Speech Compression using Wavelet Packet Transform and Vector Quantization

A. Averbuch[†] B. Gutman[‡]
[†] School of Mathematical Sciences
Tel Aviv University, Tel Aviv 69978, Israel
e-mail: amir@math.tau.ac.il

[‡]Department of Computer Science
The Hebrew University of Jerusalem
Jerusalem, Israel

Abstract

We present an algorithm for speech compression which uses wavelet packet transform, vector quantization, entropy coding and postfiltering of the decoded speech.

The experimental results indicate that the proposed coder achieves signal-to-noise ratio which is, in average, higher by 2.5 dB than achieved with GSM 06.10 standard coder, at 10% lower bit rate (11.5 kbps).The coder has algorithmic delay of 8 msec.

The vector quantizer used in the coder is a modified version of shape-gain VQ. Several original modifications to the well-known LGB codebook design algorithm are also presented.

1 Introduction

The area of digitized speech compression is recently going through rapid transition. In the past few years, we have witnessed the appearance of real-time speech communication over the Internet and the wide spread of cellular telephony.

In this paper we present a speech coder based on wavelet packet transform (WPT), vector quantization (VQ) of the transform coefficients and entropy coding of the VQ indices. The research was mainly focused on the following issues:

- Improving the LGB algorithm for VQ codebook design;
- Finding the WP statistical best basis which is best suited the speech compression.

We address the following issue: obtaining the best speech quality for a given bit rate with minimal algorithmic delay (applying it on the shortest segment).

The proposed coder outperforms the industry standard GSM 06.10 coder, both in terms of compression rate and signal-to-noise ratio. In addition, the proposed coder has a very low algorithmic delay of 8 msec (for comparison, the GSM 06.10 coder has a delay of 16 msec).

The algorithmic delay plays an important role in communication network. There is a standard which defines the maximal network delay to be about 15 msec. If the network delay is more than that, it is neccesary to use echo cancellation which is expansive and introduces extra complexity. Since the communication network consists of many types of equipments and each one has its own delay, every additional msec of delay is undesirable.

Digital speech communication is generally bandlimited to 4 kHz and sampled at 8 kHz. The simplest non-parametric coding technique is the Pulse-Code Modulation (PCM) which is simply a quantizer of sampled amplitudes. Speech coding at 64 kbit/sec using logarithmic PCM is considered as "non compressed" and is often used as a reference for comparisons for the quality of the compressed speech.

There are many types of speech compression algorithms. The description of the basic schemes is given in the next section.

The paper has the following structure: section 2 gives an overview of speech compression algorithms. Section 3 is dedicated to vector quantization and its new features; the formal definitions are given in 3.1 and the codebook design algorithm is described in 3.3. In subsection 3.3.3, *Cell Splitting*, we describe a simple modification to the LGB algorithm which improves the quality of resulting codebook as well as the efficiency of the design algorithm itself. An original variant of shape-gain VQ, which is used in our coder, is presented in 3.4.3. Section 4 describes the scheme and the individual blocks of the proposed algorithm. Finally, in Section 5 we compare the performance of the proposed coder to that of the GSM 06.10 coder. The appendix gives a short description of the wavelet packet transform.

2 Overview of Speech Compression Algorithms

There are more than five types of speech compression algorithms, where each one is a family of diverse schemes and forms. We describe here the typical basic configurations[18]: Waveform coders, vocoders and analysis-by-synthesis linear predictive coders.

2.1 Waveform Coders

Waveform coders focus upon representing the speech waveform as such without necessarily exploiting the underlying speech model. Waveform coders are generally more robust than vocoders, in the sense that they work well with a wider class of signals. However, they also operate at higher data rates than vocoders. In this description of waveform coders we include a short description of some of the fundamental scalar and vector quantization schemes which were also used in this paper.

2.1.1 Scalar Quantization

Scalar quantization methods include Pulse-Code Modulation (PCM), Differential PCM (DPCM), and Delta-Modulation (DM). Uniform PCM is a memoryless process that quantizes amplitudes by rounding off each sample to one of a set of discrete values. The difference between adjacent quantization levels, i.e. the step size, is constant in non-adaptive uniform PCM. Although uniform PCM is the simplest method for digital encoding, it is also the most expensive in terms of data rates since it has no mechanism for exploiting signal redundancy (correlation). Its performance in terms of SNR can be predicted by the expression $\mathrm{SNR} = 6B + K$ dB, where B is the number of bits per sample and K is a step-size dependent parameter. Non-uniform PCM quantizers generally employ a fine quantization step size for frequently occurring amplitudes and a coarse step size for rarely occurring amplitudes.

Another class of non-uniform PCM relies on logarithmic quantization. The operation of a logarithmic compressor can be described as smoothing the input pdf, obtaining the output pdf which resembles the uniform pdf $\mathbf{U}$. As a result, the log quantizers have a wider dynamic range than the uniform PCM; a 7-bit log quantizer for speech achieves the performance of a 12-bit uniform quantizer. The so called μ-law and A-law log compressors are, correspondingly, the North American and European PCM standards [18].

The more efficient scalar quantizer is the Differential PCM (DPCM) which utilizes the redundancy in the speech waveform by exploiting the correlation between adjacent samples. In its simplest form a DPCM transmitter encodes only the difference between successive samples and the receiver recovers the signal by integration. Practical DPCM schemes incorporate a time-invariant short-term prediction process, $A(z) = \sum a_i z^{-i}$ where z is a complex variable of the z-transform. This DPCM scheme is also called predictive differential coding and reduces the quantization error variance by reducing the variance of the quantizer input. Two other types of scalar coders are the Delta Modulation (DM) and Adaptive DPCM (ADPCM) coders. DM is a sub-class of DPCM where the difference (prediction error) is encoded only with 1 bit. DM typically operates at sampling rates much higher than the rates commonly used with DPCM. The step size in DM may also be adaptive (ADM). The DM and DPCM are low- to medium-complexity coders and perform better than ordinary PCM for rates at and below 32 kbit/s.

In backward adaptation, the parameters are estimated from past speech data, which is also available at the receiver. Therefore, the prediction parameters can be estimated locally at the receiver. Backward prediction adaptation is amenable to low-delay coding. ADPCM encoders with pole-zero decoder filters have proved to be particularly versatile in speech applications. In fact, the ADPCM 32 kbit/s algorithm adopted for the G.721 CCITT standard (1988) uses a pole-zero adaptive predictor.

2.1.2 Transform Coding

In transform coding (TC), the components of a unitary transform are quantized at the transmitter and decoded and inverse-transformed at the receiver. The potential for bit-rate reduction in TC lies in the fact that unitary transforms tend to generate near-uncorrelated transform components which can be coded independently.

Furthermore, the variances of these components often exhibit consistent or slowly time-varying patterns which can be exploited for redundancy removal using fixed or adaptive bit-allocation rules.

In TC, N successive signal samples $\{x(n)\}, n = 0, 1, \ldots, N-1$ are arranged into a block or vector $\mathbf{x}$. The vector $\mathbf{x}$ is transformed using a unitary matrix $\mathbf{T}$. The elements of $\mathbf{S} = \mathbf{T}\mathbf{s}$, i.e. the transform coefficients, are independently quantized using a bank of N quantizers $Q_i, i = 0, 1, \ldots, N-1$. The vector $\hat{\mathbf{S}}$ of quantized coefficients is then sent to the receiver. The receiver applies the inverse transform: $\hat{\mathbf{s}} = \mathbf{T}^{-1}\hat{\mathbf{S}}$. The elements of $\hat{\mathbf{s}}$ are the reconstructed output samples. In the absence of quantization and channel errors the inverse transform yields exact signal reconstruction. When the transform components are quantized, however, the reconstruction is no longer exact and depends on the quantization noise and the type of the transform.

The column vectors of $\mathbf{T}^{-1}$ are known as the basis vectors. For unitary transform $\mathbf{T}^{-1} = \mathbf{T}^H$, where ^{H} in the superscript denotes the Hermitian operator. For orthogonal transform, $\mathbf{T}^{-1} = \mathbf{T}^T$.

There are several discrete transforms that can be used for TC such as the *Discrete Cosine Transform* (DCT), the *Discrete Fourier Transform* (DFT), the *Discrete Wavelet Transform* (DWT). The basis vectors in DFT are simply sines and cosines (fundamental frequency $1/N$ and multiples thereof), while the DCT basis vectors are cosine functions with special phase and amplitude patterns. Very efficient algorithms exist for DFT and DCT computation. In fact, the JPEG image compression standard is based on DCT.

2.2 Vocoders

Vocoders rely on speech-specific analysis-synthesis which is mostly based on the source-system model. The encoder estimates the model parameters from incoming speech (***analysis***); these parameters are quantized and sent to the decoder. At the decoder, the speech is reconstructed by applying the quantized parameters to the model (***synthesis***). Due to parametric representation of speech, vocoders can operate at extremely low bit rates. The source-system representation has been utilized in most of the well-known low-rate coders. Many of the concepts associated with modern source-system vocoders have been introduced by Dudley [13, 14] In contrast to waveform coders presented in the previous section, the performance of vocoders generally degrades for non-speech signals.

2.2.1 Linear Predictive Vocoders

Linear Predictive (LP) vocoders have been the most widely researched technique for speech coding in the last twenty years. A linear speech production system has been developed and analyzed in [15]. This system assumes two-state excitation (pulse train for voiced and white noise for unvoiced speech). The vocal tract is modeled as an all-pole transfer function which is formed by a cascade of a small number of two-pole resonators representing the formants. The glottal model is represented as a two-pole lowpass filter and the lip-radiation model is represented by $L(z) = 1 - z^{-1}$. Finally, a spectral correction factor is included to compensate for the low-frequency effects of the higher poles. In digital speech representation, the spectral correction is omitted and the zero of the lip-radiation function is essentially canceled by one of the glottal poles. Hence the system can be reduced to an all-pole model.

The source-system synthesis model can be represented using z-domain functions, i.e.,

$$S(z) = g\frac{X(z)}{1 - A(z)} \tag{2.1}$$

where g represents the gain and $A(z)$ is given as $S = Ts$. The system and excitation parameters in (2.1) are unknown and must be determined from a finite set of speech samples. The coefficients of $A(z)$ are obtained using Linear Prediction (LP) [21, 22]. In a p-th order forward linear predictor the present sample of the speech sequence is predicted from a linear combination of p past samples, i.e., $\hat{s}^f(n) = \sum_{i=1}^{p} a_i^f s(n-i)$ where $\hat{s}^f(n)$ is the predicted sample.

2.2.2 Analysis-by-Synthesis Linear Predictive Coders

Analysis-by-synthesis Linear Predictive Coders is a class of source-system coders in which the system parameters are determined by linear prediction and the excitation sequence is determined by closed-loop or analysis-by-synthesis optimization. The optimization process determines an excitation sequence which minimizes a measure of the weighted difference between the input speech and the coded speech. The weighting or filtering function is chosen such that the coder is "optimized" for the human ear. The system consists of a short-term LP synthesis filter which represents the formant structure of speech, a long-term LP synthesis filter which represents the pitch (fine) structure of speech, a perceptual weighting filter $W(z)$ which shapes the error such that the quantization noise is masked by the high-energy formants, and the excitation generator which forms or selects and excitation sequence such that the weighted Mean Square Error (MSE) is minimized. The short-term predictor is typically updated at rates between 30 to 100 times per second while the long-term predictor is updated at higher rates, typically between 60 to 200 times per second. Although the analysis-by-synthesis scheme is fairly common, some of the proposed coders employ closed-loop configurations without the long-term predictor, or with the short-term predictor preceding the long-term predictor [4].

One of the problems in speech compression methods which utilize delayed-decision coders is that coding gain is achieved at the expense of coding delay. The one-way delay is basically the time elapsed from the instant a speech sample arrived at the encoder to the instant that this sample appears at the output of the decoder. This definition of one-way delay does not include channel- or modem-related delays. The delay is basically attributed to data buffering, processing, and generation of coding bits (channel symbols). The one-way delay is generally between two and four frames. For example, a typical code-excited linear predictive (CELP) algorithm with 20-ms frames is associated with delay of about 60 ms. Fast processing and encoders that transmit coding bits as they become available (on the fly) can reduce this delay.

In 1988, CCITT launched an investigation on low-delay speech coding algorithms operating at 16 kbit/s for possible standardization as a G. series recommendation for universal applications. The standardization effort motivated a great deal of research on low-delay coding. Of all the available coders the low-delay CELP (LDCELP) coder, proposed by Chen *et al.* [5] was selected. The LDCELP coder achieves low one-way delay by using a backward-adaptive predictor, and short excitation vectors (5 samples). In backward-adaptive prediction, the LP parameters are determined by operating on previously quantized speech samples which are also available at the decoder.

3 Vector Quantization

3.1 Definitions

A *vector quantizer* Q of dimension k and size N is a mapping from a vector in k-dimensional Euclidean space, $\mathcal{R}^k$, into a finite set $\mathcal{C}$ containing N output or reproduction points, called *code vectors* or *codewords*. Thus,

$$Q : \mathcal{R}^k \rightarrow \mathcal{C},$$

where $\mathcal{C} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N)$ and $\mathbf{y}_i \in \mathcal{R}^k$ for each i in $\mathcal{J} = \{1, 2, \ldots, N\}$. The set $\mathcal{C}$ is called the *codebook* and has N distinct elements. The *resolution* of a vector quantizer is $r = \log_2 N$, which measures the number of bits used to represent the input vector and gives an indication of the accuracy or precision that is achievable with a vector quantizer if the codebook is well-designed. Some authors use the term *resolution* to refer to the quantity $(log_2 N)/k$, which measures the number of bits per vector component.

Associated with every N point vector quantizer is a *partition* of $\mathcal{R}^k$ into N regions or *cells*, R_i for $i \in \mathcal{J}$. The ith cell is defined by

$$R_i = \left\{ \mathbf{x} \in \mathcal{R}^k \;\middle|\; Q(\mathbf{x}) = \mathbf{y}_i \right\},$$

sometimes it is called the *inverse image* of $\mathbf{y}_i$ under the mapping Q and denoted more concisely by $R_i = Q^{-1}(\mathbf{y}_i)$.

3.2 Nearest Neighbor Quantizers

An important special class of vector quantizers that is of particular interest, called *Voronoi* or *nearest neighbor* quantizers, has the feature that the partition is completely determined by the codebook and a distortion measure. For a given codebook such an encoder is optimal in the sense of minimizing average distortion. In fact, the term "vector quantizer" is commonly assumed to be synonymous with "nearest neighbor vector quantizer".

Suppose that $d(\mathbf{x}, \mathbf{y})$ is a distortion measure of the input/output vector space. For example, the squared error distortion measure defined by the squared Euclidean distance between two vectors:

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2 = \sum_{i=1}^{k} (\mathbf{x}_i - \mathbf{y}_i)^2, \tag{3.1}$$

This is the simplest measure and the most common for waveform coding.

A *Voronoi* or *nearest neighbor* (NN) vector quantizer is one whose partition cells are given by

$$R_i = \{ \mathbf{x} \mid d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j) \text{ for all } j \in \mathcal{J} \} .$$

In other words, with a nearest neighbor encoder, each cell R_i consists of all points $\mathbf{x}$ which have less distortion when reproduced with code vector $\mathbf{y}_i$ than with any other code vector.

3.3 Vector Quantizer Design

3.3.1 Initial Codebook Design

We begin with the problem of obtaining the initial codebook. There are a variety of techniques which generate a codebook that have been developed in cluster analysis (for pattern recognition) and in vector quantization (for signal compression). We describe here the splitting algorithm, which was used in this work.

The *splitting algorithm*, that was introduced by Linde et. al[20], produces increasingly large codebooks of a fixed dimension. The globally optimal resolution 0 codebook of a training sequence is the centroid of the entire sequence. The one code word, say $\mathbf{y}_0$, in this codebook that can be "splitted" into two code words, $\mathbf{y}_0$ and $\mathbf{y}_0 + \epsilon$, where ϵ is the vector with small Euclidean norm. This new codebook has two words and can be no worse than the previous codebook since it contains the previous codebook. The iterative improvement algorithm uses this codebook to produce a good resolution 1 code. When completed, all of the codewords in the new codebook can be splitted, forming an initial guess for a resolution 2 codebook. Once we continue in this manner we can utilize the good resolution r codebook to form by splitting the initial resolution for the $r+1$ codebook.

The choice of the new code word $\mathbf{y}' = \mathbf{y} + \epsilon$ will be described in section 3.3.3.

3.3.2 The LGB Algorithm

An iterative technique that has been successfully applied to codebook design is one that is based on a *clustering technique*. This algorithm is a generalization of the Lloyd-Max quantizer for scalar inputs, and it is called the Linde, Gray and Buzo (LGB) algorithm [20].

Suppose we have a set of observations of the signal to be quantized, called the *training set*,

$$\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$$

where M is the size of the training set. We define the *training ratio* as $\beta \equiv M/N$, where N is the codebook size. This parameter indicates how effectively the training set can describe the true pdf of the source and how well the LGB algorithm can yield a codebook that is nearly optimal for the true pdf. If we design an optimal N point quantizer for a finite training ratio, it is reasonable to expect that for M sufficiently large, it will be nearly optimal for the true distribution of the input random vector.

Lloyd Iteration

The Lloyd Iteration consists of two steps:

(a) Given a codebook $\mathcal{C}_m = \{ \mathbf{y}_i \mid i = 1, 2, \ldots, N \}$, partition the training set into cells R_i using the Nearest Neighbor Condition:

$$R_i = \{\mathbf{x} \in \mathcal{T} : d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j) \ \text{ for all } \ j \neq i)$$

where $d(\mathbf{x}, \mathbf{y}_i)$ is a distortion measure such as (3.1).

(b) Compute the centroids for the cells just found to obtain the new codebook $\mathcal{C}_{m+1}$, i.e.,

$$\mathbf{y}_{i,m+1} = cent(R_i) = \frac{1}{|R_i|} \sum_{\mathbf{x} \in R_i} \mathbf{x}.$$

The average distortion from quantizing the set $\mathcal{T}$ with the codebook $\mathcal{C}$ is defined as

$$D = D(\mathcal{T}, \mathcal{C}) = \frac{1}{M} \sum_{i=1}^{M} d(\mathbf{x}_i, \hat{\mathbf{x}}_i),$$

where $\hat{\mathbf{x}}_i$ is the nearest to $\mathbf{x}_i$ reproduction vector in $\mathcal{C}$. Alternatively, it can be expressed as

$$D = \frac{1}{M} \sum_{i=1}^{N} \sum_{\mathbf{x} \in R_i} d(\mathbf{x}, \mathbf{y}_i),$$

where R_i is the nearest neighbor cell associated with code vector $\mathbf{y}_i$.

The LGB Algorithm

The LGB algorithm for codebook design is simply a repetitive application of the Lloyd Iteration:

Step 1. Begin with initial codebook $\mathcal{C}_1$ and $m = 1$.

Step 2. Given the codebook $\mathcal{C}_m$, perform the Lloyd Iteration to generate the improved codebook $\mathcal{C}_{m+1}$.

Step 3. Compute the average distortion for $\mathcal{C}_{m+1}$. If it has changed by small enough amount since the last iteration, stop. Otherwise set $m \leftarrow m + 1$ and go to Step 2. Although various stopping criteria can be used, it is common and effective to test the relative decrease in the distortion

$$1 - \frac{D(\mathcal{C}_{m+1})}{D(\mathcal{C}_m)}$$

is below a suitable threshold.

Since each application of the Lloyd iteration reduces or leaves unchanged the average distortion, the LGB algorithm produces a sequence of codebooks with monotone non-increasing values with average distortion. Although this does not imply that the sequence of codebooks actually converges to a local optimum, nevertheless practical implementations of the algorithm have been found to be very effective.

3.3.3 Cell Splitting

If the cell R_i is empty at some iteration m, its centroid is undefined. In this case, we pick a non-empty cell R_j and split it, as was described in section 3.3.1. The new code word produced by the splitting operation becomes $\mathbf{y}_{i,m+1}$.

Since the splitting algorithm is employed both for the construction of the initial codebook and for fixing the empty cells in the LGB iteration, it deserves more attention. The most important issue in the splitting algorithm is the choice of a new codeword $\mathbf{y}'_i = \mathbf{y}_i + \epsilon$. Gersho & Gray [17] suggested to make ϵ proportional to the vector whose k-th component is the standard deviation of the k-th component of the set of training vectors, i.e. for a codebook cell R_i,

$$\epsilon_i = s \cdot \sigma(\mathbf{x} : \mathbf{x} \in R_i) = \frac{s}{|R_i|} \left[\sum_{\mathbf{x} \in R_i} (\mathbf{x} - \mathbf{y}_i)^2 \right]^{1/2},$$

where s is a scaling factor. Our experiments indicate that this method is not very effective, since many of the new cells $\mathbf{y}'_i$ are actually empty. We used the following modification to the splitting algorithm. In order to make sure that the new cell improves the codebook, we define its centroid $\mathbf{y}'_i$ as a point between the original centroid $\mathbf{y}_i$ and some other vector $\mathbf{x}_d \in R_i$ such that

$$\mathbf{y}'_i = \delta \mathbf{y}_i + (1 - \delta)\mathbf{x}_d, \ 0 < \delta < 1. \tag{3.2}$$

From this definition, it follows that $d(\mathbf{x}_d, \mathbf{y}'_i) < d(\mathbf{x}_d, \mathbf{y}_i)$, and hence, by the NN condition, the total codebook distortion will decrease. It is intuitively clear that the new codeword $\mathbf{y}'_i$ should not be too close to $\mathbf{y}_i$. Therefore, we define $\mathbf{x}_d$ as a vector in R_i whose distance from the centroid is the largest:

$$\mathbf{x}_d = \max_{\mathbf{x} \in R_i}{}^{-1} d(\mathbf{x}, \mathbf{y}_i)$$

We tested different values for δ, using the total codebook distortion $D_{total} = \sum_i D(R_i)$ as the measure for the quality of the resulting codebooks. We found that best codebooks (smallest distortion) are achieved with $\delta \approx 0.71$. In general, this method yields better codebooks than the variance-based one, and converges in fewer iterations, due to the more efficient treatment of empty cells.

We also discovered that it is better to fix empty cells not only in the LGB iteration, but also when constructing the initial codebook. Thus, some cells have to be splitted twice. For the second splitting, we use the same formula (3.2) with different value for δ, which was experimentally found to be $\delta_2 = -\delta = -0.71$.

Another issue, which rises in the context of the empty cell problem, is *which* cell should be splitted. The most natural choice is to split the cell R_i with the largest cell distortion $D(R_i)$, which is defined as

$$D(R_i) = \sum_{\mathbf{x} \in R_i} d(\mathbf{x}, \mathbf{y}_i). \tag{3.3}$$

We may argue that if many vectors fall to one cell, they deserve to be represented with greater precision, and hence the cell with the largest number of vectors should be splitted. On the other hand, if the cell distortion is very close to zero, there is no sense in splitting such a cell, no matter how many vectors it contains. To reflect this trade-off, we consider a weighted cell distortion

$$D_f(R_i) = \sum_{\mathbf{x} \in R_i} f(d(\mathbf{x}, \mathbf{y}_i)) \tag{3.4}$$

where $f(t)$ is a non-decreasing weighting function. The cell R_i, for which $D_f(R_i)$ is the largest, is chosen for splitting. Note that (3.3) is a special case of (3.4) for $f(t) = t$. In general, the steeper

$f(t)$, more "weight" $D_f(R_i)$ is assigned to these vectors in R_i which are far from the centroid (i.e., $d(\mathbf{x}, \mathbf{y}_i)$ is large). At the other extreme, for $f(t) \equiv 1$, $D_f(R_i)$ equals the number of vectors in R_i, and is not affected by distortion at all. We have compared the following weighting functions: $f(t) = t$; t^2; $t^{1/2}$; $t^{1/4}$; 1; $\log(t+1)$ and found that the best results are achieved with $f(t) = t$, which is also the natural choice.

3.4 Constrained Vector Quantization

There is no better way to quantize a single vector than to use VQ with a codebook that is optimal for the probability distribution that describes the random vector. However, direct use of VQ suffers from a serious complexity barrier that greatly limits its practical use as a complete and a self-contained technique.

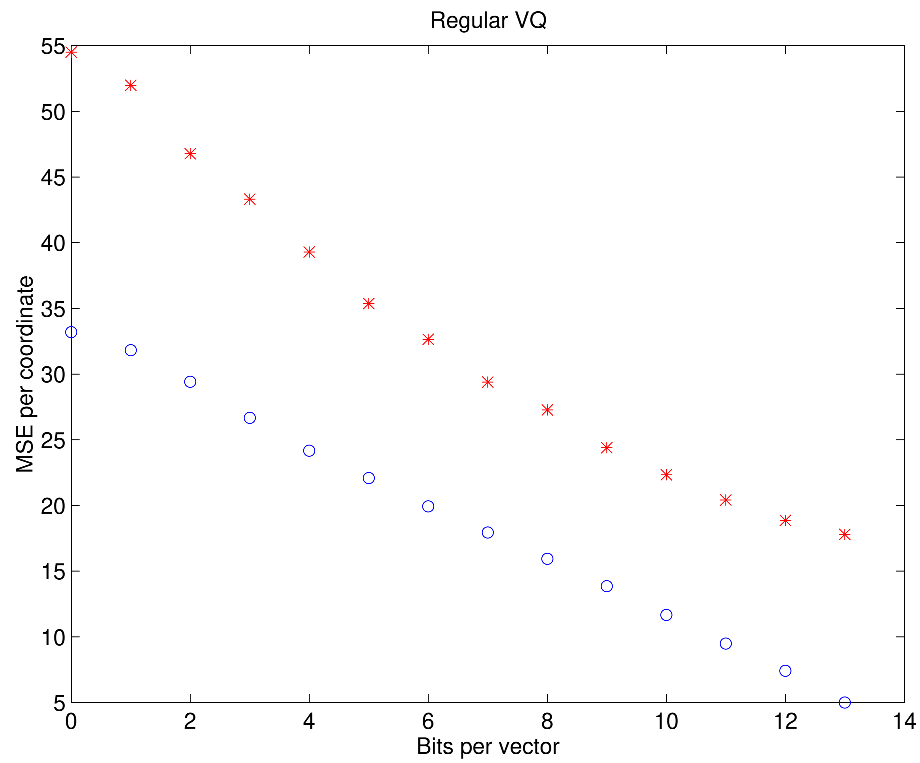
Unconstrained VQ is severely limited to rather modest vector dimensions and codebook sizes for practical problems. In speech coding, for example, the largest codebook sizes used typically lie in the range of 1024 to 4096 and the largest vector dimensions used are typically between 40 to 60. More commonly, codebook sizes range from 64 to 156 and vector dimensions from 4 to 10. Reducing the dimension of a vector often sacrifices the possibility of effectively exploiting the statistical dependency known to exist among the samples set or among the signal parameters. Thus, when designing a signal compression system, we tend to push toward the complexity limits of VQ to obtain the best coding gain.

Besides storage and complexity limitations, the size of the training set also limits the range of usable VQ resolutions. As the codebook grows, the training ratio decreases proportionally, and if the training ratio is not high enough, the resulting codebook departs even more from the optimal true pdf source. In other words, a larger codebook is better suited for the training set, but it does not necessarily guarantee better performance on other source samples.

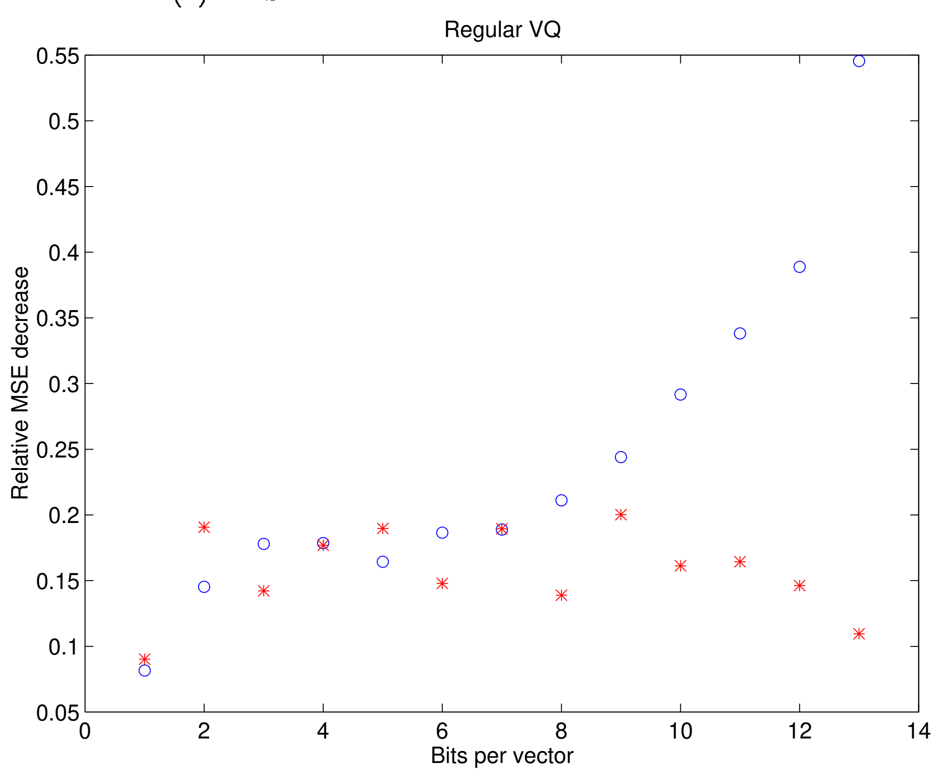
As a test, we used the LGB algorithm to build codebooks of resolutions 0,1,...,13 bits per vector from the training set $\mathcal{T}$ that contains 34541 vectors of dimension 8. The square root of the average distortion per vector coordinate vs. the codebook resolution is shown in Figure 1(a). Note how the average distortion of the training set (plotted with o) decreases almost linearly with n, while the distortion of test data (plotted with $*$) flattens out. This is yet more prominent in Figure 1(b), which shows that the relative distortion drops, defined as

$$1 - \frac{D(\mathcal{C}_n)}{D(\mathcal{C}_{n-1})},$$

for $n = 1, 2, \ldots, 13$. In essence, this quantity indicates how efficient is the increase in resolution from $n-1$ to n. The distortion drop graph for the test data (plotted with $*$) indicates that for given training set size, the use of codebooks larger than $2^{12} = 4096$ is unjustified.



(a) - MSE as a function of codebook resolution



(b) - relative MSE drop

Figure 1: VQ performance at different resolutions: $\circ$ - training set, $*$ - test data (outside the training set)

Several techniques have been developed which apply various constraints to the structure of the VQ codebook and yield a correspondingly altered encoding algorithm and design technique. As a result, higher vector dimensions and larger codebook sizes become feasible without hitting a "brick wall" which is complexity barrier. This section focuses on some of the most important constrained VQ techniques: Partitioned VQ, Mean-Removed VQ, Shape-Gain VQ, Multistage VQ.

3.4.1 Partitioned VQ

The simplest and the most direct way to reduce the search and storage complexity in coding a high dimensional vector is simply to partition the vector into two or more subvectors. Thus, the input vector

$$\mathbf{X} = (X_1, X_2, \ldots, X_k)$$

can be partitioned into

$$\mathbf{X}_a = (X_1, X_2, \ldots, X_m)$$

which is an m-dimensional vector with $m < k$ and

$$\mathbf{X}_b = (X_{m+1}, X_{m+2}, \ldots, X_k)$$

is a vector of dimension $k - m$.

Using the squared-error measure, we find

$$||\mathbf{X} - \hat{\mathbf{X}}||^2 = ||\mathbf{X}_a - \hat{\mathbf{X}}_a||^2 + ||\mathbf{X}_b - \hat{\mathbf{X}}_b||^2$$

from which it is obvious that the independent quantizer encoder structure is optimal. The encoder must find the nearest code vector to $\mathbf{X}_a$ in the codebook $\mathcal{C}_a$ and as a separate and independent task, the nearest code vector to $\mathbf{X}_b$ in the codebook $\mathcal{C}_b$. Furthermore, the optimal codebook design procedure is to derive two separate training sets, $\mathcal{T}_a$ and $\mathcal{T}_b$ of m- and $(k-m)$-dimensional vectors, respectively, from the original training set $\mathcal{T}$ of k-dimensional vectors and separately find optimal codebooks of specified sizes for each training set.

The partitioned VQ technique is directly generalized to a larger number than the two partitions considered above.

3.4.2 Mean-Removed VQ

Usually we deal with vectors that have statistical zero mean in the sense that the expected value of each component is zero. Nevertheless, the average amplitude of the components of the vector can differ widely form one vector to another and often this average can be regarded as statistically independent of the variation of the vector, that is, of the way the components vary about this average. The term *mean* of a vector will be used in this section specifically to refer to the *sample mean*, i.e., the average of the components of the vector. Thus, the mean, m, of $\mathbf{X}$ is itself a scalar random variable given by

$$m = \frac{1}{k}\sum_{i=1}^{k} \mathbf{X}_i = \frac{1}{k}\mathbf{1}^t\mathbf{X}$$

where $\mathbf{1} = (1, 1, \ldots, 1)$, the k-dimensional vector with all the components equal to unity.

The mean-removed *residual*, $\mathbf{R}$, of the random variable $\mathbf{X}$, is defined as

$$\mathbf{R} = \mathbf{X} - \frac{1}{k}(\mathbf{1}^t\mathbf{X})\mathbf{1} = \mathbf{X} - m\mathbf{1}$$

Hence, $\mathbf{X}$ can be represented as the sum of the mean vector $m\mathbf{1}$ and the residual vector $\mathbf{R}$ according to:

$$\mathbf{X} = m\mathbf{1} + \mathbf{R}. \tag{3.5}$$

The residual is the "mean-removed" version of the vector $\mathbf{X}$ and has zero mean. Thus, we have a natural decomposition of the original vector into separate features, a mean (representing a general background level) and a residual (representing the shape of the vector about its mean). Quantizing these features using separate codebooks is referred to as MRVQ for "mean-removed VQ" or "mean-residual VQ".

The codebook design procedure for the mean-removed VQ using the independent quantizer structure and the squared error distortion is straightforward. A training set of means $\mathcal{T}_m$ is formed and used to design the mean codebook. Similarly, a training set of residuals $\mathcal{T}_r$ is formed and used to design the residual codebook. There is no interaction between the two designs. Note also that the centroid of a set of zero mean vectors will also have zero mean. Hence, the designed residual codebook using the LGB algorithm will result in code vectors that have zero mean.

We have found that mean-removed VQ is less efficient than shape-gain VQ using the same number of bits. Shape-gain VQ is described in detail in the next section.

3.4.3 Shape-Gain VQ

The idea of shape-gain VQ is that the same pattern of variation in a vector may recur with a wide variety of gain values. If this is true, it suggests that the probability distribution of the shape is approximately independent of the gain. We would then expect very little compromise in optimality with a product code structure. From another perspective, we can say that the code is permitted to handle the dynamic range of the vector separately from the shape of the vector. Shape-gain VQ was introduced in [2] and optimized in [23]. It is particularly useful in speech and audio coding where the waveform has a wide dynamic range of short term power levels.

Shape-gain VQ is based on representation of the input vector $\mathbf{X}$ as

$$\mathbf{X} = g\mathbf{S}, \tag{3.6}$$

where the *gain* g, the Euclidean norm of $\mathbf{X}$, is given by $g = \|\mathbf{X}\| = (\sum_{i=1}^{k} X_i^2)^{1/2}$, and the *shape* $\mathbf{S}$ of a vector $\mathbf{X}$ with nonzero gain is given by $\mathbf{S} = \frac{\mathbf{X}}{g}$ so that $\|\mathbf{S}\| = 1$. The quantized vector is $\hat{\mathbf{X}} = \hat{g}\hat{\mathbf{S}}$ where $\hat{g}$ is the quantized version of g and $\hat{\mathbf{S}}$ is the quantized version of $\mathbf{S}$. The gain and shape codebooks are denoted by $\mathcal{C}_g$ and $\mathcal{C}_s$ will have sizes N_g and N_s, respectively. All shape vectors in the shape codebook $\mathcal{C}_s$ have unit gain.

In [17], a codebook design procedure for shape-gain VQ is described. We consider shape-gain VQ in a broader sense: it is still based on the product representation (3.6), but does not rely on shape vectors having unit gain. By examining the performance objective $d(\mathbf{X}, \hat{\mathbf{X}})$, we obtain:

$$d(\mathbf{X}, \hat{\mathbf{X}}) = \|\mathbf{X} - \hat{\mathbf{X}}\|^2 = \|\mathbf{X} - \hat{g}\hat{\mathbf{S}}\|^2 = \left(\hat{g}\|\hat{\mathbf{S}}\| - \frac{\mathbf{X}^t\hat{\mathbf{S}}}{\|\hat{\mathbf{S}}\|}\right)^2 + \mathbf{X}^2 - \frac{(\mathbf{X}^t\hat{\mathbf{S}})^2}{\|\hat{\mathbf{S}}\|^2}. \tag{3.7}$$

The third term is always non-positive and independent of $\hat{g}$. Hence, the *best shape* for the input vector $\mathbf{X}$ is

$$\hat{\mathbf{S}} = \max_{\mathbf{S}\in\mathcal{C}_s}{}^{-1} \frac{(\mathbf{X}^t\mathbf{S})^2}{\|\mathbf{S}\|^2}, \tag{3.8}$$

and the *best gain* for $\mathbf{X}$ and shape $\hat{\mathbf{S}}$ is given by $g = \mathbf{X}^t\hat{\mathbf{S}}/||\hat{\mathbf{S}}||^2$. The quantity

$$d(\mathbf{X}, g\hat{\mathbf{S}}) = \mathbf{X}^2 - \frac{(\mathbf{X}^t\hat{\mathbf{S}})^2}{||\hat{\mathbf{S}}||^2} \tag{3.9}$$

is the minimal distortion achievable with a given shape codebook. Although this lower bound can only be attained if the gain is quantized with unlimited resolution, i.e., $\hat{g} = g$, it is still a valuable indication of the effectiveness of gain-shape quantizer. Specifically, by comparing $d(\mathbf{X}, \hat{g}\hat{\mathbf{S}})$ to $d(\mathbf{X}, g\hat{\mathbf{S}})$ we can conclude how the gain-shape quantizer's performance will improve if we increase the resolution of the gain (scalar) quantizer.

In order to design a gain-shape quantizer, one could use a codebook produced by the regular LGB algorithm. While using this codebook as the shape codebook, we then generate the set of best gains $\{\mathbf{X}^t\hat{\mathbf{S}}/||\hat{\mathbf{S}}||^2\}$ and design a scalar quantizer for the gain from this set. The resulting quantizer, however, will be suboptimal, since the shape codebook does not take advantage of the gain-shape structure. Roughly speaking, some vectors in the shape codebook may be wasted on representing vectors which actually have the same shape. In order to incorporate gain into the codebook design, we replace the Nearest Neighbor partition in the Lloyd iteration with the *nearest shape* partition, i.e.

$$R_i = \{\mathbf{x} \in \mathcal{T} : \frac{\mathbf{x}^t\mathbf{y}_i}{||\mathbf{y}_i||^2} \geq \frac{\mathbf{x}^t\mathbf{y}_j}{||\mathbf{y}_j||^2} \text{ for all } \; j \neq i\}$$

It also reasonable to modify the centroid computation in the Lloyd iteration (which is essentially an average of all the vectors in a partition cell) so that each vector in the cell is divided by the appropriate best gain (we shall refer to this modification as *gain scaling*), i.e. instead of

$$\mathbf{y}_{i,m+1} = \frac{1}{|R_i|} \sum_{\mathbf{x} \in R_i} \mathbf{x}$$

we define

$$\mathbf{y}_{i,m+1} = \frac{1}{|R_i|} \sum_{\mathbf{x} \in R_i} \frac{\mathbf{x}}{g},$$

where $g = (\mathbf{x}^t\mathbf{y}_{i,m})/||\mathbf{y}_{i,m}||^2$.

The motivation for gain scaling is as follows. Consider a nearest shape cell which contains two vectors: $\mathbf{x}$ and $-\mathbf{x}$. Obviously, the best shape for representing these vectors is $k\mathbf{x}$ (where $k \neq 0$ is an arbitrary scalar). However, if we do not employ gain scaling, the reproduction vector will be set to the cell's centroid, which is zero. Also, vectors with larger norm tend to "pull" the centroid towards them; intuitively, gain scaling might offer an advantage by smoothing this effect.

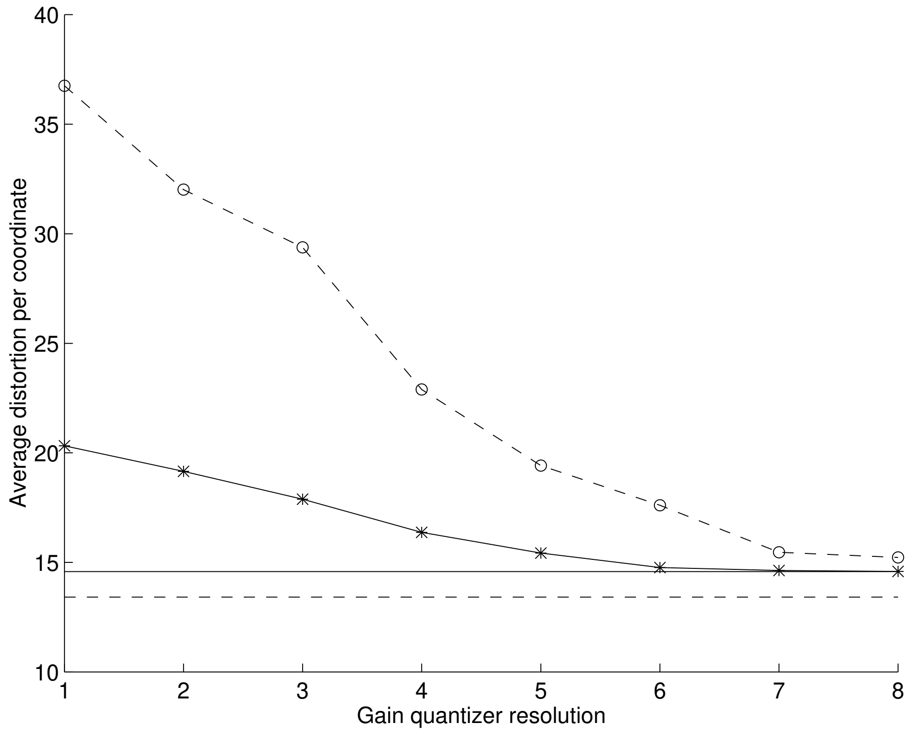


Figure 2: Performance of the gain-shape quantizer for a codebook that was built without gain scaling (solid line) and with gain scaling (dashed line).

It turns out that, although this modification indeed produces better codebooks in the sense of having smaller average *lower bound* distortion $d(\mathbf{X}, g\hat{\mathbf{S}})$, the actual average distortion of a gain-shape quantizer designed from such codebook is very far from this bound. As a result, shape codebooks built without gain scaling perform significantly better. Figure 2 depicts the performance of two 12-bit codebooks: 1. built without gain scaling (depicted with $*$, solid lines). 2. built with gain scaling (depicted with $\circ$, dashed lines). The square root of average distortion is shown for gain quantizer resolutions with 1,2,...,8 bits; the horizontal lines depict the square root of the distortion lower bound. Notice that codebook 1 performs fairly well for all gain resolutions. As the gain resolution increases, the performance rapidly approaches the optimum; at 6-bit resolution, the lower bound is practically achieved. In contrast, the performance of codebook 2 is very sub-optimal, even when as much as 8 bits are used for gain quantization.

A possible explanation for this phenomenon is that the above argument in favor of gain scaling implicitly assumes that gain level $1/k$ can be found in $\mathcal{C}_g$. Thus, even a modest distortion in gain quantization results in a poor gain-shape quantizer. Indeed, the situation is very different if we fix the set of possible gain levels (gain codebook $\mathcal{C}_g$) in advance, *before* designing the shape codebook. In this case, gain scaling is expressed as

$$\mathbf{y}_{i,m+1} = \frac{1}{|R_i|} \sum_{\mathbf{x} \in R_i} \frac{\mathbf{x}}{\hat{g}},$$

where $\hat{g} \in \mathcal{C}_g$ is the *quantized* best gain g. The set $\mathcal{C}_g$ can be obtained by quantizing the set of best gain levels for the training set and an initial codebook, resulting in a sort of closed-loop design algorithm. For pre-determined set of gain levels, we found the gain scaling technique to be a valuable modification, which significantly improves the quantizer performance (compared to the plain nearest-shape iteration).

Of particular interest is the case of 1-bit gain quantization. We define gain codebook which consists of the unity gain and one alternative gain: $\mathcal{C}_g = \{1, a\}$. Shape codebooks for $a = -4, -2, -1, -0.5, 0.5, 2, 4$ have been built; the results of quantizing 4 test files with these codebooks are shown in Figure 3.

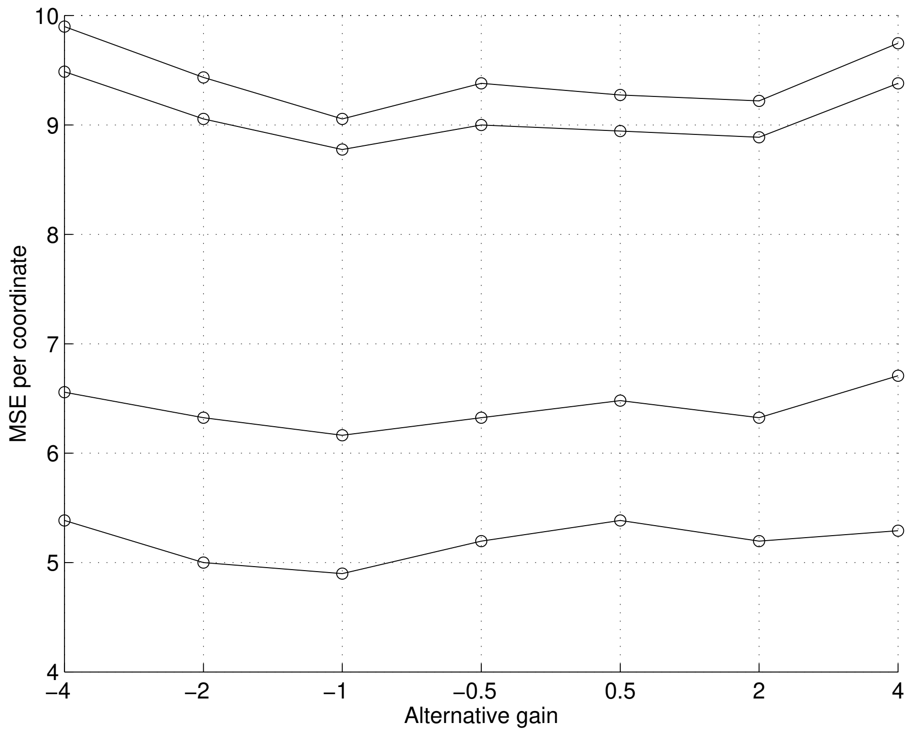


Figure 3: Performance of the gain-shape quantizer for different alternative gains (1 bit gain quantization).

The value of $a = -1$ produces the best results. Besides being optimal in terms of the distortion, this *shape-sign* representation of the vector offers an effective way for VQ encoding. Recall that the best shape computation, according to equation (3.8), requires division by $||\mathbf{S}||^2$. However, if $\hat{g} \in \mathcal{C}_g = \{+1, -1\}$, then $\hat{g}^2 = 1$ and expanding the quantization error (3.7) yields

$$d(\mathbf{X}, \hat{g}\hat{\mathbf{S}}) = ||\mathbf{X}||^2 - 2\hat{g}(\mathbf{X}^t\hat{\mathbf{S}}) + ||\hat{\mathbf{S}}||^2 = ||\mathbf{X}||^2 + 2\left(\frac{||\hat{\mathbf{S}}||^2}{2} - \hat{g}(\mathbf{X}^t\hat{\mathbf{S}})\right)$$

Thus, the best shape $\hat{\mathbf{S}}$ is the codevector which minimizes $||\hat{\mathbf{S}}||^2/2 - |\mathbf{X}^t\hat{\mathbf{S}}|$, and the gain $\hat{g}$ is simply

$$\hat{g} = \begin{cases} +1 & \text{if } \mathbf{X}^t\hat{\mathbf{S}} > 0 \\ -1 & \text{otherwise} \end{cases}$$

If the set of half-norms $\{||\mathbf{S}||^2/2 : \mathbf{S} \in \mathcal{C}_s\}$ is computed in advance, this encoding algorithm requires the same number of multiplications as the regular VQ. The overhead is one addition for each codevector.

3.4.4 Multistage VQ

If the random vector $\mathbf{X}$ is such that is does not have a wide variation of gain or mean values, then the shape-gain or mean-removed VQ methods are not likely to be very helpful. If the dimension is quite high, partitioned VQ would certainly handle the complexity problem but might severely degrade the performance when there is substantial statistical interdependence between different subvectors.

One alternative technique that has proved valuable in a number of speech and image coding applications is the *multistage* or *cascaded VQ* [19]. This technique is also sometimes referred to as *residual* [1, 16]. The basic idea of multistage VQ (MSVQ) is to divide the encoding task into successive stages, where the first stage performs a relatively crude quantization of the input vector using a small codebook. Then, the second stage quantizer operates on the error vector between the original and the quantized first stage output. The quantized error vector then provides a second approximation to the original input vector thereby leading to a refined or more accurate representation of the input. A third stage quantizer may then be used to quantize the second stage error vector to provide a further refinement, and so on.

Two-stage VQ is illustrated in Figure 4.

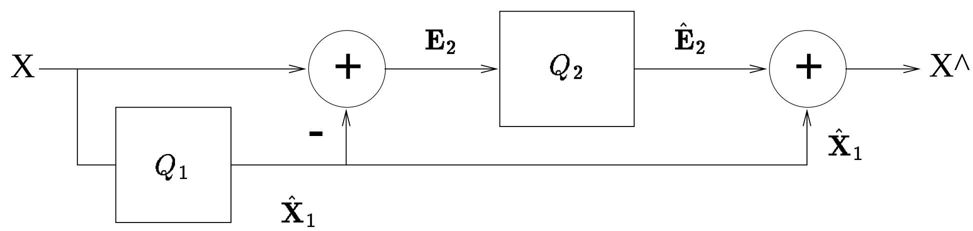


Figure 4: Two-Stage VQ

The input vector $\mathbf{X}$ is quantized by the initial or the first stage vector quantizer denoted by Q_1. The quantized approximation $\hat{\mathbf{X}}_1$ is then subtracted from $\mathbf{X}$ producing the error vector $\mathbf{E}_2$. This error vector is then applied to a second vector quantizer Q_2 yielding the quantized output $\hat{\mathbf{E}}_2$. The overall approximation $\hat{\mathbf{X}}$ to the input $\mathbf{X}$ is formed by summing first and second approximations, $\hat{\mathbf{X}}_1$ and $\hat{\mathbf{E}}_2$, respectively. The encoder for this VQ scheme generates a pair of indices that specify the selected code vectors for each stage and the task of the decoder is to perform two table lookups to generate and then sum two code vectors.

The general multistage VQ method can be generalized by induction from the two-stage scheme. The vector quantizer at each stage operates on the quantization error vector from the previous stage. Each stage generates an index that is sent to the decoder. Note that the complexity is reduced from $N = \prod_{i=1}^{m} N_i$ to $\sum_{i=1}^{m} N_i$ and the equivalent product codebook is generated from the Cartesian product $\mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_m$. Thus, both the complexity and storage requirements can be substantially reduced using multistage VQ. There is, however, a performance penalty with multistage technique. The coding gain of the quantizer depends on the statistics of the input signal and the successive quantizers tend to have rapidly diminishing coding gains (i.e., the SNR per bit decreases) due to the fact that vector components of successive stages tend to be less correlated. In general, the quantization error vector has more randomness than the input to the quantizer since its components tend to be less statistically dependent than those of the input vector. Thus, in practice, multistage coders often have only two and occasionally three stages.

Codebook design for multistage VQ is also performed in stages. First, the original training set $\mathcal{T}$ is used to generate the first stage codebook of the desired size in the conventional manner. Next, a new training set $\mathcal{T}_1$ is generated by applying the training set to the first stage VQ and generating the set of error vectors. This training set has the same size as the original and the vectors have the same dimension. The process is repeated for successive stages. Note that the codebook design complexity is reduced here in comparison with the design of a single stage VQ of dimension N.

This codebook design procedure is not optimal in the sense that it does not find the best set of codebooks for sequentially encoding a vector stage-by-stage with the multistage structure. It is, however, *greedy* in the sense that it finds the codebook for the first stage that would be optimal if there were only one stage. Then, it finds the best codebook for the second stage given the first-stage codebook and assumes that there are only two stages. Similarly, each successive stage is optimal given all previous codebooks while assuming that this is the last stage.

As we pointed out earlier, a vector quantizer performs very differently on vectors outside the training set than it does on the training set itself. In particular, the average Euclidean norm of the error vectors is much smaller in the latter case. Therefore, the error vector codebook $\mathcal{T}_1$ does not represent well the true pdf of the error vectors. This observation leads to a simple but valuable improvement to the above design technique: we apply a *different* training set $\mathcal{T}'$ ($\mathcal{T}' \cap \mathcal{T} = \emptyset$) to the first-stage codebook, using the generated error vectors as the training set for the second-stage codebook. For the general case of m-stage VQ, m distinct training sets are required, say $\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \ldots, \mathcal{T}^{(m)}$. For $k = 2, \ldots, m$, the training set of error vectors for the k-th stage codebook is generated by applying $\mathcal{T}^{(k)}$ to the first $k-1$ stages.

4 Description of the Compression Algorithm

The proposed compression scheme is based on the following phases (Fig. 5):

- Application of the wavelet packet transform (WPT) [7] on a fixed size small segment (frame) of input speech. The determination of the best filter and the statistical best basis/level[8] is done once.

- Vector quantization (VQ) of the wavelet packet coefficients

- Entropy coding

- Postfiltering (PF) of the reconstructed speech at the decoder side

The block diagram of the proposed algorithm is shown in Figure 5. The individual blocks are described in the following sections.

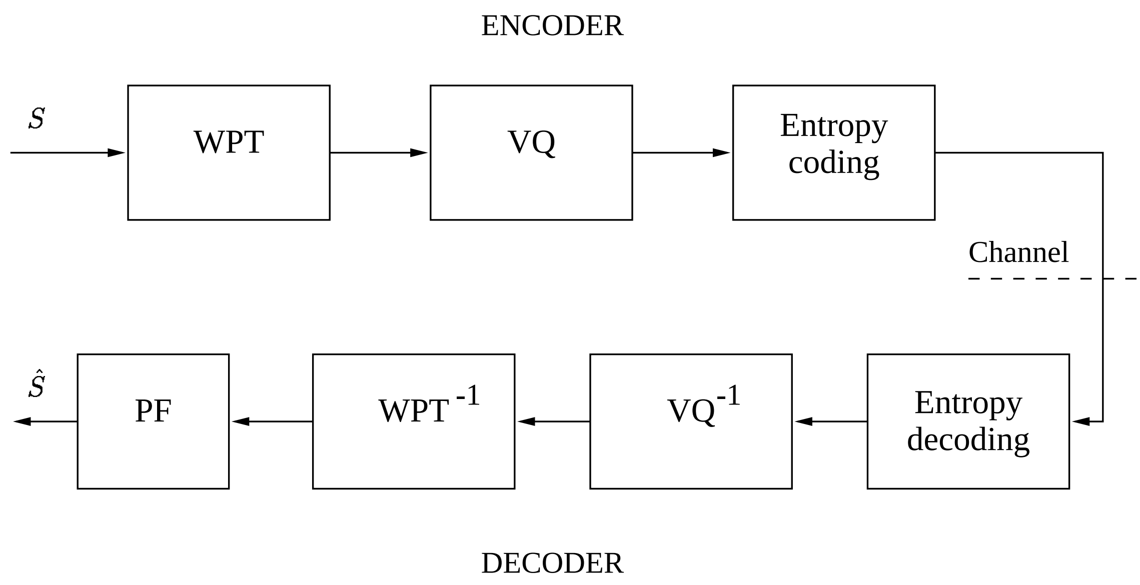


Figure 5: The block diagram of the proposed speech compression algorithm

4.1 Wavelet Packet Transform and Vector Quantization

Theoretical background on wavelet packet is given in the Appendix. For convenience, we index the nodes of the wavelet packet transform (WPT) binary tree as follows:

- The tree root (original signal) is node 0 at level 0. The k-th node at level n is denoted $k{:}n$.
- If $k{:}n$ is not a leaf, its left (lowpass component) and right (highpass component) child nodes are $2k{:}(n+1)$ and $(2k+1){:}(n+1)$, respectively. Thus, the sons of 0:0 are 0:1 and 1:1; the sons of 1:1 are 2:2 and 3:2, etc.

All nodes at n-th tree level are vectors of length $N/2^n$.

Applying VQ to longer vectors allows for more efficient exploitation of the dependency among the vector components. Therefore, we seek that the WP best basis will have the longest possible leaf nodes, within the limitations of feasible codebook size. Each processing segment has 64 samples. We found that nodes 1:1 (length 32) and 1:2 (length 16) can be quantized with acceptable distortion using 12-bit shape-sign quantizers (described at the end of section 3.4.3). This resolution, however, is not sufficient for adequate representation of nodes 0:3 and 1:3. This is undertstandable since they correspond to the lowest frequency band, which contains most of the speech energy. There are several possible ways to improve the quantization of these nodes:

- Increase codebook size; but, as we realized, it is not efficient to use VQ with resolution greater than 12 bits.
- Use more gain levels. However, the relative distortion drop decreases as the gain resolution increases; in other words, each next bit which is added to gain quantization is less effective than the previous. Besides, shape-sign VQ (1-bit gain quantization) is computationally attractive.
- Employ multistage VQ.
- Employ partitioned VQ, quantizing the first and the last 4 coordinates of each node as separate vectors.
- Perform additional WP decomposition, yielding 4 vectors of length 4.

The last three options are compared in table 1. All methods use regular VQ with 12-bit codebooks, total of 48 bits (3 bits per vector component). As expected, two-stage VQ performs significantly better than the other methods, which operate with smaller vector dimensions. We also compared different bit allocations for 2- and 3-stage VQ, and found the lowest distortion to be achieved with two-stage VQ.

Scheme	SNR, dB
Two-stage VQ	15.47
Partitioned VQ (two groups of 4 components)	14.41
Regular VQ of child nodes 0:4, 1:4, 2:4 and 3:4	13.79

Table 1: Comparison among different quantization schemes for nodes 0:3 and 1:3

Thus, we arrived at WP best basis which is shown in Figure 6. This is done with Beylkin's filter that has 18 taps.

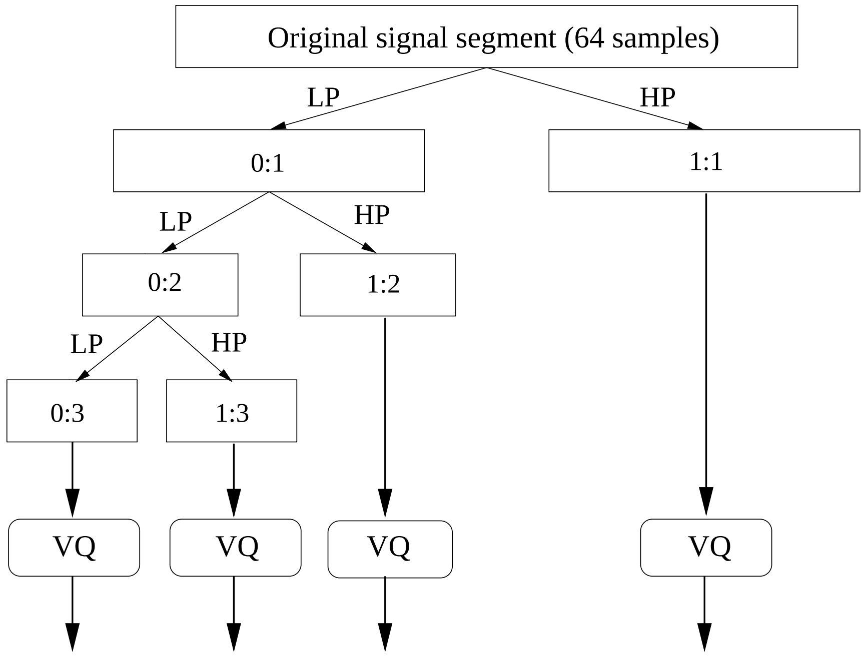


Figure 6: The statistical best wavelet packet basis

In order to improve the speech quality, we added second-stage quantizers for nodes 1:2 and 1:1; these quantizers can be independently turned off if higher compression is desired. Each box which is labeled "VQ" in Figure 6 is a two-stage quantizer, with 12-bit shape-sign quantizer in each stage. In the absence of entropy coding, the encoder outputs $(12 + 1) \cdot 8 = 104$ bits for every processed signal segment. For the sampling rate of 8 kHz, there are 125 segments per second, so the encoder can operate at the fixed rate of 13000 bps (or $13 \cdot 6 \cdot 125 = 9750$ bps if the optional quantizers are off). The algorithmic delay is 64 samples (segment size), which for 8kHz the sample rate it is 8 msec.

4.2 Entropy Coding

The use of entropy coding inevitably leads to a variable-rate coder. Entropy coding can significantly improve the compression ratio losslessly, although in some applications a fixed-mode operation can do better. We proposed two opions: one with Huffman coding and one with arithmetic coding.

4.2.1 Huffman Coding

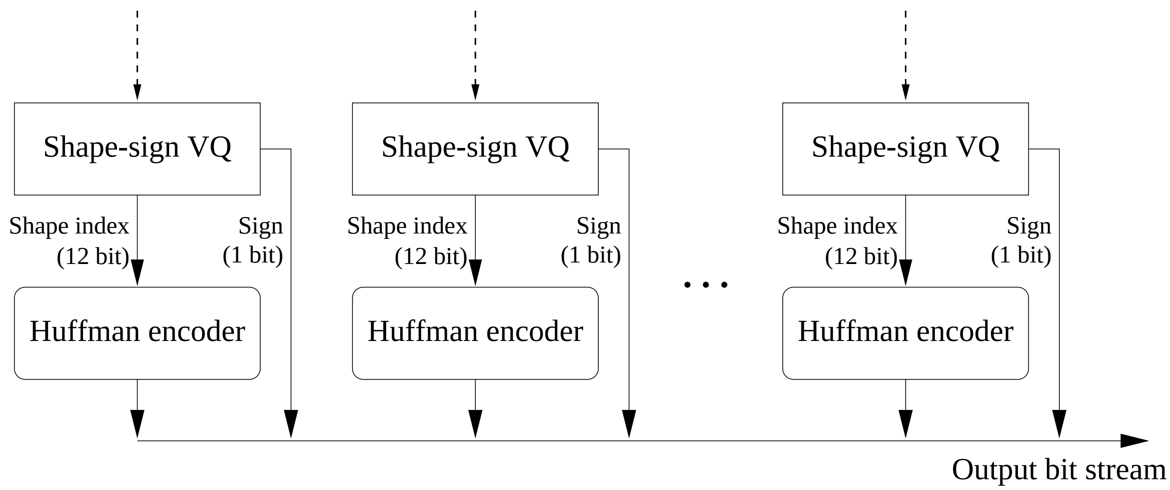


Figure 7: Configuration with independent Huffman coders

The first approach is to assign an independent entropy coder to each VQ, as shown in Figure 7. Arithmetic coding cannot be used in such scheme, since the output of several arithmetic coders should not be merged into one bit stream. Therefore, this scheme uses 8 independent Huffman coders. Each coder accepts the 12-bit shape index output by its VQ, and outputs the corresponding variable-length codeword.

In order to exploit a possible correlation between the gain bits produced by the VQs, we formed from them a 8-bit index. Unfortunately, this index has entropy 7.9, which means that there is no potential for bit-rate reduction if we apply entropy coding to this variable. Therefore, each gain bit is appended to the output stream as is.

The average Huffman code lengths are shown in table 2. The expected average bit-rate of the coder is $(74.1 + 1) \cdot 8 = 10263$ bps, or $(56.1 + 1) \cdot 6 = 7763$ bps without the optional VQs.

Coder	Entropy	Average Huffman code length
1:1	8.3	8.3
1:2	8.8	8.9
0:3	9.7	9.7
0:3 (2nd stage)	10.5	10.6
1:3	8.5	8.6
1:3 (2nd stage)	10.0	10.0
Total:	55.8	56.1
1:1 (2nd stage)	8.7	8.7
1:2 (2nd stage)	9.2	9.3
Total:	73.7	74.1

Table 2: Independent Huffman coders: entropies and code lengths

4.2.2 Arithmetic Coding

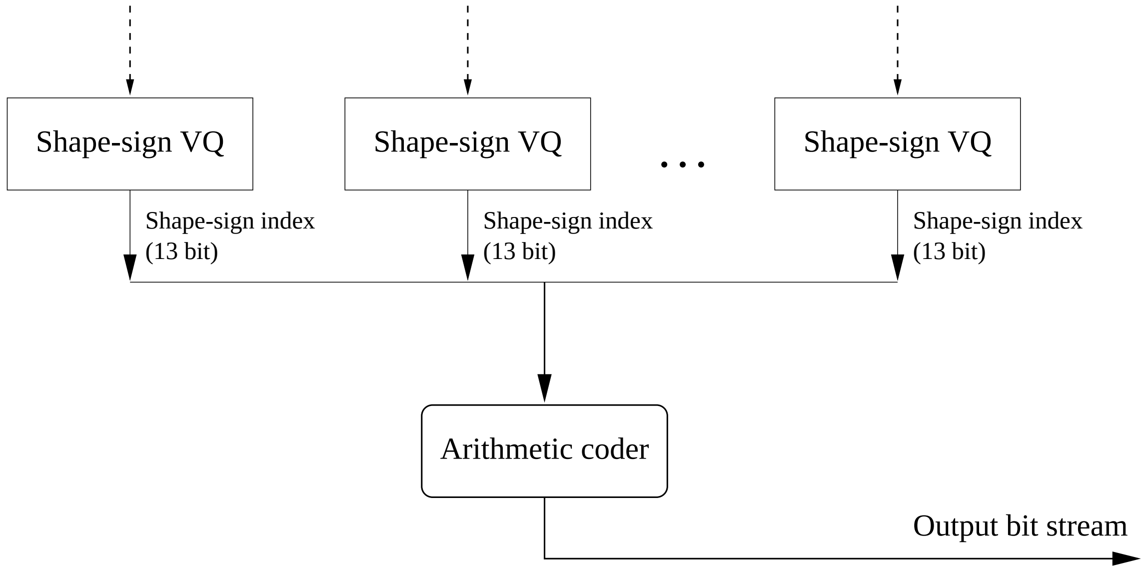


Figure 8: Configuration with global arithmetic coder

In order to employ the arithmetic coding, the output indices from all the VQs must be merged, as shown in Figure 8; our decision to use VQs of identical resolution makes this merging straightforward. Note that shape and sign are combined to form a 13-bit shape-sign index, since the sign bits can no longer be inserted into the output stream. The global arithmetic coder receives these 13-bit indices and outputs a sequence of bits.

We want that the merged output of all the VQs will have the lowest entropy. Therefore, we arrange the codewords of each VQ in a "probability" appearance order; i.e., in each codebook, codeword 0 is the most frequent etc. The resulting merged index has entropy 11.5, which leads to the expected average bit rate of $11.5 \cdot 8 \cdot 125 = 11500$ bps (or $11.5 \cdot 6 \cdot 125 = 8625$ without the two extra quantizers). Although the average bit rate is higher than the Huffman coding, we found that the arithmetic coder achieves better compression in all cases. This can be explained by two reasons. First, Huffman coding does not have the ability to approach the entropy lower bound as closely as the arithmetic coding. Second, the combination of shape and sign into one index turns out to be an advantage, since it allows the coder to exploit the asymmetry in distribution of sign bits.

Adaptivity and *compact model representation* is an additional advantage of arithmetic coding over Huffman coding. While a Huffman coder, in theory, can be made to perform adaptively, rebuilding the code tree is a computationally heavy task, which therefore should not be performed too frequently; arithmetic coder, on the other hand, adapts to each incoming symbol, with very small overhead in computations. Also, the data structures required for building Huffman code tree are fairly large; in fact, replacing 8 Huffman coders with one arithmetic coder in our implementation reduced the memory requirements by approximately 1 Mb. Thus, arithmetic coding is clearly the preferred choice.

4.3 Postfilter

The postfilter block filters the decoded speech to enhance the perceptual quality. It can be turned off, for instance, for non-speech signals.

The postfilter block scheme [3] is shown in Figure 9.

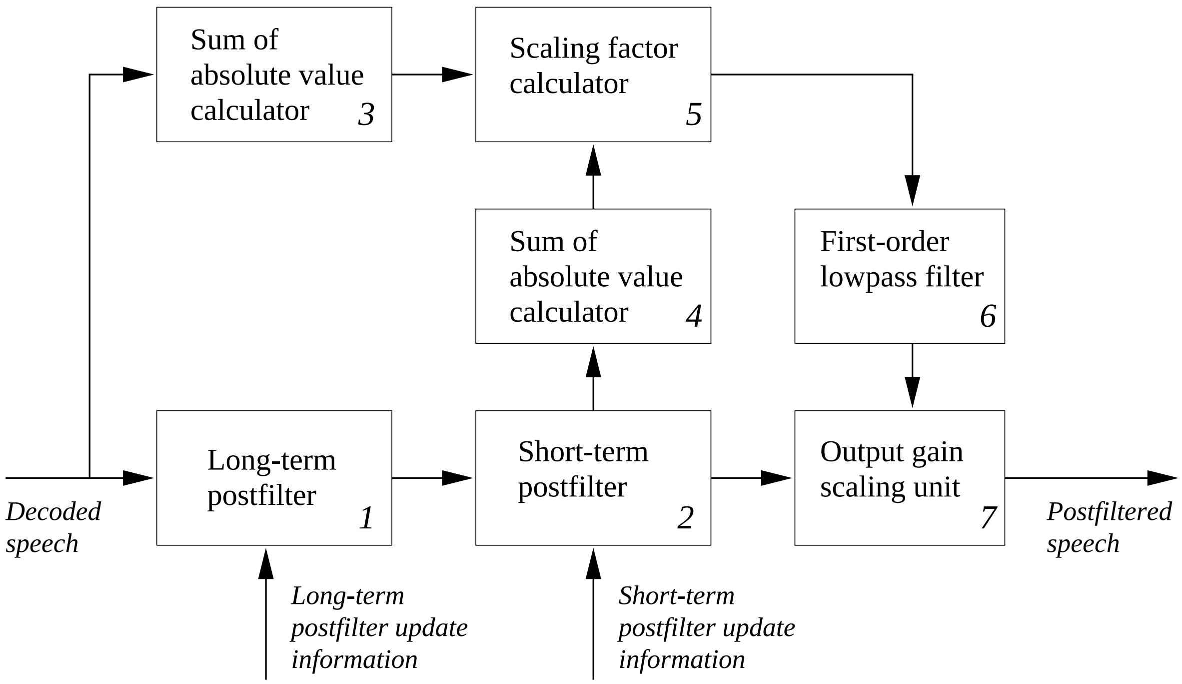


Figure 9: The postfilter block

The *long-term postfilter* "1", sometimes called the *pitch postfilter*, is a comb filter with its spectral peaks located at multiplies of the fundamental frequency (or *pitch frequency*) of the speech to be postfiltered. The reciprocal of the fundamental frequency is called the *pitch period*. The pitch period can be extracted from the decoded speech using a pitch detector (or pitch extractor). Let p be the fundamental pitch period (in samples) obtained by a pitch detector, then the transfer function of the long-term postfilter can be expressed as

$$H_l(z) = g_l(1 + bz^{-p}),$$

where the coefficients g_l, b and the pitch period p are updated once every 4 vectors (each vector consists of 5 consecutive speech samples).

The *short-term postfilter* "2" consists of a 10th-order pole-zero filter in cascade with a first-order all-zero filter. The 10th-order pole-zero filter attenuates the frequency components between formant peaks, while the first-order all-zero filter attempts to compensate for the spectral tilt in the frequency response of the 10th-order pole-zero filter.

Let $\tilde{a}_i, i = 1, 2, \ldots, 10$ be the coefficients of the 10th-order linear predictor obtained by backward LPC analysis of the decoded speech, and let k_1 be the first reflection coefficient obtained by the same analysis. Then, the transfer function of the short-term postfilter is

$$H_s(z) = \frac{1 - \sum_{i=1}^{10} \bar{b}_i z^{-i}}{1 - \sum_{i=1}^{10} \bar{a}_i z^{-i}}[1 + \mu z^{-1}],$$

where $\bar{b}_i = \tilde{a}_i(0.65)^i$, $\bar{a}_i = \tilde{a}_i(0.75)^i$ for $i = 1, 2, \ldots, 10$, and $\mu = (0.15)k_1$. The coefficients $\bar{a}_i$, $\bar{b}_i$ and μ are also updated once every 4 vectors.

In general, after the decoded speech is passed through the long-term and the short-term postfilter, the filtered speech will not have the same power level as the decoded unfiltered speech. To avoid occasional large gain excursions, it is necessary to use automatic gain control to force the postfiltered speech to have roughly the same power as the unfiltered speech. This is done by blocks "3" through "7".

The sum of absolute value calculator "3" operates vector-by-vector. It takes the current decoded speech vector $s_d(n)$ and calculates the sum of the absolute values of its five vector components. Similarly, the sum of absolute value calculator "4" performs the same type of calculation, but on the current output vector $s_f(n)$ of the short-term postfilter. The scaling factor calculator "5" then divides the output value of block "3" by the output value of block "4" to obtain a scaling factor for the current vector $s_f(n)$. Ths scaling factor is then filtered by a first-order lowpass filter "6" to get a separate scaling factor for each of the five components of $s_f(n)$. The first-order lowpass filter "6" has a transfer function of $0.01/(1 - 0.99z^{-1})$. The lowpass filtered scaling factor if used by the output gain scaling unit "7" to perform sample-by-sample scaling of the short-term postfilter output. Note that since the scaling factor calculator "5" only generates one scaling factor per vector, it would have a staircase effect on the sample-by-sample scaling operation of block "7" if the lowpass filter "6" were not present. The lowpass filter "6" effectively smooths out such a staircase effect.

5 Experimental Results

The signal-to-noise ratio (SNR) is one of the most common objective measures for evaluating the performance of a compression algorithm. The SNR is defined as

$$10 \log_{10} \frac{\sum s^2(n)}{\sum [s(n) - \hat{s}(n)]^2},$$

where $s(n)$ is the original signal and $\hat{s}(n)$ is the coded signal.

File name	SNR, *dB*			Bit rate, *bps*		
	`WPACK`	`GSM`	gain	`WPACK`	`WPACK-H`	`GSM`
male48	11.62	8.91	2.71	12452	12964	13200
male49	13.47	10.86	2.61	11578	11580	13200
male50	10.99	7.22	3.77	11597	11617	13200
male51	11.16	7.29	3.87	11810	11865	13200
female35	14.93	12.77	2.16	10991	10772	13200
female36	10.37	7.30	3.07	11201	11334	13200
female37	13.70	10.61	3.09	12119	12360	13200
female38	15.13	13.63	1.50	11176	11091	13200
music1	8.00	5.62	2.38	11086	11174	13200
music2	12.84	9.88	2.96	13277	14277	13200
music3	9.85	5.93	3.92	13156	14041	13200

Table 3: Comparison between the GSM 06.10 coder and the proposed compression algorithm (WPACK). The `WPACK` uses arithmetic coding, `WPACK-H` uses Huffman coding.

In Table 3, the performance of the proposed algorithm is compared to that of GSM 06.10 standard coder, which uses RPE/LTP (residual pulse excitation/long term prediction). The GSM coder operates at the fixed rate of 13200 bps.

As is evident from the table, the proposed coder (WPACK) achieves higher SNR at lower average bit rate. The SNR gain of WPACK over the GSM coder, shown in a separate column, ranges between 1.5 to 3.9 dB, being over 2.9 dB in average. Although the proposed coder uses arithmetic coding, the bit rate achieved with Huffman coding (WPACK-H) is shown for comparison. For tests shown in Table 3, the average bit rate of WPACK-A is 11856 bps, which is approximately 10% less than the fixed rate of GSM coder. Note that the highest bit rate corresponds to non-speech samples *music2, music3*; in fact, *music2* is the only test for which the bit rate exceeds 13200 bps, and the excess is only 77 bps.

For the highest possible compression ratio, the optional second-stage quantizers can be turned off. As can be seen from Table 4, even in this case the SNR is still comparable (and in many cases, higher) than what is achieved with the GSM coder, and the bit rate is, in average, 30% lower . In terms of subjective quality, the decoded speech contains audible distortions, but nevertheless is intelligible and recognizable.

File name	SNR, *dB*		Bit rate, *bps*	
	`WPACK`	`GSM`	`WPACK`	`GSM`
male48	9.56	8.91	9356	13200
male49	11.49	10.86	8743	13200
male50	8.45	7.22	8732	13200
male51	9.10	7.29	8950	13200
female35	12.82	12.77	8419	13200
female36	7.53	7.30	8331	13200
female37	11.71	10.61	9123	13200
female38	13.57	13.63	8537	13200
music1	5.13	5.62	8336	13200
music2	10.86	9.88	10079	13200
music3	7.28	5.93	9886	13200

Table 4: Comparison between the GSM 06.10 coder and the proposed compression algorithm (WPACK) in the highest compression mode (without the second-stage quantizers for nodes 1:1 and 1:2).

Another useful property of WPACK is that high speech quality is retained after several encoding/decoding stages. Figure 10 depicts the SNR after $n = 1, 2, 3, 4$ successive applications of the WPACK algorithm to a test file (graph plotted with $*$) . After four applications of the coder, the SNR decreases by less than 1.5 dB. In fact, this decrease is almost solely due to successive applications of the decoding postfilter. If the postfilter is turned off, the SNR decreases by just 0.56 dB (graph plotted with o). In contrast, four applications of the GSM coder to the same file cause SNR decrease of 3 dB (lower graph, plotted with x).

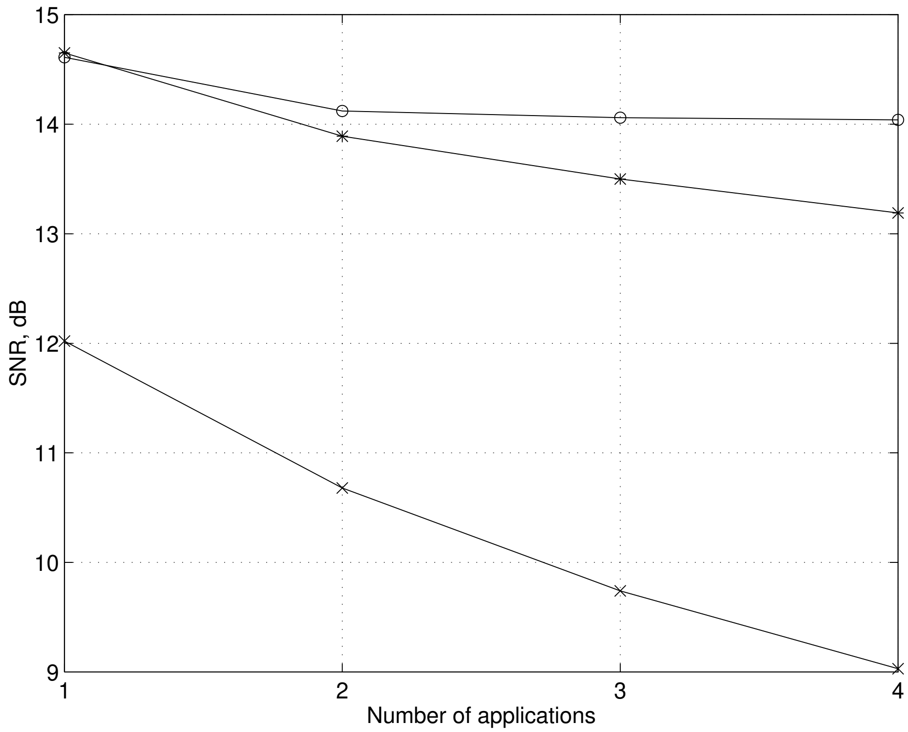


Figure 10: The SNR after $n = 1, 2, 3, 4$ encoding/decoding stages. $*$ - WPACK, $\circ$ - WPACK without postfiltering, x - GSM

The only serious drawback of the proposed algorithm is its rather high computational and storage requirements. The coder uses 8 codebooks of $2^{12} = 4096$ vectors each. Four of these codebook quantize vectors of length 8 (first and second-stage VQ for 0:3 and 1:3), two of length 16 (two-stage VQ for 1:2) and two of length 32 (two-stage VQ for 1:1). For each codevector, its Euclidean norm divided by 2 is precomputed and also stored. Hence, the total storage required for the codebooks is $4096 \cdot (4 \cdot (8+1) + 2 \cdot (16+1) + 2 \cdot (32+1)) = 557056$ values, which amounts to 2,176 Kbytes of data if 4-byte floats are used. Since quantization of vector of length k with codebook of size N requires kN multiplications, the VQ block performs $4096 \cdot (4 \cdot 8 + 2 \cdot 16 + 2 \cdot 32) = 524288$ multiplications for each encoded segment, or $\approx$65 million multiplications per second of encoded speech. In our implementation for Pentium 133 processor, the encoding takes approximately 6 seconds per second of encoded speech (4 seconds without the extra quantizers). The WPT and the arithmetic coding stages take a negligible fraction of the encoding time.

6 Conclusion

Despite its heavy computational and storage requirements, the proposed coder outperforms the standard LPC coder both in terms of SNR and compression ratio, and also has twice lower coding delay. As the experimental results demonstrate, the use of WPT for speech compression is a promising alternative to LPC for medium to high bit-rate applications. Besides, the possible applications of WPT are not limited to speech coding. Finally, any coder using VQ can benefit from the proposed modifications to the LGB algorithm.

Appendix

The wavelet packet bases is another large class of orthonormal bases. This class of bases comes from applying quadrature mirror filters (QMFs). See [12] for background on wavelets and QMF's. We first give a non-mathematical description. A pair of filters is associated with each mother wavelet function ψ having compact support (such as the Daubechies wavelets and Coiflets). This pair of high pass and low pass filter generates bases which are included in our library. For each such pair of QMFs, the high pass and low pass filters are applied successively to a finite sampled signal, to build a (finite) binary tree of coefficients. We can specify how many levels the tree will have, but it cannot have more than $\log_2$ of the signal length levels. At each filter applications We have to treat borders with special care. A more mathematical description of the above process is given in [12] and we will use the notation and terminology there.

We are given an exact quadrature mirror filter $h(n)$ satisfying the conditions of the Theorem 3.6 pp. 964 in [11]: i.e.

$$\sum h(n-2k)h(n-2l) = \delta_{k,l} \ , \ \sum h(n) = \sqrt{2}.$$

The easiest example, associated with the Haar basis, is given by $h_0 = \frac{1}{\sqrt{2}}$, $h_1 = -\frac{1}{\sqrt{2}}$. We let $g_k = h_{k+1}(-1)^k$ and define the operations F_i on $\ell^2(\mathbf{Z})$ into "$\ell^2(2\mathbf{Z})$"

$$\begin{aligned} F_0\{s_k\}(i) &= 2\sum s_k h_{k-2i} \\ F_1\{s_k\}(i) &= 2\sum s_k g_{k-2i}. \end{aligned}$$

The map $\mathbf{F(s_k) = F_0(s_k) \oplus F_1(s_k) \in l^2(2Z) \oplus l^2(2Z)}$ is orthogonal and

$$F_0^* F_0 + F_1^* F_1 = I$$

We now define the following sequence of functions.

$$\begin{aligned} W_{2n}(x) &= \sqrt{2}\sum h_k W_n(2x-k) \\ W_{2n+1}(x) &= \sqrt{2}\sum g_k W_n(2x-k). \end{aligned}$$

Clearly the function $W_0(x)$ can be identified with the function φ in [12] and W_1 with the function ψ.

All of the functions W_n have a fixed scale, but we observe that mixed-scale decompositions of L^2 are also possible. This allows us to refine the decomposition L^2 by scales as embodied in the following, see [10] Theorem 1.2:

Theorem: For every partition P of the nonnegative integers into sets of the form $I_{kn} = \{2^k n, \ldots, 2^k(n+1)-1\}$, the collection of functions $\{2^{k/2}W_n(2^k t - j) : I_{kn} \in P, j \in \mathbf{Z}\}$ is an orthonormal basis for $L^2(\mathbf{R})$.

Remark: We may also think of I_{nk} as the dyadic subinterval $[2^{-k}n, 2^{-k}(n+1)[$ of $[0,1[$. Such an indexing convention gives a faithful correspondence between disjoint dyadic decompositions of the frequency line and orthonormal wavelet packet subsets of L^2.

Definition: A wavelet packet basis of $L^2(\mathbf{R})$ is any orthonormal basis selected from among the functions $2^{k/2}W_n(2^k t - j)$.

A useful picture of the tree of wavelet packet coefficients is that of a rectangle of dyadic blocks. The row number within the rectangle indexes the scale of the wavelet packets listed therein. The column number indexes both the frequency and position parameters. In what follows, Ω_n represents the span of the translates of W_n and δ is normalized dilation by 2.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
s_1	s_2	s_3	s_4	d_1	d_2	d_3	d_4
ss_1	ss_2	ds_1	ds_2	sd_1	sd_2	dd_1	dd_2
sss_1	dss_1	sds_1	dds_1	ssd_1	dsd_1	sdd_1	ddd_1

Figure 11: A rectangle of wavelet packet coefficients

The boxes of coefficients in the rectangle correspond to the decomposition of $\delta^L\Omega_0$ into the subspaces $\delta^k\Omega_n$, for $0 \leq k \leq L$ and $0 \leq n < 2^{L-k}$. The top box corresponds to $\delta^L\Omega_0$, the bottom boxes correspond to Ω_n, for $0 \leq n < 2^L$, and box n on level k (counting the bottom as level 0) corresponds to subspace $\delta^k\Omega_n$.

For definiteness, consider a function defined at 8 points $\{x_1, \ldots, x_8\}$, i.e., a vector in $\mathbf{R}^8$. Then the (periodized) wavelet packet coefficients of this function look like the what is described in Fig. 11.

Each row is computed from the row above it by one application of either F_0 or F_1, which we think of as "summing" (s) or "differencing" (d) operations, respectively. Thus, for example, the subblock $\{ss_1, ss_2\}$ comes from the application of F_0 to $\{s_1, s_2, s_3, s_4\}$, while $\{ds_1, ds_2\}$ comes similarly from F_1. The two descendant s and d subblocks on row $n+1$ are determined by their mutual parent on row n, which conversely is determined by them through the adjoint anti-convolutions F_0^* and F_1^*.

Note that the tree corresponding to each quadrature mirror filter contains many bases, including the wavelet bases. The wavelet basis is obtained by choosing the second block in each row and the first and second blocks in the last row.

We now describe the use of a cost function to choose one particular basis, which has the lowest cost for the signal under consideration compared to all the possible bases than can be extracted from the trees.

Least cost is a somewhat arbitrary criterion. The coefficients corresponding to speech should have relatively large amplitudes, and be few in number. This requirement is precisely what is measured by a low Shannon entropy (the cost function we used, described below). Thus, we have decided that the mathematical criterion of least entropy cost should most likely approximate the subjective criteria of noiseless speech.

We associate a cost function to an expansion of a unit vector y in some particular orthonormal basis $\{w_i\}$, with coefficients y_i. We used the Shannon entropy which in the above notation is defined as $-\sum {y_i}^2 \log_2 {y_i}^2$. If y is not a unit vector, it needs to be normalized. The Shannon entropy measures how well y is represented by a particular basis. One extreme case is when all y_i's are 0 except one. Then the entropy is 0, the basis is the best possible representation for the signal. For the other extreme, when all the y_i's have equal magnitude, i.e. y is completely spread out relative to the basis, entropy is $\log_2 n$, where n is the number of entries in the vector. The closer the entropy is to 0, the better the basis represents the signal, i.e. only a few relatively large y_i's, the rest small. If the entropy is close to $\log_2 n$, all, or most, of the coefficients of y are of roughly equal size.

For the interested reader, we give a more precise description of the Shannon entropy, extracted from [8, 10]. Let H be a Hilbert space. Let $v \in H$, $\|v\| = 1$ be a unit vector and assume

$$H = \oplus \sum H_i \quad ,$$

an orthogonal direct sum. We define

$$\varepsilon^2(v, \{H_i\}) = -\sum \|v_i\|^2 \ell n \|v_i\|^2$$

as a measure of distance between v and the orthogonal decomposition. ε^2 is characterized by the Shannon equation which is a version of Pythagoras' theorem. Let

$$\begin{aligned} H &= \oplus(\sum H^i) \oplus (\sum H_j) \\ &= H_+ \oplus H_- \quad . \end{aligned}$$

H^i and H_j give orthogonal decompositions $H_+ = \sum H^i$, $H_- = \sum H_j$. Then

$$\begin{aligned} \varepsilon^2(v;\ \{H^i, H_j\}) &= \varepsilon^2(v, \{H+, H_-\} \\ &+ \|v_+\|^2 \varepsilon^2 \left(\frac{v_+}{\|v_+\|}, \{H^i\}\right) + \|v_-\|^2 \varepsilon^2 \left(\frac{v_-}{\|v_-\|}, \{H_j\}\right). \end{aligned}$$

This is Shannon's equation for entropy. We interpret $\|P_{H_+} v\|^2$ as the "probability" of v to be in the subspace H_+.

References

[1] C.F.Barnes and R.L.Frost. Necessary conditions for the optimality of residual vector quantizers. In *Abstracts of the 1990 IEEE International Symposium on Information Theory*, p.34, San Diego, Calif., January 1990.

[2] A.Buzo, A.H.Gray, Jr., R.M.Gray, and J.D.Markel. Speech coding based upon vector quantization. *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-28:562-574, October 1980.

[3] International Telecommunication Union, CCITT. *Coding of speech at 16 kbit/s using low-delay code excited linear prediction.* Recommendation G.728. Geneva, 1992.

[4] J.Chen and A.Gersho. Vector adaptive predictive coding of speech at 9.6 kbps. In *Proc.ICASP-86*, pp.1693-1696, 1986.

[5] J.Chen, R.Cox, Y.Lin, N.Jayant and M.Melchner. A low-delay CELP coder for the CCITT 16 kb/s speech coding standard. *IEEE Trans.Selected Areas Commun. (Special Issue on Speech and Image Coding)*, N.Hubing, Ed., vol.10, pp.830-849, June 1992.

[6] R.Coifman, Y.Meyer, S.Quake, and V.Wickerhauser. Signal processing and compression with wave packets. *Numerical Algorithms Research Group*, New Haven, CT: Yale University, 1990

[7] R.R.Coifman and Y.Meyer. *Orthonormal Wave Packet Bases*, preprint, Dept. of Mathematics, Yale University, New Haven, 1990.

[8] R.R.Coifman and M.V.Wickerhauser. Entropy-based Algorithms for Best Basis Selection, IEEE Trans. on Information Theory, vol. 38, No.2, pp.713-718, March 1992.

[9] R.R.Coifman and Y.Meyer. Size properties of wavelet packets, in *Wavelets and their Applications*, edited by Ruskai et.al., pp.125-150, Jones and Bartlett, 1992.

[10] R.R. Coifman, V. Wickerhauser. Wavelets and Adapted Waveform Analysis. A Toolkit for Signal Processing and Numerical Analysis, *Proc. Symp. in Applied Mathematics,* Vol. 47, pp. 119-153, 1993.

[11] I. Daubechies, Orthonormal Bases of Compactly Supported Wavelets, Comm. Pure Appl. Math., vol. 41, pp. 909-996, 1988.

[12] I.Daubechies. *Ten Lectures on Wavelets.* SIAM, 1992.

[13] H.Dudley. Remaking Speech. *J. Acoust. Soc. Amer.*, vol.11, p.169, 1939.

[14] H.Dudley. Phonetic Pattern Recognition Vocoder for Narrowband Speech Transmission. *J. Acoust. Soc. Amer.*, vol.30, pp.733-739, 1958.

[15] G.Fant. *Acoustic Theory of Speech Production.* Gravenhage, The Netherlands. Mounton and Co., 1960.

[16] R.L.Frost, C.F.Barnes, and F.Xu. Design and performance of residual quantizers. In *Proceedings of Data Compression Conference*, pp.129-138, Snowbird, Utah, April 1991. IEEE Computer Society Press.

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

[18] N.S.Jayant and Peter Noll. *Digital Coding of Waveforms.* Prentice-Hall, 1984.

[19] B.H.Juang and A.H.Gray,Jr. Multiple stage vector quantization for speech coding. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pp.597-600, Paris, April 1982.

[20] Y.Linde, A.Buzo and R.M.Gray. An Algorithm for Vector Quantizer Design, *IEEE Trans. on Communications*, pp.84-95, Jan.1980.

[21] J.Makhoul. Linear Prediction: A Tutorial Review. *IEEE*, vol.63, no.4, pp.561-580, Apr. 1975.

[22] J.Markel and A.Gray, Jr. *Linear Prediction of Speech.* Springer Verlag, New York, 1976.

[23] M.J.Sabin and R.M.Gray. Product code vector quantizers for waveform and voice coding. *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-32:474-488, June 1984.