Nevertheless, all of the SQNR curves in the two figures are close to each other, implying that even the traditional TSVQ algorithm can achieve quite good performance for the source under consideration.

While the GTSVQ algorithms can achieve improved performance over the traditional TSVQ algorithm for the Gauss-Markov source, their computational require-

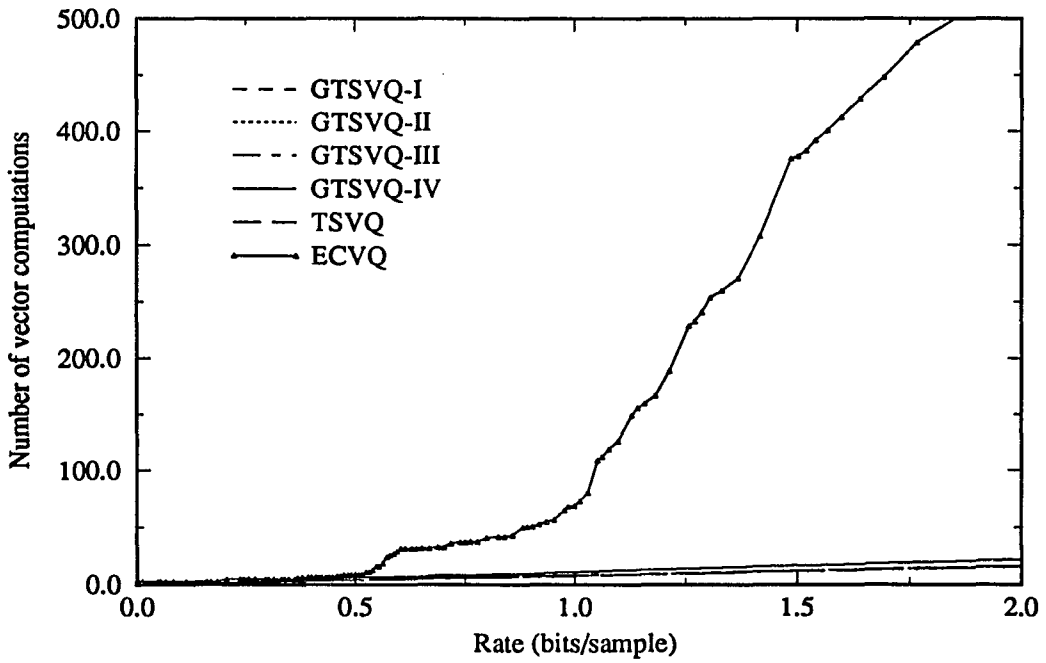


Figure 6.6: Computational requirement versus code rate of the GTSVQ, traditional TSVQ, and ECVQ algorithms for a Gauss-Markov source with $\rho = 0.9$

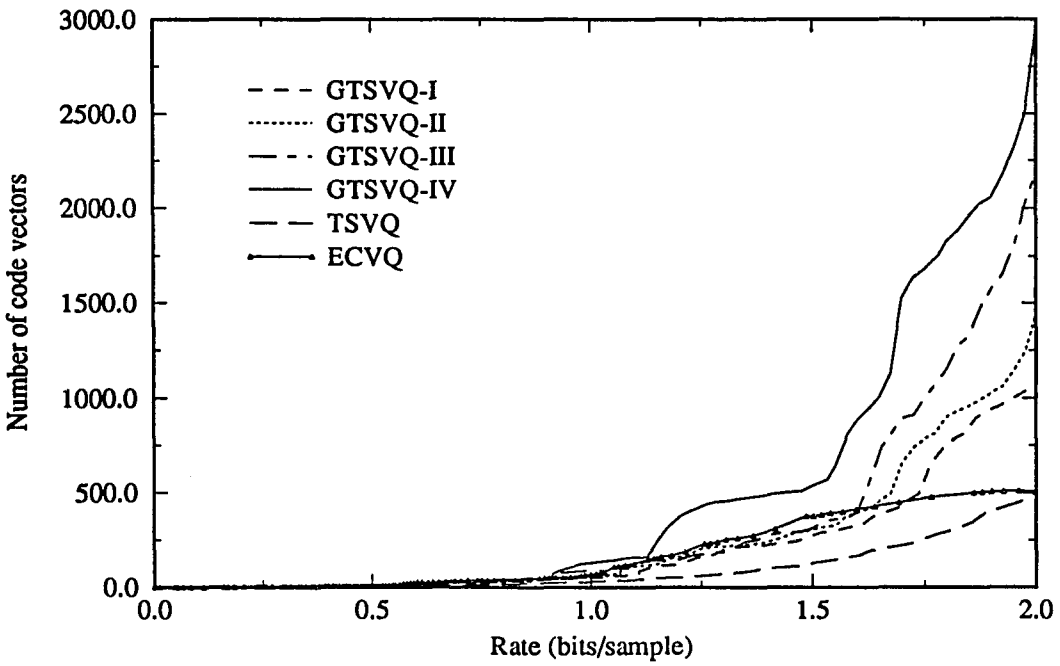


Figure 6.7: Memory requirement versus code rate of the GTSVQ, traditional TSVQ, and ECVQ algorithms for a Gauss-Markov source with $\rho = 0.9$

ments remain roughly unchanged. This can be verified by examining the curves given in Figure 6.6. The memory requirements of the new algorithms, however, are significantly increased, as being evident from Figure 6.7. Therefore, it can be concluded that the GTSVQ algorithms increase the memory requirement in tradeoff for the improved rate-distortion performance.

6.4.2 Quantization of a set of digitized images

The four new GTSVQ algorithms have also been tested against other VQ algorithms on the 15 medical MR images used in the first part of this study. Like the experiments summarized in Figures 5.6 and 5.7, 4 out of the 15 images were used to form the training vector sequence in this case and the remainders were used as test images. Also, the vector dimension was fixed at 16, and each of the input (training or test) vectors contained a contiguous block of 4×4 pixels of an image. The rate-distortion performances and the computational and memory requirements of the GTSVQ algorithms evaluated from the experiments are shown in Figures 6.8, 6.9, and 6.10, respectively, together with the corresponding results of the traditional TSVQ algorithm (with and without entropy coding of the code vector indices).

From Figure 6.8, it can be clearly seen that all the new algorithms achieved significantly improved performances over the traditional TSVQ algorithm for the images under consideration. Also, the improvement increases as the code rate increases with the difference in SQNR between the GTSVQ-IV and the traditional TSVQ algorithms going up to as high as about 15 dB for code rates in the range from 0.4 to 0.5 bit per pixel. Once again, the four GTSVQ algorithms successively outperform the preceding algorithms.

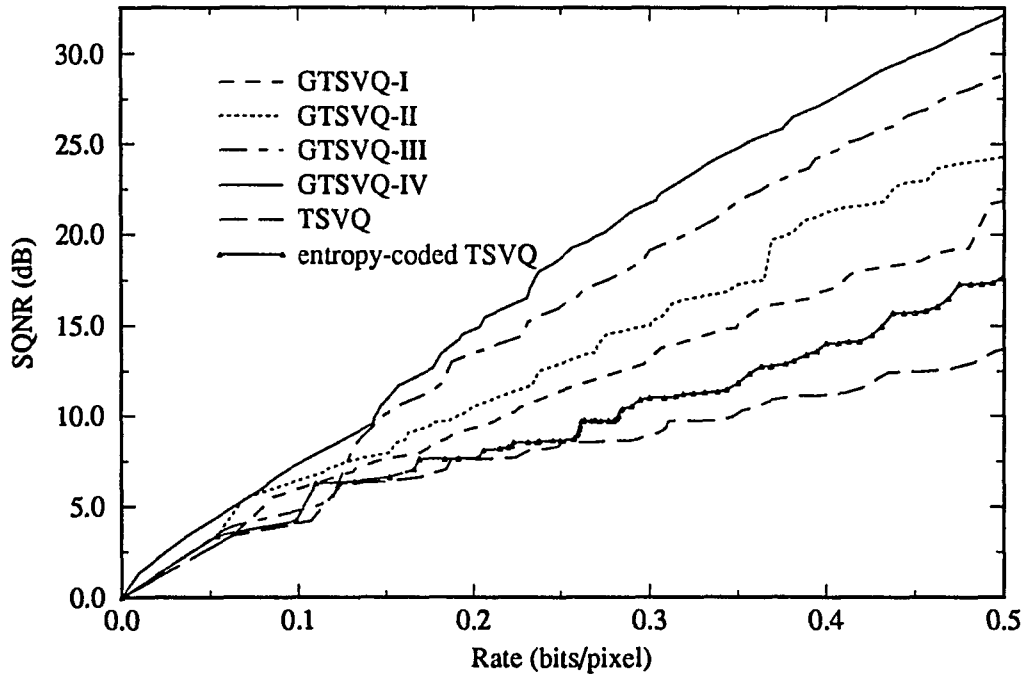


Figure 6.8: SQNR versus code rate of the GTSVQ and traditional TSVQ algorithms for a set of training images

An examination of Figure 6.9 indicates that while the GTSVQ-I algorithm can achieve better performance than the traditional TSVQ algorithm, its computational complexity remains unchanged as expected for any given code rate. The other three GTSVQ algorithms, however, require more computations for quantizing an input vector in tradeoff for the improved performance. Moreover, as implied by the curves given in Figure 6.10, the memory requirements of all the new algorithms are substantially increased.

For the purpose of comparison, the curves in Figures 6.8, 6.9, and 6.10 corresponding to the GTSVQ algorithms are redrawn in Figures 6.11, 6.12, and 6.13, respectively. Also shown in the figures are the corresponding results of the ECVQ

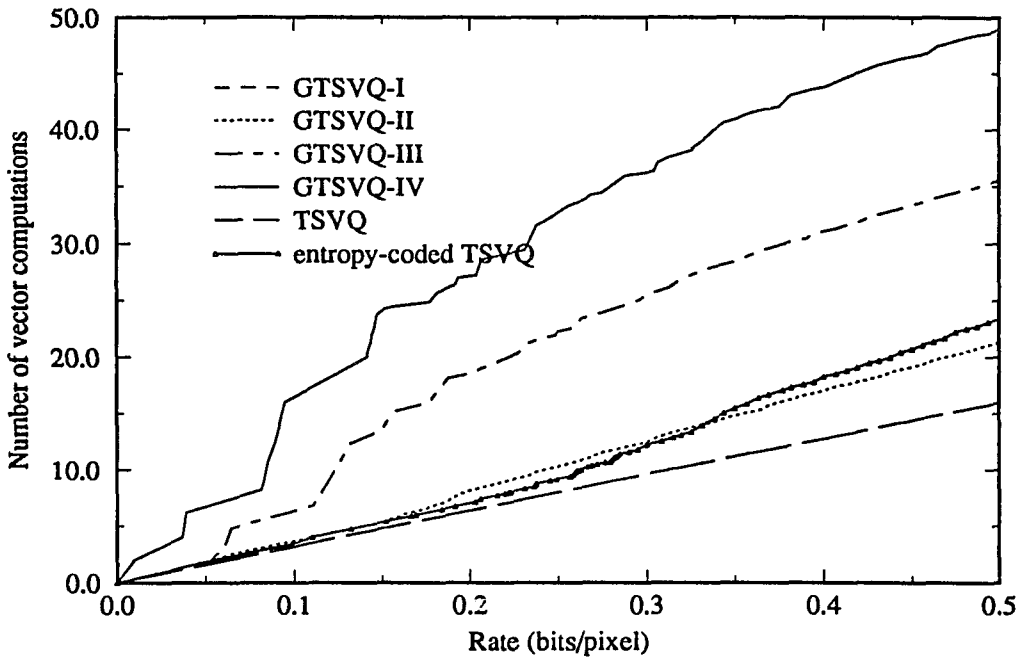


Figure 6.9: Computational requirement versus code rate of the GTSVQ and traditional TSVQ algorithms for a set of training images

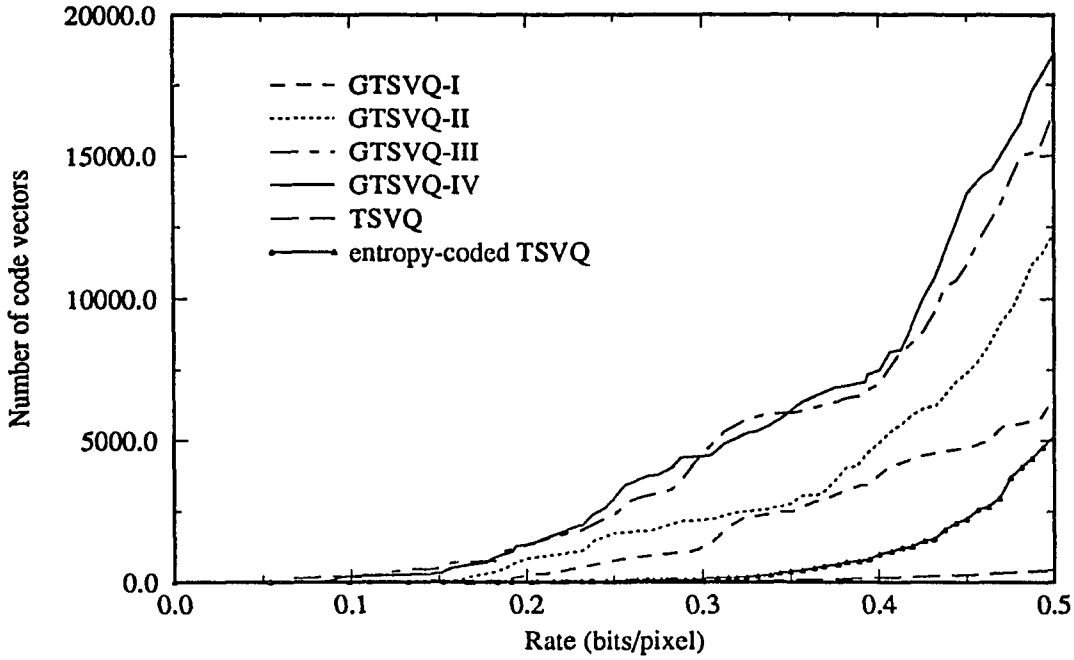


Figure 6.10: Memory requirement of the GTSVQ and traditional TSVQ algorithms for a set of training images

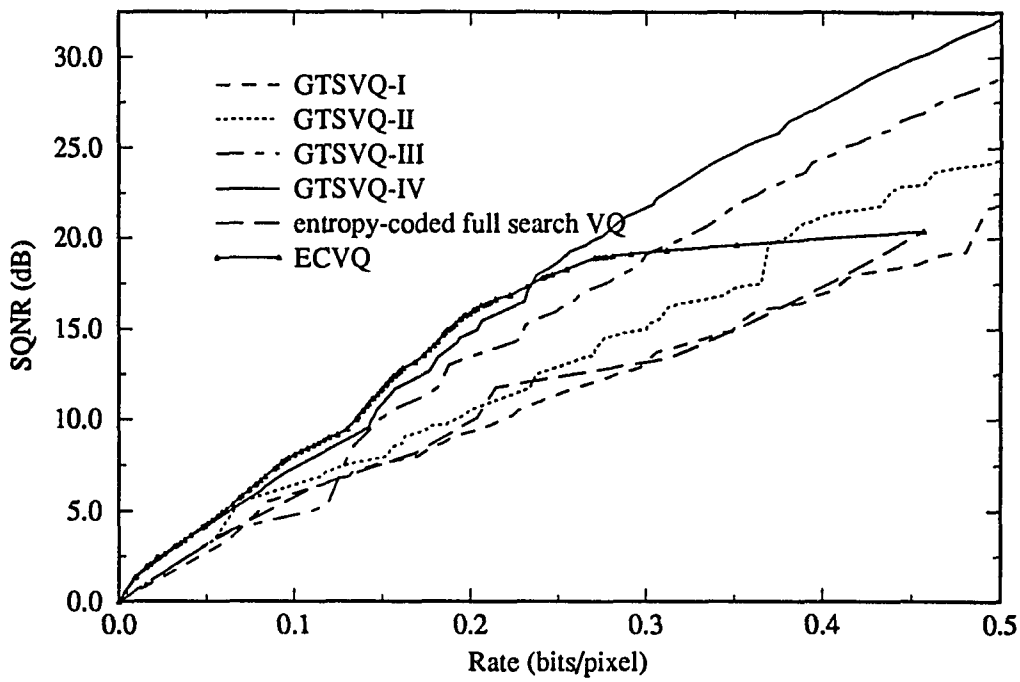


Figure 6.11: SQNR versus code rate of the GTSVQ, full search VQ, and ECVQ algorithms for a set of training images

(with initial vector codebook size of 512) and full search VQ algorithms already presented in Figures 5.6 and 5.7. From Figure 6.11, it can be seen that for code rates in the range from 0 to 0.25 bit per pixel, the GTSVQ-III and GTSVQ-IV systems closely follow the ECVQ system in improving the SQNR as code rate increases. Beyond that range, the two ECGTSVQ systems keep increasing the SQNR as code rate increases whereas the ECVQ system drops out from the race due to the limited vector codebook size. If the ECVQ system were designed with a larger initial codebook size, it would have been able to hold the SQNR improvement trend for higher code rates. In that case, however, its computational complexity would also be increased. An inspection of the curves given in Figure 6.12 reveals that even in the current case,

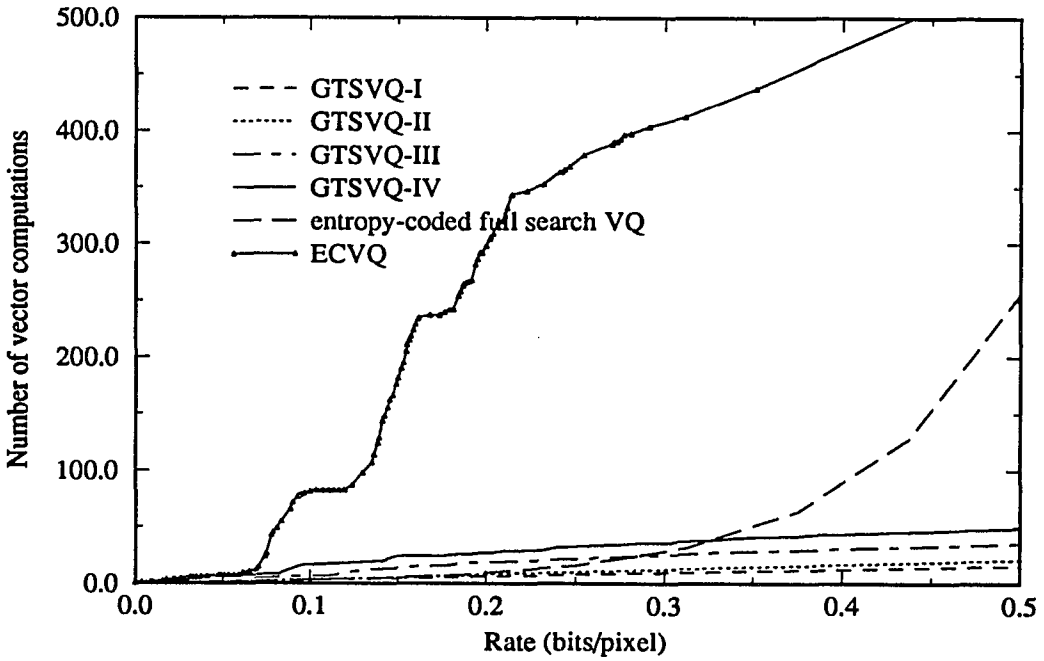


Figure 6.12: Computational requirement versus code rate of the GTSVQ, full search VQ, and ECVQ algorithms for a set of training images

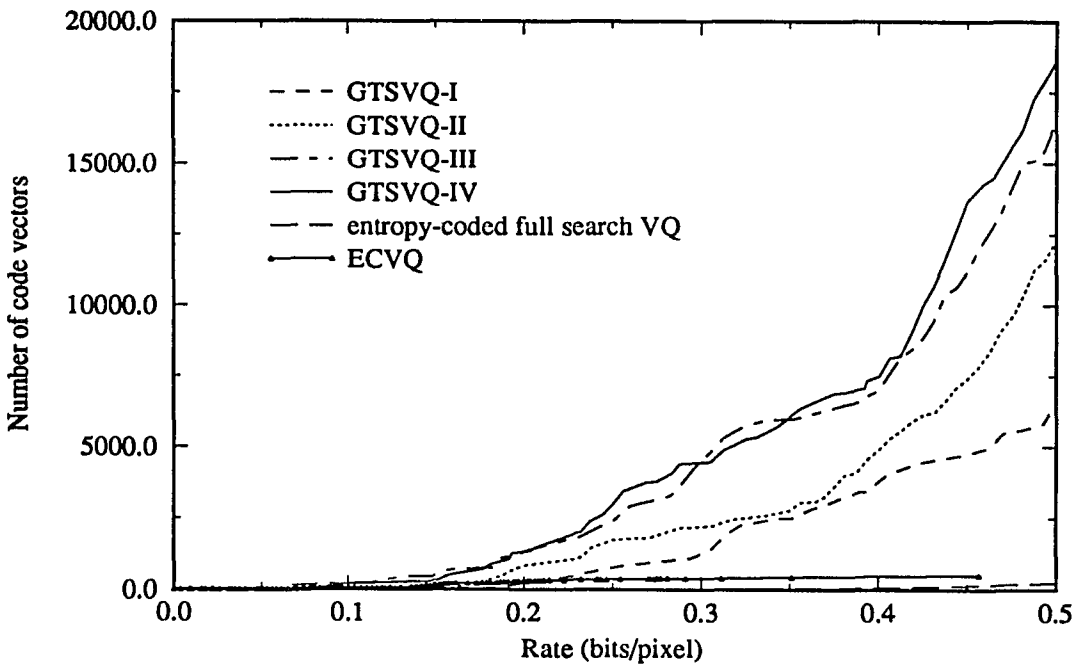


Figure 6.13: Memory requirement versus code rate of the GTSVQ, full search VQ, and ECVQ algorithms for a set of training images

the ECVQ system has already required much more computations than the GTSVQ systems.

Although the computational complexities of the GTSVQ algorithms are in general higher than that of the traditional TSVQ algorithm, they are still in a roughly linear relationship with the code rate. This can be easily verified by inspecting the curves given in Figures 6.9 and 6.12. Also, the complexities are much lower than that of the full search VQ algorithm. As a tradeoff for the improved rate-distortion performance and reduced computational complexity, the GTSVQ algorithms require much more memory locations to store the tree-structured vector codebook (see Figures 6.10 and 6.13).

While all the experimental results presented above were obtained from the design processes of different VQ systems based on the training vector sequences, Figure 6.14 shows the SQNR curves resulting from the actual quantizing operations of the GTSVQ-IV system on a training image and a test image together with the curve corresponding to the design process. From the figure, it is seen that the three SQNR curves closely follow each other. This, in turn, implies that the set of images used in the experiments is well represented by the training images and the GTSVQ-IV system has been appropriately designed for the images. To illustrate the subjective quality of the reconstructions obtained using GTSVQ-IV, the original version of the test image and the reconstructed versions at different code rates are shown in Figures 6.15 to 6.20, respectively.

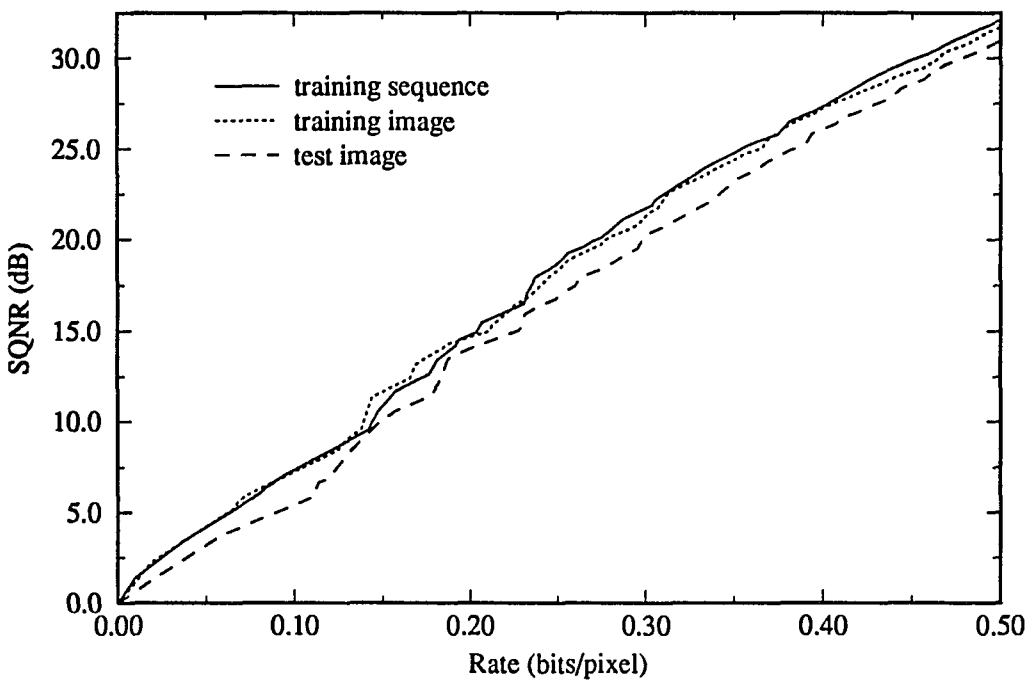


Figure 6.14: SQNR versus code rate of the GTSVQ-IV algorithm for a training image and a test image

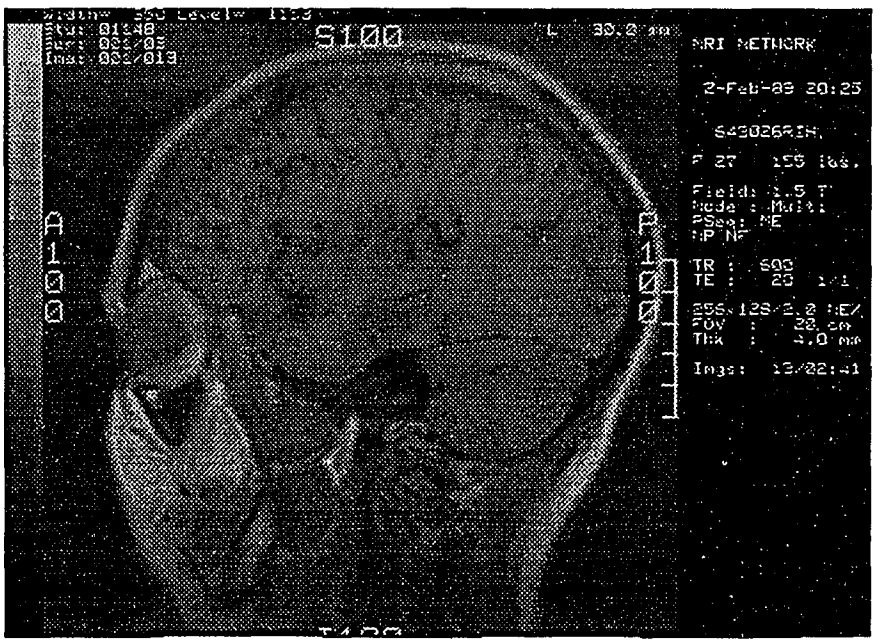


Figure 6.15: Test image: original version

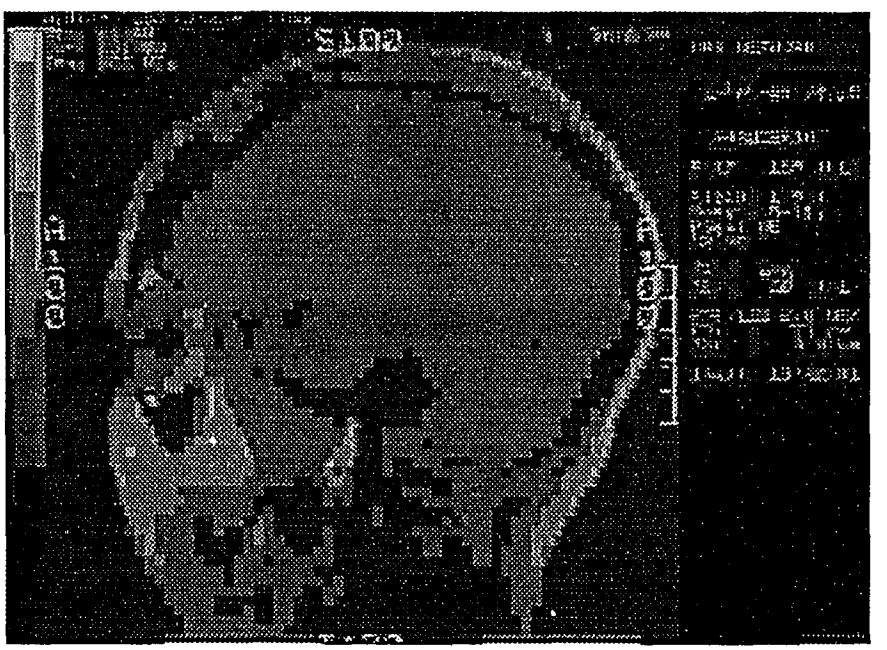


Figure 6.16: Test image compressed using the GTSVQ-IV algorithm: reconstructed version at code rate of 0.11 bits/pixel and PSNR of 20.18 dB

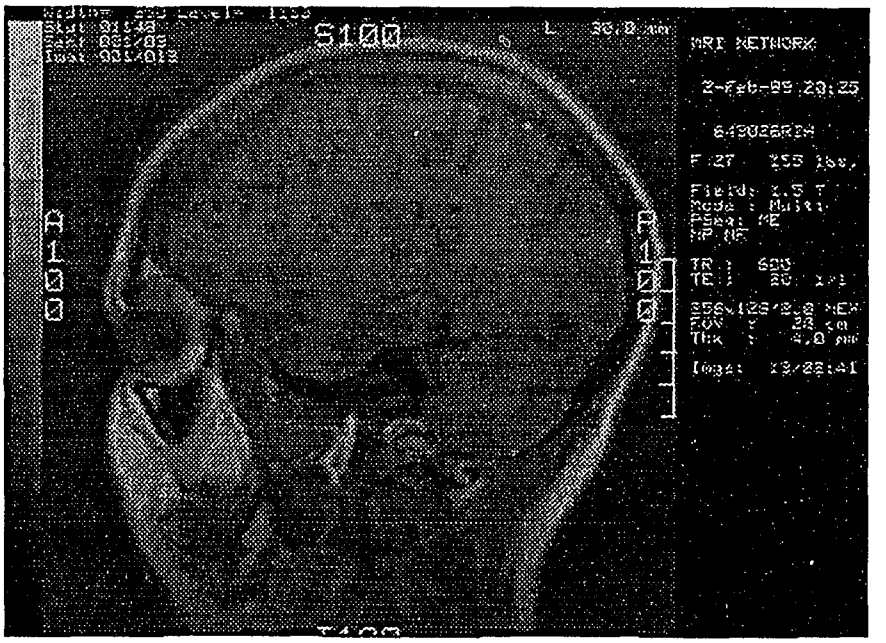


Figure 6.17: Test image compressed using the GTSVQ-IV algorithm: reconstructed version at code rate of 0.2 bits/pixel and PSNR of 28.37 dB

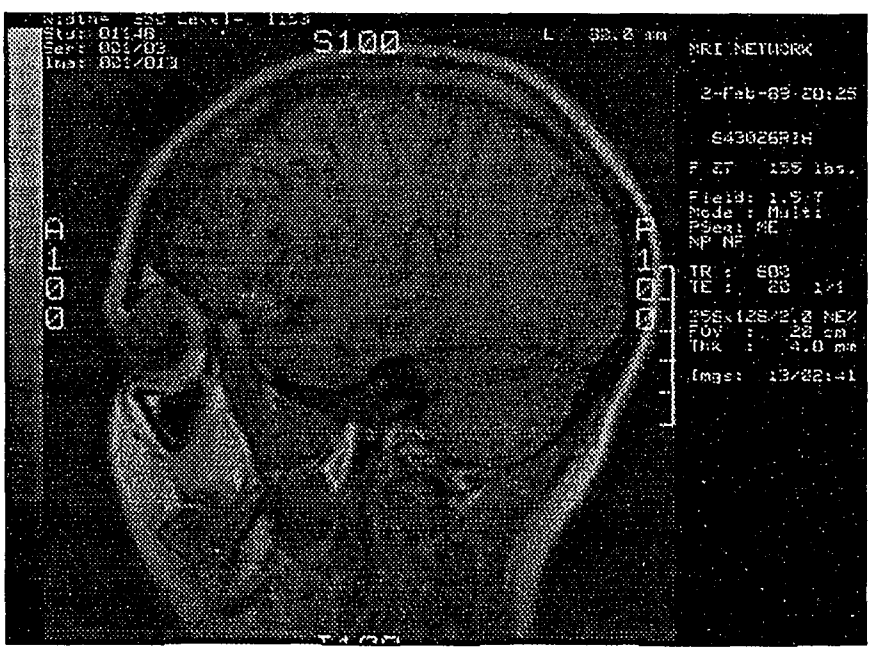


Figure 6.18: Test image compressed using the GTSVQ-IV algorithm: reconstructed version at code rate of 0.3 bits/pixel and PSNR of 34.58 dB

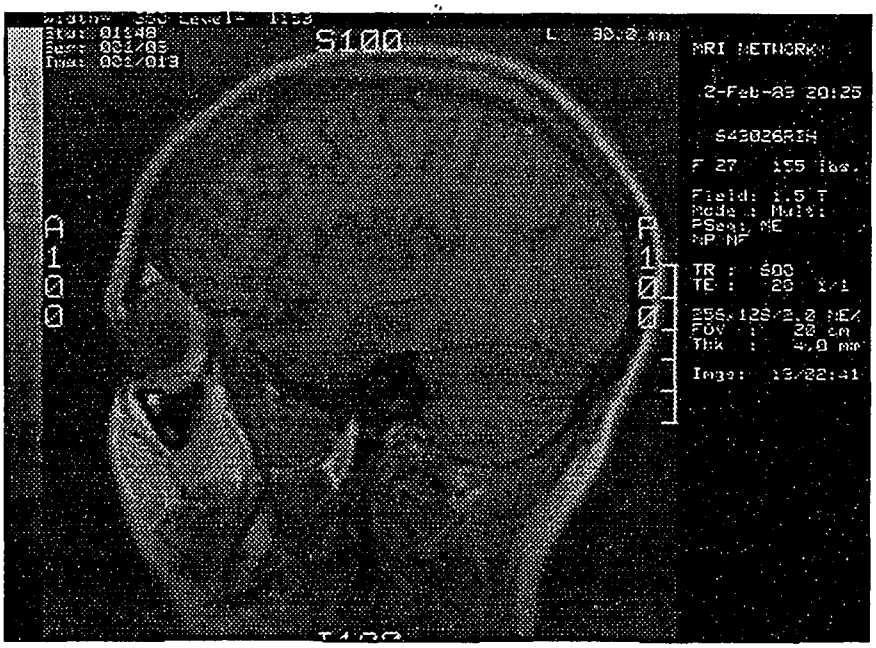


Figure 6.19: Test image compressed using the GTSVQ-IV algorithm: reconstructed version at code rate of 0.4 bits/pixel and PSNR of 40.42 dB

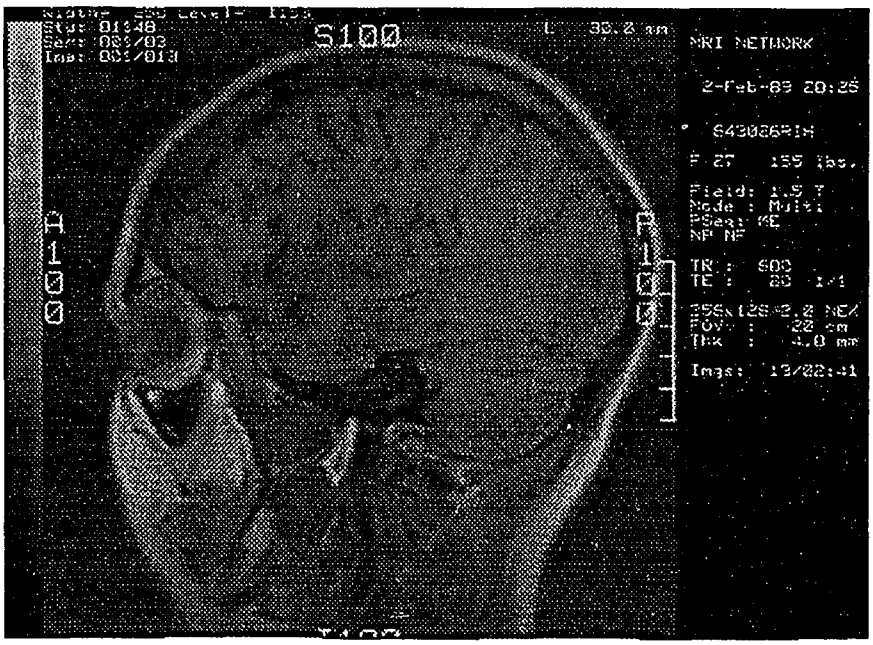


Figure 6.20: Test image compressed using the GTSVQ-IV algorithm: reconstructed version at code rate of 0.5 bits/pixel and PSNR of 45.29 dB

6.5 Summary

Four new tree-structured VQ algorithms have been described in this chapter. All the four algorithms follow the same approach to the design of variable-rate TSVQ systems, *i.e.*, grow an unbalanced tree only one node at a time by always splitting the terminal node that results in the largest ratio ϕ of decrease in distortion ΔD to increase in rate ΔR. The algorithms differ in two respects of the tree growing procedure: 1) the definition of rate in the calculation of ΔR and, 2) the algorithm used in the design of terminal node splitting, as well as in whether an entropy code is required to encode the code vector indices in the practical quantizing operations. A summary of these differences is given in Table 6.1.

The four new algorithms are somewhat more complicated than the straightforward traditional TSVQ algorithm, especially the tree-structured vector codebook design procedure, but they are more advantageous in the rate-distortion sense and can achieve substantially better rate-distortion performance. From experimental

Table 6.1: Summary of differences between GTSVQ algorithms

	GTSVQ-I	GTSVQ-II	GTSVQ-III	GTSVQ-IV
Definition of Rate	average length of code vector indices	average length of code vector indices	entropy of code vector indices	entropy of code vector indices
Algorithm Used to Design Splits	GL algorithm	GL algorithm	GL algorithm	ECVQ algorithm
Entropy Coding of Code Vector Indices	no	yes	yes	yes

results presented in the last section, it is seen that, starting from the improvement achieved by GTSVQ-I over the traditional TSVQ algorithm, the four new algorithms successively outperform the preceding algorithms, and the two entropy-constrained GTSVQ algorithms can even achieve better rate-distortion performances than the full search VQ algorithm. Moreover, the experimental results have also shown that the ECGTSVQ algorithms possess the capability of holding the distortion reduction trend closely comparable with the well-appreciated ECVQ algorithm as code rate increases.

As far as computational and memory requirements are concerned, the GTSVQ algorithms retain the low complexity tree-structured search characteristic of the traditional TSVQ algorithm. Consequently, the computational requirements of these algorithms remain the same or are slightly higher than that of the traditional algorithm, but are much lower than those of the full search VQ and the ECVQ algorithms. As a tradeoff for the improved performance, the new algorithms usually require more memory space than the traditional algorithm as well as other VQ algorithms to store their tree-structured vector codebooks. These claims are supported by experiments that have been conducted.

Lastly, it should be mentioned that the new GTSVQ algorithms all inherit the property of the traditional TSVQ algorithm that the code vector indices are embedded. Therefore, they are suitable for use in progressive transmission applications, like the traditional algorithm.

7. CONCLUDING REMARKS

7.1 Summary and Discussions

The compression of digitized images using both quantization and variable-length coding has been investigated in this study. In the first part of study, the problem of scalar compression was considered and a new scheme, *viz.*, ECDPCM, that combines both lossy and lossless DPCM methods into a common framework was developed. This new scheme can be regarded as an enhanced version of the traditional entropy-coded DPCM method. It uses available results on the design of linear predictors and scalar quantizers that incorporate properties of human visual perception. However, instead of coding the quantized prediction errors using a memoryless model as in the traditional DPCM method, a statistical source model with multiple contexts is employed in the new scheme in order to capture the inter-sample dependencies present in the sequence of the quantized prediction errors so as to achieve improved compression performance. For a given image, the contexts of the source model are defined based on the estimated horizontal and vertical gradients as well as the predicted gray-level value of a pixel or, more specifically, are defined by a partition of the three-dimensional vector space formed by the reconstructions of the north, west, and northwest neighbors of the pixel. These contexts are generated adaptively along with the coding process by successively splitting the existing contexts that meet

some appropriate criteria. As a result, the structure of the generated contexts can be viewed as a variable-depth binary tree, and each terminal node of the tree or, more accurately, the bit pattern corresponding to the path from the root to the terminal node, represents a context. In accordance with the adaptive generation of contexts, the statistical parameters of the context-based source model, *i.e.*, the conditional probability distribution of the quantized prediction error values under each context, are also calculated adaptively. The quantized prediction errors are then coded using these contextual statistics with the powerful arithmetic coding technique. Experimental results have shown that the new scheme can provide compression factors in the range from 4 to 11 with a peak SNR of about 50 dB for 8-bit medical images, and the use of context-based source model can improve compression performance by about 25% to 35%. While higher performance is achieved by this new scheme over the the traditional method, the computational complexity of the compression operation, as a tradeoff, is increased by about 20% to 40%. Moreover, the memory requirement is also increased for maintaining the large number of statistical parameters of the context-based source model.

The second part of study was devoted to the problem of image compression using tree-structured vector quantization. As a result of the study, a new design method referred to as GTSVQ for vector codebook generation was developed together with four different implementation algorithms. In this new method, an unbalanced binary tree-structured vector codebook is designed in a greedy fashion under the constraint of rate-distortion tradeoff. More specifically, the tree is grown one node at a time starting from scratch by always splitting the terminal node that results in the largest ratio of decrease in distortion to increase in rate. In order to reduce the distortion even

more at each node split and thereby achieve better rate-distortion performance, the entropy-constrained vector quantization (ECVQ) design technique is also used in one of the implementation algorithms for designing the splits of terminal nodes. The tree-structured vector codebooks designed by the new method can be used to implement variable-rate VQ compression systems and, due to the embedded property of the code vector indices, these systems can used for progressive transmission applications. From experiments, it has been found that all of the four GTSVQ algorithms can achieve better rate-distortion performance over the traditional TSVQ algorithm, especially for the images used in the experiments. These performances are close to or better than that of full search VQ algorithm, and the algorithm that incorporates ECVQ in the design of node splits can even hold the distortion reduction trend closely comparable with the full search ECVQ algorithm as average code rate increases. Also, the GTSVQ algorithms retain the low complexity characteristic of the traditional TSVQ algorithm and, therefore, are much more computationally efficient than the ECVQ and full search VQ algorithms. As a tradeoff for the improved rate-distortion performance and reduced computational complexity, the memory requirements of these new algorithms are considerably higher than those of many traditional VQ algorithms. With memory prices falling the way it has been in recent years, however, this should not present any major concern.

Comparatively speaking, the ECDPCM method developed from the first part of study is more suitable for applications where high-fidelity reconstruction is desired, while GTSVQ from the second part is preferable to low rate applications. The two new methods are however closely related to each other. This relation can be more easily identified by considering the following extension of the GTSVQ algorithms.

Suppose an image has already been compressed and transmitted by using one of the GTSVQ algorithms and it is decided that higher fidelity or even lossless reconstruction is needed. In this case, one may calculate the difference between a vector $\mathbf{x}$ being quantized and its code vector $\mathbf{y}_t$, *i.e.*, the reconstruction error vector $\mathbf{e} = \mathbf{x} - \mathbf{y}_t$, and then quantize each component of the error vector $\mathbf{e}$ in scalar fashion. The quantized error components can be subsequently encoded for further transmission by using an arithmetic code and treating the index of the code vector $\mathbf{y}_t$ as the context. With this extension, it becomes clear that the GTSVQ method can be viewed as a generalization of the ECDPCM method: in ECDPCM, the predicted value for each pixel is formed by the reconstructions of its north, west, and northwest neighbors and the contexts of the source model for the pixel are defined by a partition of the three-dimensional vector space made up with the three reconstructions, whereas in GTSVQ, the VQ reconstructed value is taken as the predicted value for a pixel and the contexts of the source model are defined by a partition of the K-dimensional vector space formed by all the components within a vector. The contexts in both methods are tree structured and are both generated through a growing procedure by splitting the terminal nodes under some constraints. The major difference between the two methods is that in ECDPCM the contexts need not be transmitted or stored whereas in GTSVQ the contexts or, more precisely, the indices or identifications of the contexts are required to be transmitted or stored which, in turn, can be used to produce a first stage of reconstruction.

It should be mentioned that an algorithm similar to GTSVQ-I has been introduced by Riskin in her doctoral dissertation [92, 94, 26], which occurred independently to the author of this dissertation, who was unaware of Riskin's result when the study

reported here was pursued. Although the two algorithms are very similar, the approach that Riskin took in her investigation is quite different from what this author followed. Basically, Riskin considered the problem of designing a tree-structured VQ codebook beginning with ideas from classification and regression tree design [15] whereas the starting point of development of the GTSVQ algorithms presented here originated from the work done in developing the ECDPCM method.

7.2 Recommendations for Future Investigation

Some interesting issues remain open in the two proposed image compression methods. One of these issues is the incorporation of a more reasonable human perception model into the design of compression systems. This is particularly important for applications where the GTSVQ algorithms are applied and very limited communication resources are available. Moreover, a model that can represent the human perception properties for a wide range of distortion levels should be used in the GTSVQ algorithms for designing progressive transmission systems.

Another issue is the application of the constrained tree growing principle embodied in the GTSVQ algorithms to the design of unbalanced and nonuniform tree-structured VQ systems, *i.e.*, the number of immediate descendents resulting from the split of a terminal node is allowed to be more than two so as to be more flexible to attain higher ratio of decrease in distortion to increase in rate. It is expected that such a design procedure would result in VQ systems for which the memory requirements are reduced at the cost of increased computational complexity. Therefore, a suitable measure of the tradeoff between memory requirement and computational complexity as well as rate-distortion performance needs to be developed and incorporated into

the nonuniform GTSVQ design procedure.

Yet a third possible investigation is to apply the pruned TSVQ (PTSVQ) algorithm to prune back the trees designed by the GTSVQ algorithms. With such a pruning operation, it is expected that the resulting VQ systems can achieve even better rate-distortion performance while being more computationally efficient as well as requiring less memory locations.

BIBLIOGRAPHY

[1] Abramson, N. *Information Theory and Coding.* McGraw-Hill, Inc., New York, 1963.

[2] Adoul, J.-P., Debray, J.-L., and Dalle, D. "Spectral distance measure applied to the optimum design of DPCM coders with L predictors." *Proceedings of the 1980 IEEE International Conference on Acoustics, Speech, and Signal Processing* (April 1980): 512-515.

[3] Ahmed, N. and Rao, K. R. *Orthogonal Transforms for Digital Signal Processing.* Springer-Verlag, Berlin, 1975.

[4] Antonini, M., Barlaud, M., Mathieu, P., and Daubechies, I. "Image coding using wavelet transform." *IEEE Transactions on Image Processing,* IP-1, No. 2 (April 1992): 205-220.

[5] Atal, B. S. and Schroeder, M. R. "Predictive coding of speech signals." *The Bell System Technical Journal,* 49, No. 10 (October 1970): 1973-1986.

[6] Baker, R. L. and Gray, R. M. "Image compression using non-adaptive spatial vector quantization." *Conference Record of the 16th Asilomar Conference on Circuits Systems and Computers* (October 1982): 55-61.

[7] Baker, R. L. and Gray, R. M. "Differential vector quantization of achromatic imagery." *Proceedings of the International Picture Coding Symposium* (March 1983).

[8] Barnes, C. F. and Frost, R. L. "Necessary conditions for the optimality of residual vector quantizers." *Abstracts of the 1990 IEEE International Symposium on Information Theory* (January 1990): 34.

[9] Bell, T. C., Cleary, J. G., and Witten, I. H. *Text Compression.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1990.

[10] Berger, T. *Rate Distortion Theory.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

[11] Berger, T. "Optimum quantizers and permutation codes." *IEEE Transactions on Information Theory,* IT-18, No. 6 (November 1972): 759-765.

[12] Berger, T. "Minimum entropy quantizers and permutation codes." *IEEE Transactions on Information Theory,* IT-28, No. 2 (March 1982): 149-157.

[13] Blahut, R. E. "Computation of channel capacity and rate-distortion functions." *IEEE Transactions on Information Theory,* IT-18, No. 4 (July 1972): 460-473.

[14] Bradley, J. "xv - Interactive image display for the X-Window system." JPEG i/o code provided by the independent JPEG group.

[15] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees.* Wadsworth International Group, Belmont, California, 1984.

[16] Burt, P. J. and Adelson, E. H. "The Laplacian pyramid as a compact image code." *IEEE Transactions on Communications,* COM-31, No. 4 (April 1983): 552-540.

[17] Buzo, A., Gray, A. H., Jr., Gray, R. M., and Markel, J. D. "Speech coding based upon vector quantization." *IEEE Transactions on Acoustics, Speech, and Signal Processing,* ASSP-28, No. 10 (October 1980): 562-574.

[18] Chan, W.-Y. and Gersho, A. "Constrained-storage quantization of multiple vector sources by codebook sharing." *IEEE Transactions on Communications,* COM-38, No. 1 (January 1991): 11-13.

[19] Chen, K. "Adaptive Source Models for Reversible Compression of Digitized Images." *M.S. thesis,* Iowa State University, Ames, Iowa, 1991.

[20] Chen, K. and Ramabadran, T. V. "Reversible compression of medical images with adaptive context selection." *Medical Imaging 1993: Image Capture, Formatting, and Display,* Yongmin Kim, Editor, Proc. SPIE 1897 (1993): 503-510.

[21] Chou, P. A., Lookabaugh, T., and Gray, R. M. "Entropy-constrained vector quantization." *IEEE Transactions on Acoustics, Speech, and Signal Processing,* ASSP-37, No. 1 (January 1989): 31-42.

[22] Chou, P. A., Lookabaugh, T., and Gray, R. M. "Optimal pruning with applications to tree-structured source coding and modeling." *IEEE Transactions on Information Theory,* IT-35, No. 2 (March 1989): 299-315.

[23] Clarke, R. J. *Transform Coding of Images.* Academic Press, Orlando, Florida, 1985.

[24] Cleary, J. G. and Witten, I. H. "A comparison of enumerative and adaptive codes." *IEEE Transactions on Information Theory,* IT-30, No. 2 (March 1984): 306-315.

[25] Cleary, J. G. and Witten, I. H. "Data communication using adaptive coding and partial string matching." *IEEE Transactions on Communications,* COM-32, No. 4 (April 1984): 396-402.

[26] Cosman, P. C., Oehler, K. L., Riskin, E. A., and Gray, R. M. "Using vector quantization for image processing." *Proceedings of the IEEE,* 81, No. 9 (September 1993): 1326-1341.

[27] Cuperman, V. and Gersho, A. "Adaptive differential vector coding of speech." *Proceedings of IEEE Global Telecommunications Conference* (November 1982): 1092-1096.

[28] Daubechies, I. *Ten Lectures on Wavelets.* SIAM, Philadelphia, Pennsylvania, 1992.

[29] Elias, E. "Predictive coding - Part I and Part II." *IRE Transactions on Information Theory,* IT-1, No. 1 (March 1955): 16-33.

[30] Equitz, W. H. "Fast algorithms for vector quantization picture coding." *Proceedings of the 1987 IEEE International Conference on Acoustics, Speech, and Signal Processing* (April 1987): 725-728.

[31] Equitz, W. H. "A new vector quantization clustering algorithm." *IEEE Transactions on Acoustics, Speech, and Signal Processing,* ASSP-37, No. 10 (October 1989): 1568-1575.

[32] Farvardin, N. and Modestino, J. W. "Optimum quantizer performance for a class of non-gaussian memoryless sources." *IEEE Transactions on Information Theory,* IT-30, No. 3 (May 1984): 485-497.

[33] Fischer, T. R. and Tinnen, D. J. "Quantized control with differential pulse code modulation." *Proceedings of the 21st Conference on Decision and Control* (December 1982): 1222-1227.

[34] Foster, J., Gray, R. M., and Dunham, M. o. "Finite-state vector quantization for waveform coding." *IEEE Transactions on Information Theory,* IT-31, No. 3 (May 1985): 348-359.

[35] Frost, R. L., Barnes, C. F., and Xu, F. "Design and performance of residual quantizers." *Proceedings of Data Compression Conference,* (April 1991): 129-138.

[36] Gallager, R. G. *Information Theory and Reliable Communication.* John Wiley and Sons, Inc., New York, 1968.

[37] Gersho, A. and Cheng, D. "Fast nearest neighbor search for nonstructured Euclidean codes." *Abstracts of the 1983 IEEE International Symposium on Information Theory* (September 1983): 88.

[38] Gersho, A. and Shoham, Y. "Hierarchical vector quantization of speech with dynamic codebook allocation." *Proceedings of the 1984 IEEE International Conference on Acoustics, Speech, and Signal Processing* (March 1984): 10.9.1-10.9.4.

[39] Goldberg, M. and Wang, L. "Comparative performance of pyramid data structures for progressive image transmission." *IEEE Transactions on Communications,* COM-39, No. 4 (April 1991): 540-548.

[40] Gersho, A. and Gray, R. M. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, Boston, Massachusetts, 1992.

[41] Gray, R. M., Kieffer, J. C., and Linde, Y. "Locally optimal block quantizer design." *Information and Control,* 45, No.2 (May 1980): 178-198.

[42] Gray, R. M., Buzo, A., Gray, A. H., Jr., and Matsuyama, Y. "Distortion measures for speech processing." *IEEE Transactions on Acoustics, Speech, and Signal Precessing,* ASSP-28, No. 8 (August 1980): 367-376.

[43] Gray, R. M. and Karnin, E. "Multiple local optima in vector quantizers." *IEEE Transactions on Information Theory,* IT-28, No. 2 (March 1982): 256-261.

[44] Gray, R. M. "Vector quantization." *IEEE ASSP Magazine,* 1, No. 2 (April 1984): 4-29.

[45] Gray, R. M. *Probability, Random Processes, and Eryodic Properties.* Springer-Verlag, New York, New York, 1988.

[46] Gray, R. M. *Source Coding Theory.* Kluwer Academic Publishers, Boston, Massachusetts, 1990.

[47] Guazzo, M. "A general minimum-redundancy source-coding algorithm." *IEEE Transactions on Information Theory,* IT-26, No. 1 (January 1980): 15-25.

[48] Hang, H.-M. and Haskell, B. "Interpolative vector quantization of color images." *IEEE Transactions on Communications,* COM-36, No. 4 (April 1988): 465-470.

[49] Huffman, D. A. "A method for the construction of minimum redundancy codes." *Proceedings of the IRE,* 40, No. 9 (September 1952): 1098-1101.

[50] Jain, A. K. "Image data compression." *Proceedings of the IEEE,* 69, No. 3 (March 1981): 349-389.

[51] Jain, A. K., Farrelle, P. M., and Algazi, V. R. "Image data compression." *Digital Image Processing Techniques,* 171-226, Academic Press, 1984.

[52] Jain, A. K. *Fundamentals of Digital Image Processing.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989.

[53] Jayant, N. S. and Noil, P. *Digital Coding of Waveforms - Principles and Applications to Speech and Video.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[54] Jayant, N. S., Johnston, J., and Safranek, R. "Signal compression based on models of human perception." *Proceedings of the IEEE,* 81, No. 10 (October 1993): 1385-1422.

[55] Jelinek, F. and Anderson, J. B. "Instrumentable tree encoding of information sources." *IEEE Transactions on Information Theory,* IT-17, No. 1 (January 1971): 118-119.

[56] Jones, C. B. "An efficient coding system for long source sequences." *IEEE Transactions on Information Theory,* IT-27, No. 3 (May 1981): 280-291.

[57] Juang, B.-H. and Gray, A. H., Jr. "Multiple stage vector quantization for speech coding." *Proceedings of the 1982 IEEE International Conference on Acoustics, Speech, and Signal Processing* (April 1982): 597-600.

[58] Le Gall, D. "MPEG: A video compression standard for multimedia applications." *Communications of the ACM,* 34, No. 4 (April 1991): 46-63.

[59] Kieffer, J. C. "Stochastic stability for feedback quantization schemes." *IEEE Transactions on Information Theory,* IT-28, No. 2 (March 1982): 248-254.

[60] Kirkpartrick, S., Gellatt, C. D., Jr., and Vecchi, M. P. "Optimization by simulated annealing." *Science,* 220, No. 4598 (May 1983): 671-680.

[61] Knowlton, K. "Progressive transmission of grey-scale and binary pictures by simple, efficient, and lossless encoding schemes." *Proceedings of IEEE,* 68, No. 7 (July 1980): 885-896.

[62] Kohonen, T. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, 1984.

[63] Kolmogorov, A. N. "On the Shannon theory of information in the case of continuous signals." *IRE Transactions on Infomation Theory,* IT-2, No. 1 (January 1956): 102-108.

[64] Langdon, G. G., Jr. "An introduction to arithmetic coding." *IBM Journal of Research and Development,* 28, No. 2 (March 1984): 135-149.

[65] Linde, Y., Buzo, A., and Gray, R. M. " An algorithm for vector quantizer design." *IEEE Transactions on Communications,* COM-28, No. 1 (January 1980): 84-95.

[66] Lloyd, S. P. "Least squares quantization in PCM." *IEEE Transactions on Information Theory,* IT-28, No. 2 (March 1982): 127-135.

[67] Makhoul, J. "Linear prediction: A tutorial review." *Proceedings of the IEEE,* 63, No. 4 (April 1975): 561-580.

[68] Makhoul, J., Roucos, S., and Gish, H. "Vector quantization in speech coding." *Proceedings of the IEEE,* 73, No. 11 (November 1985): 1551-1588.

[69] MacQueen, J. "Some methods for classification and analysis of multivariate observations." *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics, and Probability,* 1 (1967): 281-296.

[70] Maragos, P. A., Schafer, R. W. and Mersereau, R. M. "Two-dimensional linear prediction and its application to adaptive predictive coding of images." *IEEE Transactions on Acoustics, Speech, and Signal Precessing,* ASSP-32, No. 6 (December 1984): 1213-1229.

[71] Hetherington, M. D. and Ramabadran, T. V. "Lossless compression of codebook indices in image vector quantization." *IEEE Transactions on Communications,* under review.

[72] Mallat, S. G. "A theory for multiresolution signal decomposition: The wavelet representation." *IEEE Transactions on Pattern Analysis and Machine Intelligence,* PAMI-11, No. 7 (July 1989): 674-693.

[73] Max, J. "Quantizing for minimum distortion." *IRE Transactions on Information Theory,* IT-6, No. 2 (March 1960): 7-12.

[74] Meyer, Y. *Wavelets: algorithms and applications.* SIAM, Philadelphia, Pennsylvania, 1993.

[75] Murakami, T., Asai, K., and Yamazaki, E. "Vector quantizer of video signals." *Electronics Letters,* 7 (November 1982): 1005-1006.

[76] Mussman, H. G. "Predictive image coding." *Image Transmission Techniques, Advances in Electronics and Electron Physics,* 12, Academic Press, New York, New York, 1979.

[77] Nasrabadi, N. M. and King, R. A. "Image coding using vector quantization: A review." *IEEE Transactions on Communications,* COM-36, No. 8 (August 1988): 957-971.

[78] Netravali, A. N. and Saigal, R. "Optimum quantizer design using a fized-point algorithm." *The Bell System Technical Journal,* 55, No. 9 (November 1976): 1423-1435.

[79] Netravali, A. N. "On quantizers for DPCM coding of picture signals." *IEEE Transactions on Information Theory,* IT-23, No. 3 (May 1977): 360-370.

[80] Netravali, A. N. and Limb, J. O. "Picture coding: A review." *Proceedings of the IEEE,* 68, No. 3 (March 1980): 366-406.

[81] Ngan, K. N. "Image display technique using the cosine transform." *IEEE Transactions on Acoustics, Speech, and Signal Processing,* ASSP-32, No. 1 (January 1984): 173-177.

[82] Oliver, B. N. "Efficient coding." *The Bell System Technical Journal,* 31, No. 7 (July 1952): 724-750.

[83] Pasco, R. C. "Source coding algorithms for fast data compression." *Ph.D. dissertation,* Stanford University, Stanford, California, 1976.

[84] Popat, A. C. "Scalar quantization with arithmetic coding." *M.S. thesis,* Massachusetts Institute of Technology, Cambridge, Massachusetts, 1990.

[85] Pursley, M. B. and Davisson, L. D. "Variable rate coding for nonergodic sources and classes of ergodic sources subject to a fidelity constraint." *IEEE Transactions on Information Theory,* IT-22, No. 3 (May 1976): 324-337.

[86] Rabbani, M. and Jones, P. W. "Digital image compression techniques." *SPIE Technical Publication,* TT07, 1991.

[87] Ramabadran, T. V. and Cohn, D. L. "An adaptive algorithm for the compression of computer data." *IEEE Transactions on Communications,* COM-37, No. 4 (April 1989): 317-324.

[88] Ramabadran, T. V. and Chen, K. "The use of contextual information in the reversible compressin of medical images." *IEEE Transactions on Medical Imaging,* MI-11, No. 2 (June 1992): 185-195.

[89] Ramabadran, T. V. and Hetherington, M. D. "A hybrid approach to high-fidelity compression of medical images." *IEEE Transactions on Medical Imaging,* under review.

[90] Ramamurthi, B. and Gersho, A. "Classified vector quantization of images." *IEEE Transactions on Communications,* COM-34, No. 11 (November 1986): 1105-1115.

[91] Riskin, E. A., Lookabaugh, T., Chou, P. A., and Gray, R. M. "Variable rate vector quantization for medical image compression with applications to progressive transmission." *Medical Imaging III: Image Capture and Display,* Roger H. Schneider, Samuel J. Dwyer III, R. Gilbert Jost, Editors, Proc. SPIE 1091 (1989): 110-117.

[92] Riskin, E. A. "Variable rate vector quantization of images." *Ph.D. dissertation,* Stanford University, Stanford, California, 1990.

[93] Riskin, E. A., Lookabaugh, T., Chou, P. A., and Gray, R. M. "Variable rate vector quantization for medical image compression." *IEEE Transactions on Medical Imaging,* MI-9, No. 3 (September 1990): 290-298.

[94] Riskin, E. A. and Gray, R. M. "A greedy tree growing algorithm for the design of variable rate vector quantizers." *IEEE Transactions on Signal Processing,* SP-39, No. 11 (November 1991): 2500-2507.

[95] Rissanen, J. J. "Generalized Kraft inequality and arithmetic coding." *IBM Journal of Research and Development,* 20, No. 3 (May 1976): 198-203.

[96] Rissanen, J. J. and Langdon, G. G., Jr. "Arithmetic coding." *IBM Journal of Research and Development,* 23, No. 2 (March 1979): 149-162.

[97] Rissanen, J. J. and Langdon, G. G., Jr. "Universal modeling and coding." *IEEE Transactions on Information Theory,* IT-27, No. 1 (January 1981): 12-23.

[98] Roos, P., Viergever, M. A., Van Dijke, M. C. A. and Peters, J. H. "Reversible intraframe compression of medical images." *IEEE Transactions on Medical Imaging,* MI-7, No. 4 (December 1988): 328-336.

[99] Rosenfeld, A. and Kak, A. C. *Digital Image Processing.* Academic Press, New York, 1982.

[100] Roucos, S., Schwartz, R., and Makhoul, J. "Segment quantization for very-low-rate speech coding." *Proceedings of IEEE Global Telecommunications Conference* (November 1982): 1074-1078.

[101] Rubin, F. "Arithmetic stream coding using fixed precision registers." *IEEE Transactions on Information Theory,* IT-25, No. 6 (November 1979): 672-675.

[102] Sabin, M. J. and Gray, R. M. "Product code vector quantizers for waveform and voice coding." *IEEE Transactions on Acoustics, Speech, and Signal Processing,* ASSP-32, No. 6 (June 1984): 474-488.

[103] Sabin, M. J. and Gray, R. M. "Global convergence and empirical consistency of the generalized Lloyd algorithm." *IEEE Transactions on Information Theory,* IT-32, No. 2 (March 1986): 148-155.

[104] Sanz, A., Munoz, C., and Garcia, N. "Approximation quality improvement techniques in progressive image transmission." *IEEE Journal on Selected Areas in Communications,* SAC-2, No. 2 (March 1984); 359-373.

[105] Shannon, C. E. "A mathematical theory of mommunication." *The Bell System Technical Journal,* 27, No. 3 (July 1948): 379-423 (Part I).

[106] Shannon, C. E. "A mathematical theory of communication." *The Bell System Technical Journal,* 27, No. 4 (October 1948): 623-656 (Part II).

[107] Shannon, C. E. "Coding theorems for a discrete source with a fidelity criterion." *IRE National Convention Record,* 4 (1959): 142-163.

[108] Sharma, D. K. and Netravali, A. N. "Design of quantizers for DPCM coding of picture signals." *IEEE Transactions on Communications,* COM-25, No. 11 (November 1977): 1267-1274.

[109] Sloan, K. R., Jr., and Tanimoto, S. L. "Progressive refinement of raster images." *IEEE Transactions on Computers,* C-28, No. 11 (November 1979): 871-874.

[110] Stewart, L. C. Stewart, Gray, R. M., and Linde, Y. "The design of trellis waveform coders." *IEEE Transactions on Communications,* COM-30, No. 4 (April 1982): 702-710.

[111] Thoma, W. "Optimizing the DPCM for video signals using a model of the human visual system." *Preceedings of the International Zurich Seminar on Digital Communications* (March 1974): C3.1-C3.7.

[112] Tou, J. T. and Gonzales, R. C. *Pattern Recognition Principles.* Addison-Wesley, Reading, Massachusetts, 1974.

[113] Tzou, K. H. "Progressive image transmission: A review and comparison of techniques." *Optical Engineering,* 26, No. 7 (July 1987): 581-589.

[114] Wallace, G. K. "The JPEG still picture compression standard." *Communications of the ACM,* 34, No. 4 (April 1991): 30-44.

[115] Witten, I. H., Neal, R. M. and Cleary, J. G. "Arithmetic coding for data compression." *Communication of the ACM,* 30, No. 6 (June 1987): 520-540.

[116] Wong, D., Juang, B.-H., and Gray, A. H., Jr. "An 800 bit/s vector quantization LPC vocoder." *IEEE Transactions on Acoustics, Speech, and Signal Processing,* ASSP-30, No. 10 (October 1982): 770-779.

[117] Williams, R. N. *Adaptive Data Compression.* Kluwer Academic Publishers, Boston, Massachusetts, 1991.

[118] Wilson, R. "Quad-tree predictive coding: A new class of image data compression." *Proceedings of the 1984 IEEE International Conference on Acoustics, Speech , and Signal Processing* (March 1984): 29.3.1-29.3.4.

[119] Wood, R. C. "On optimum quantization." *IEEE Transactions on Information Theory,* IT-15, No. 2 (March 1969): 248-252.

[120] Ziv, J. and Lempel, A. "A universal algorithm for sequential data compression." *IEEE Transactions on Information Theory,* IT-23, No. 3 (May 1977): 337-343.

[121] Ziv, J. "Coding theorems for individual sequences." *IEEE Transactions on Information Theory,* IT-24, No. 4 (July 1978): 405-412.

[122] Ziv, J. and Lempel, A. "Compression of individual sequences via variable-rate coding." *IEEE Transactions on Information Theory,* IT-24, No. 5 (September 1978): 530-536.

[123] Ziv, J. "On universal quantization." *IEEE Transactions on Information Theory,* IT-31, No. 3 (May 1985): 344-347.