A Study of Bit Allocation for Gaussian Mixture Model Quantizers and Image Coders

Denis Tran

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

September 2005

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Masters of Engineering.

Abstract

This thesis describes different bit allocation schemes and their performances when applied on coding line spectral frequencies (LSF) using the GMM-based coder designed by Subramaniam and a simple image transform coder. The new algorithms are compared to the original bit allocation formula, the *Pruning* algorithm used by Subramaniam, Segall's method and the *Greedy* bit allocation algorithm using the Log Spectral Distortion and the Mean-Square Error for the LSF quantizer and the Peak Signal-to-Noise Ratio for the image coder.

First, a Greedy level allocation algorithm is developed based on the philosophy of the Greedy algorithm but it does so level by level, considering the best benefit and bit cost yielded by an allocation. The Greedy level allocation algorithm is computationally intensive in general, thus we discuss combining it with other algorithms to obtain lower costs.

Second, another algorithm solving problems of negative bit allocations and integer level is proposed. The level allocations are to keep a certain ratio with respect to each other throughout the algorithm in order to remain closest to the condition for lowest distortion. Moreover, the original formula assumes a 6dB gain for each added bit, which is not generally true. The algorithm presents a new parameter k, which controls the benefit of adding one bit, usually set at 0.5 in the high-rate optimal bit allocation formula for MSE calling the new algorithm, the Two-Stage Iterative Bit Allocation (TSIBA) algorithm. Simulations show that modifying the bit allocation formula effectively brings about some gains over the previous methods.

The formula containing the new parameter is generalized into a formula introducing a new parameter which weights not only the variances but also the dimensions, training the new parameter on their distribution function. The TSIBA was an *a-posteriori* decision algorithm, where the decision on which value of k to select for lowest distortion was decided after computing all distortions. The Generalized TSIBA (GTSIBA), on the other hand, uses a training procedure to estimate which weighting factor to set for each dimension at a certain bit rate. Simulation results show yet another improvement when using the Generalized TSIBA over all previous methods.

Sommaire

Cette thèse décrit différentes méthodes d'attribution de bits et leurs performances quand elles sont appliquées au codage de fréquences spectrales (LSF) de lignes en utilisant le codeur de la parole conçu par Subramaniam et un codeur d'image à transformation. Ces méthodes sont comparées avec la formule d'attribution de bit original, la méthode de *Pruning* utilisée par Subramaniam, la méthode de Segall et l'algorithm *Avare* d'attribution de bits en utilisant la distortion spectrale logarithmique (LSD) et l'erreur carrée moyenne (MSE) pour le quantizeur de fréquences spectrales en ligne (LSF) et le rapport maximal de signal-bruit (PSNR) pour le codeur d'images.

Premièrement, un algorithm *Avare* d'attribution de niveau (GRLA) est conçu basé sur la philosophie de l'algorithm Avare d'attribution de bits, parcontre attribuant niveau par niveau en considérant le meilleur bénéfice et le coût d'une attribution de niveau. Le GRLA est très onéreux au point de vue calculs et, donc, sa pratiqualité est discutable et l'algorithme doit être combiné avec un algorithm plus simple.

Deuxièmement, un algorithme résolvant les problèmes d'attribution négative de bits et de nombres entiers de niveaux est proposé. L'attribution de niveau doit garder un certain rapport entre les dimensions pour avoir une distortion la plus basse. De plus, la formule originale d'attribution de bits sous-entend un gain de 6dB par bit alors qu'en général, ce n'est pas le cas. L'algorithm présente un nouveau paramètre k qui contrôle le bénéfice d'une addition d'un bit, habituellement ajusté à 0.5 dans la formule optimale pour haut débit binaire. Les simulations montrent que la modification de la formule d'attribution de bit amène un gain.

La formule contenant le nouveau paramètre est ensuite généralisé en une formule introduisant un nouveau paramètre qui varie aussi dépendemment de la fonction de distribution de chaque dimension. L'algorithme trouvant le k optimal décide *a-posteriori* du k optimal alors que cet algorithme général utilise une procédure d'entraînement pour estimer le facteur pesant l'attribution de bit de cette dimension. Les résultats de la simulation montrent encore une amélioration sur les méthodes précédentes.

Acknowledgments

I would like to thank my professors, Dr. Peter Kabal for guidance, supervision, financial support and kind presence, Dr. Richard D. Rose for financial support and friendliness and Dr. Fabrice Labeau for providing help with writing papers.

All of TSP laboratory, especially Dr. Turaj Zakizadeh Shabestary for solid help on theory and practice, Wei-Shou Hsu for miscellaneous help on how to use a computer, Alexander Wyglinski for all the details on Rules and Regulations of Graduate Studies, and Patrick Kechichian for the great team work on preliminary research, all without whom graduation may have been but a dream.

And finally, last but not least, family and friends for constant love, support and keeping me sane throughout the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Need for Compression

Nowadays, most signals are processed digitally rather than using traditional analog devices. Telephone conversations, photos and movies all go through analog-to-digital conversion before being processed digitally and replayed.

For digital transmission of speech signals, lower bit rates make more efficient use of transmission bandwidth. Digital images can also be compressed to occupy less storage space.

Source coding exploits speech and image predictibility through various coding strategies. It can be used to efficiently compress signals such as speech or images.

1.2 Linear Prediction

Many low rate speech coders use linear prediction (LP), described in Appendix A, where the current sample is predicted with a weighted linear combination of previous samples. These coefficients can be adapted at each time frame. The residual error is sent, but since the decoder needs to know the predictive coefficients in order to synthesize the original signal, the coefficients need to be sent as side information as shown in Figure 1.1. Efficiently coding the LP coefficients will reduce the side information.

LP is simply a generalization of the differential pulse code modulation (DPCM)[12] scheme to a higher predictive order. From the LP coefficients of a frame, which can be seen as a linear filter, one can reconstruct the spectral envelope of the signal for the given

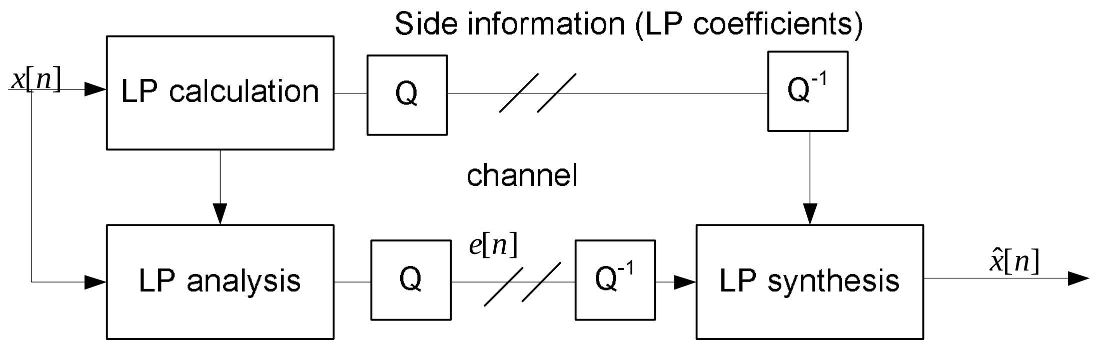


Fig. 1.1 Block Diagram of an LP coder

frame. In fact, the same procedure can be undertaken to transform a time-domain filter into a frequency response on the LP coefficients and one would get the frequency spectrum contents of the frame that was just analyzed. The higher the order of the LP coefficients, the more precisely the envelope will be approximated.

This coding scheme has great compression capability, however, the coefficients need to be quantized themselves. Small rounding errors in direct form quantization of these coefficients can yield a large error when extracting the spectral envelope. Therefore, different representations of the LP coefficients have been developed to improve quantization noise robustness. One such representation is called line spectral frequencies (LSF) also known as line spectral pairs (LSP), described in more details in Appendix A.

1.3 Image and Transform Coding

In image coding, an image is often split into blocks, typically 8 x 8 pixels. Each block of data is then transformed using a transformation such as the Karhunen-Loève transform (KLT) or the Discrete Cosine Transform (DCT) hoping to decorrelate the block. Quantization is applied to those transformed coefficients.

One can use the Karhunen-Loève transform (KLT), described in Appendix B, in order to decorrelate the data set. The KLT is a data-dependent transform and, when applied to the data set, is known to diagonalize the covariance matrix, effectively decorrelating the data set.

The Gaussian Mixture Model (GMM) based coder, described in section 2.8, is also a transform coder. The GMMs model an arbitrary distribution function as a sum of many single Gaussian mixtures, or clusters, each having its own average and covariance matrix. The GMM-based coder then transforms the data using the optimal KLT trained specially for each cluster. A vector to be coded, in our case a representation of the LP coefficients called the LSF defined in Appendix A, is coded using the coder described in section 2.8 for each cluster. The cluster yielding the least distortion is chosen as the best path. Using a GMM also allows the use of standard parameter estimation such as the Expectation-Maximization (EM) developed in Appendix C.

1.4 Quantization

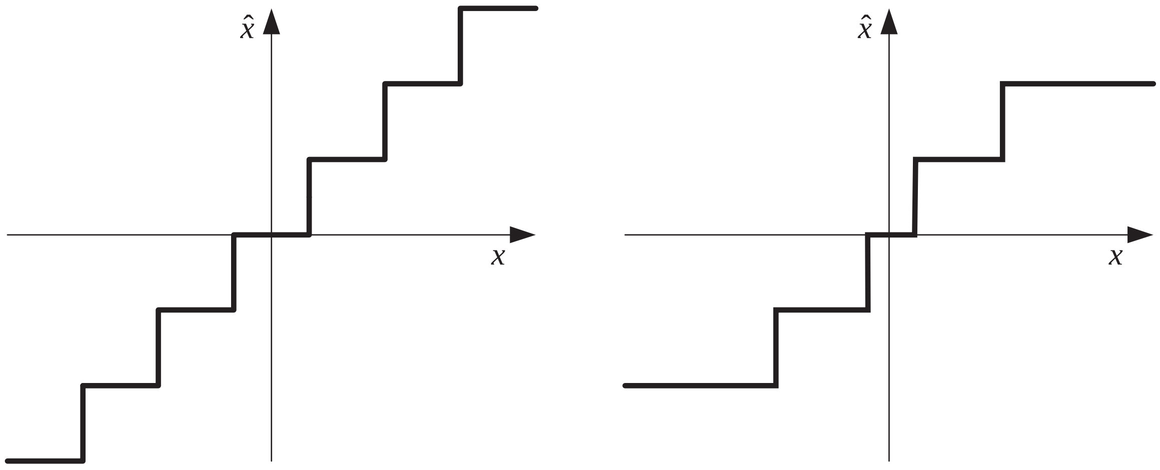
When digitizing an analog signal, the signal needs to be quantized to occupy finite space or bits in the digital world. There are two ways of quantizing a signal.

1.4.1 Scalar Quantization

The first way is using a scalar quantizer (SQ). The scalar quantizer is the simplest way of quantizing a sample: the sample is simply mapped to the closest level in the quantizer. At the decoder, the received signal is mapped to a value, usually the level itself. There also exists two types of scalar quantizers: uniform and non-uniform as seen in Figure 1.2. A uniform quantizer has equally spaced levels. This is a simple and computationally efficient quantization since the quantization regions are highly organized as seen in Figure 1.2(a), efficient quantization methods such as rounding can be used. The non-uniform quantizer,as seen in Figure 1.2(b) permits finer quantization of regions of larger interest, i.e. with more probability of occurrence, such as the region around the mean for a Gaussian distribution function. However, the non-uniform quantizer requires more computationally intensive quantization involving search to find the optimal quantization point since the regions are unequally spaced. A compandor can be used to compress the quantization levels and effectively implement a non-uniform quantizer using a uniform quantizer, quantization then becomes simple and computationally efficient as one can use rounding functions.

1.4.2 Vector Quantization

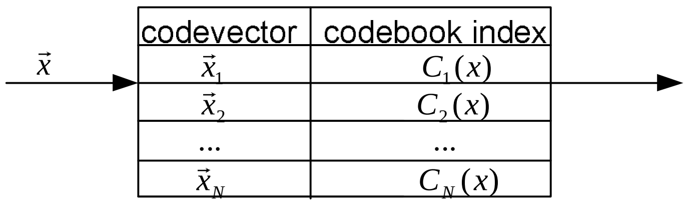
The second way is using vector quantization (VQ). To maximally exploit correlation between samples, one can code a vector of samples rather than one by one independently



(a) Uniform Scalar Quantizer (b) Non-Uniform Scalar Quantizer

Fig. 1.2 Types of Scalar Quantizers

matching it to a most corresponding vector of points(codebook entry or codevector), then only the codebook index is sent. This is vector quantization (VQ) as seen in Figure 1.3, where C_i is the codebook index associated with the codevector x_i. Codebook entries can be chosen such that, for a training set and a given distortion measure, there will be least distortion, and thus and optimal distribution of codevectors. However, this comes with memory requirements and computational cost as the codebook needs to be stored and the best matching point to the vector needs to be searched for.



codevector	codebook index
$\vec{x}_1$	$C_1(x)$
$\vec{x}_2$	$C_2(x)$
...	...
$\vec{x}_N$	$C_N(x)$

Fig. 1.3 Block Diagram of a Vector Quantizer

VQ has three advantages over SQ. The first advantage is memory advantage, which only exists when the variables to be coded are correlated. It relates to the coder knowing that if dimension 1 is a certain value, then, dimension 2 is likely to be in a certain range. The second advantage is the space-filling advantage which is related to the form of a quantization cell. When applying SQ to n-dimensions, the quantization cells are hypercubes whereas VQ can have hexagonal n-dimensional polygons, which reduces the average error. The third advantage is the shape advantage and is related to the more optimal way of VQ to place codevectors in d-dimensions depending on the marginal probability density functions (PDF) when compared to the SQ. These advantages are discussed in more details in Appendix D.

A VQ which simply contains many entries, where the distance between a vector to be coded and a codebook entry (centroid) is computed and the index of the entry giving the least distortion taken as the code is called an unstructured VQ. This type of VQ, where the centroids are placed without order is named in contrast with a structured VQ where the centroids are placed following a way which makes it unnecessary to have an exhaustive search through all centroids to find a suitable code. An SQ is an example of a structured VQ.

There are two major problems associated with an unstructured VQ. First, all the codevectors and indices have to be stored in a codebook which number of entries grows exponentially with the bit rate and quickly grows into unrealizable sizes. The second problem is computational cost. The encoder finds the best codevector to represent each vector by a search through the codebook entries, computing distortion for each. Since the codebook size grows exponentially with the bit rate, the computational cost at the encoder also grows exponentially.

1.5 Transform Coding

As stated in the previous section, the SQ is more computationally realizable than the VQ at the cost of the three advantages. The SQ can approach the performance of the VQ using the mean squared error (MSE) as the distortion measure, assuming high-rate if the variables to be coded are independent, in that way, the VQ is no longer needed to exploit the memory advantage. Thus, it is desirable to make the variables independent by means of a transform.

The high-rate assumption refers to a bit rate that is high enough so that the benefit of adding one bit to the quantizer yields a gain sufficiently close to +6dB.

1.6 Gaussian Mixture Models

Gaussian Mixture Model (GMM)-based quantization is an approach which models data distribution by several Gaussian mixtures and uses a different transform coder for each mixture, treating each cluster as a Gaussian.

A GMM-based quantizer often uses a quantizer designed for each mixture having a compander and an SQ for each dimension avoiding the need for a codebook, therefore, one can use a GMM instead of a VQ. GMM-based quantizers are an example of structured VQ, which does not need a codebook because the codevectors are well structured.

1.7 Bit Allocation

A vector of random variables usually has a different quantizer for each dimension because each dimension has a different distribution function. The distortion yielded by each quantizer is a function of the number of bits allocated to that dimension's quantizer. One should distribute bits in a manner so that the total distortion is minimized, discriminating more important dimensions from less important ones as seen in Figure 1.4 in which one sees that giving more bits to dimension x_1 is more beneficial than bits to dimension x_2.

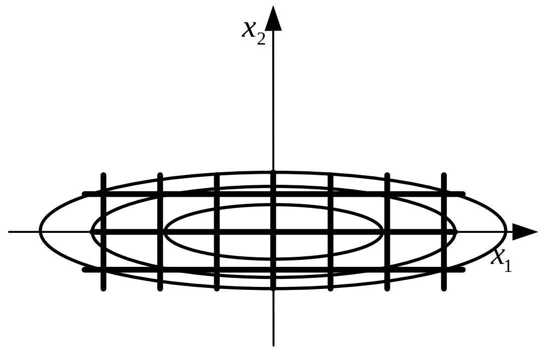


Fig. 1.4 Level Allocation where the straight lines are the quantization levels and the concentric oval shapes represent equiprobable contour lines

Huang and Schultheiss derived an optimal bit allocation formula assuming high-rate and Gaussian distribution functions [2]. The derived formula did not put a constraint on integer number of levels or non-negative bit allocation. Moreover, they assume high-rate, which is not usually the case. The method used by Subramaniam is a *Pruning* algorithm where levels are overallocated and then removed until there are just enough. The problem lies in keeping the proper ratio of levels between the dimensions which the Pruning algorithm does not do, this requirement will be shown in section 2.4.2. Segall [6] designed another method which takes into account low-rate and constrains the allocations to non-negative bit allocations. It can however get quite far from the required bit rate when rounding down the number of levels of each dimension to ensure integer number of levels. The *Greedy* bit allocation algorithm is also a very popular method which consists of distributing bits one by one to the dimension with the current largest need until there are no bits left. The Greedy bit algorithm, due to its integer bit nature, has trouble getting to the optimal bit allocation ratios and would often have a few dimensions with too many levels and others with too few.

1.8 Description of Thesis Work

The basis LSF quantizer structure used in this thesis is the quantizer described by Subramaniam [1]. The coder in question is simply a transform coder which uses GMM to cluster the data first, and then, for each cluster, the KLT decorrelation matrix associated with the covariance matrix of the cluster is used to transform the data. This scheme is quite simple and yet powerful as the data distribution in each cluster approaches that of a Gaussian, as the number of Gaussian mixtures tends to infinity.

The bit allocation has often been done using *ad hoc* methods to fix the problem of negative bit allocations and of non-integer bits, or levels, allocations. Moreover, some of those methods assume Gaussian distribution, which may not be the case. And finally, some of them do not make good use of the given bits and end up wasting fractions of bits.

A method is proposed to iteratively prune dimensions with negative bit allocations, increase the wanted bit rate and use the well-known optimal bit allocation formula given in Chapter 2 (or a modified version) to allocate bits to the remaining dimensions, i.e. those having high enough variances.

The proposed method does not get close enough to the required bit rate, because it keeps the ratio of levels between the dimensions, it wastes a lot of bits. Therefore, a method following the philosophy of the Greedy algorithm is designed to allocate levels in a Greedy way, i.e. level-by-level until none can be allocated anymore. Because of high computational complexity, the Greedy Level Allocation algorithm (GRLA) is simply used as a filler algorithm, i.e. it acts after the bit allocation formula. The GRLA allocates bits according to a definition of the benefit-to-cost ratio, different definitions and their effects are studied.

Another problem lies in the optimal bit allocation formula itself: the bit allocation formula assumes high-rate and Gaussian distribution or uniform distribution. In nowadays applications, high-rate is rarely a valid assumption, and Gaussianity is only valid if one uses a GMM with an infinite (or very high) number of mixtures. Therefore, the optimal bit allocation formula will have to be modified to match the requirements of the quantizer. The proposed method called the Two-Stage Iterative Bit Allocation (TSIBA), developed in Chapter 4, uses a factor k to change the benefit of adding one bit in a scalar quantizer, which is usually assumed to be 6dB per bit. The proposed method gave quite remarkable improvements over other bit allocation schemes, i.e. the Greedy algorithm, Segall's method and the Pruning algorithm described earlier.

This method is then generalized to use a different factor for each dimension since each dimension has a different distribution and number of bits allocated to them. The performance of the generalized algorithm gives even better performance and adds a possibility of training rather than an *a-posteriori* decision on which factor k to use for each dimension as in the TSIBA.

The proposed methods are adapted to the probability density functions and therefore are expected to yield some good performance gain.

1.9 Thesis Organization

This thesis topic is bit allocation. Different bit allocation methods will be presented and results are shown comparing these methods with more standard techniques such as the Greedy algorithm, the method designed by Segall [6] and the Pruning algorithm used by Subramaniam.

The second chapter presents some background information about the coder designs that will be used throughout the work.

The third chapter shows a method of distributing the levels one-by-one using a Greedy-like algorithm. Many definitions of the benefit-to-cost ratio used to decide which dimension should be allocated the levels are studied and the results are compared. The concept of levels allocation flattening and unflattening is also discussed.

The fourth chapter shows a bit allocation algorithm which keeps the ratio of bits between dimensions, and fixes the problem of negative bit allocations and non-integer levels. This method also varies the benefit of adding a bit to a dimension, making the levels allocation flatten and unflatten depending on the value of the chosen benefit. Then, since the proper benefit depends on bit rate and distribution function, the algorithm is generalized to give different benefits for each dimension to better match the benefits at the bit rates and for the distribution in question.

And finally, the fifth chapter summarizes and concludes what has been done in this work.

Chapter 2

Coder Designs

2.1 Gaussian Mixture Models

The transform coder yields great performance given that the input has Gaussian distribution which, in practice, is rarely the case. The distribution could be some other probability density function (PDF) such as Laplacian, Gamma or general multimodal distribution. If one were to model data as one Gaussian and have a uniform scalar quantizer, the resulting distortion could be large since most codepoints are rarely used and precision would be lost for the regions of interest as shown on Figure 2.2a). Therefore, one could model the distribution of data with several Gaussian mixtures rather than a single one, each having their own quantizer and transform as shown on Figure 2.2b), then the distribution inside each cluster would approach a Gaussian PDF and thus, the assumption of Gaussianity would be more valid. For instance, in modelling LSFs, data is most likely to be clustered since to each sound forming a phoneme, there is a set of formant frequencies associated [12]. Therefore, the use of GMM to represent this clustered distribution as seen in Figure 2.1 is justified.

However, one drawback caused by using GMM would be in case of overlap of Gaussians which becomes more important as the number of Gaussians increases. An overlap wastes levels because levels are allocated in the overlap region by both mixtures whereas they could be used elsewhere, such as in non-overlapping regions.

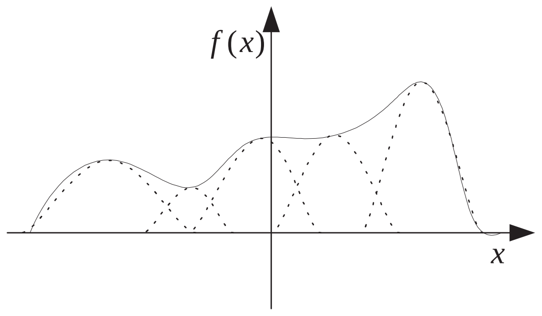
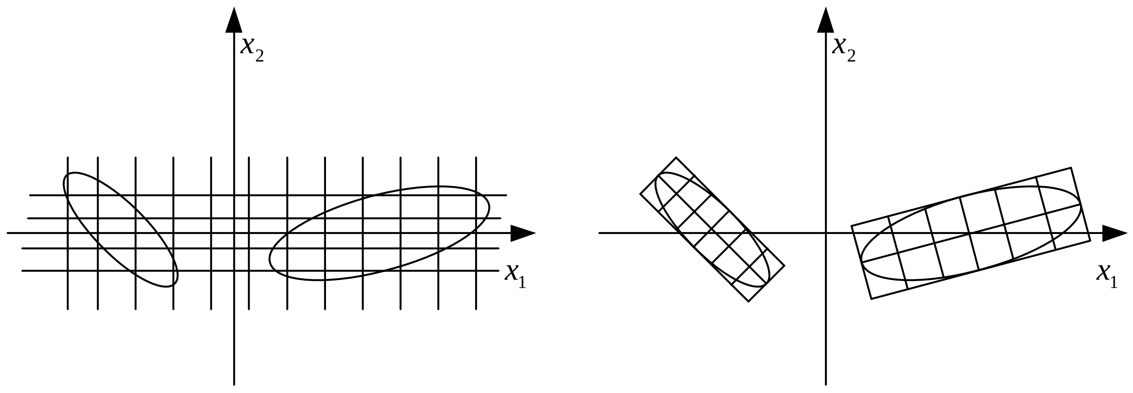


Fig. 2.1 Gaussian Mixture Models to Represent a Distribution Function



(a) Bit Allocation not Using GMM (b) Bit Allocation Using GMM

Fig. 2.2 Effect of GMM on Bit Allocation

2.1.1 Expectation-Maximization Algorithm

Another advantage of using GMM is to be able to use the EM algorithm to estimate the parameters of the model. The Expectation-Maximization (EM) algorithm is a maximum-likelihood estimation algorithm which is both simple and stable. When using the EM algorithm, the likelihood of the training data is guaranteed to not decrease every iteration [14] until it reaches a local optimum. The proof of convergence and derivation of the update equations for the GMM parameters can be found in Appendix C.

2.2 Distortion Measures

Algorithms need to be compared to determine which ones give the best performance. Furthermore, often an algorithm will have several paths it can follow, the decision as to which path is the best will be made based on an error criterion, in which the minimum error will be chosen as the best path.

There exist several distortion measures that are used in various situations, the three that will be used in this work are the mean squared error (MSE), the peak signal-to-noise ratio (PSNR), and the log spectral distortion (LSD).

2.2.1 Mean Square Error

The MSE relates to the Euclidean distance between the original signal and the quantized signal in d dimensions. If the signal is a time signal or frequency spectrum, then the MSE is also related to the power of the error.

This distortion measure is simply:

$$MSE = \sum_{i=1}^{d} (\hat{x_i} - x_i)^2 \tag{2.1}$$

where d is the number of dimensions, x is the input signal, x_i is the ith component of the vector x and $\hat{x}$ is the quantized signal.

This distortion criteria has the advantage of very low complexity, however, in many cases, the MSE is not subjectively meaningful [9]. For instance, in the case of quantizing Line Spectral Frequencies (LSF), it is clear that certain coefficients will affect the frequency spectrum more than others. The importance of each coefficient is measured in a sensitivity matrix, which is the derivative of the log spectral distortion (LSD) with respect to the dimension in question. The LSD, to be defined next, can also be viewed as a weighted mean-square error, where the weight is data-dependent [17].

2.2.2 Log Spectral Distortion

The log spectral distortion (LSD) is the preferred distortion measure for LSF quantization. The LSD relates to the differences in frequency spectral envelope and therefore has more significance in comparing the quantized LSF to the original LSF vector as seen in Figure 2.3. Moreover, the log spectral distortion is considered to be a perceptually meaningful measure. For transparent quality on the spectral envelope as side information, i.e. when the quantized LP coefficients are recombined with the residual signal, as shown in Figure 1.1, the degradation is not perceptible, the equivalent LSD benchmark would be an average LSD lower than 1dB, a percentage of outliers 2dB–4dB of less than 1% and no outliers larger than 4dB [17].

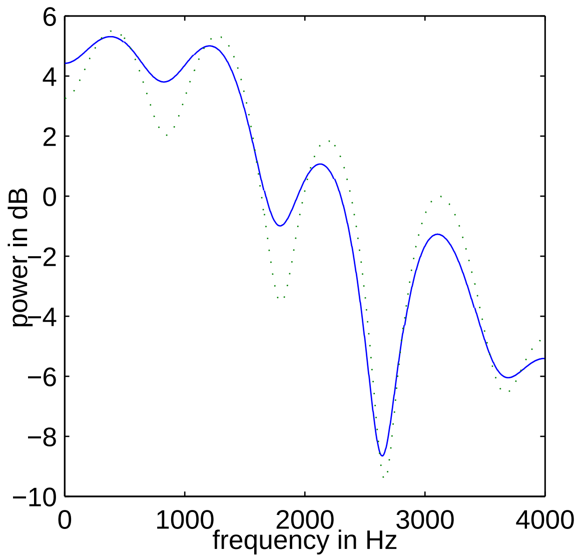


Fig. 2.3 Log Spectral Distortion, Solid Line: Original Signal Spectrum; Dotted Line: Quantized Signal Spectrum

The formula for computing the LSD (in dB) is

$$SD^2_{f_s/2} = \frac{20^2}{f_s/2} \int_0^{f_s/2} \left(\log_{10} \frac{\left|H(e^{j2\pi f})\right|}{\left|\hat{H}(e^{j2\pi f})\right|} \right)^2 df \qquad (2.2)$$

where f_s is the sampling frequency, $H(e^{j2\pi f})$ is the frequency representation of the spectral envelope and $\hat{H}(e^{j2\pi f})$ is the quantized frequency representation of the spectral envelope. This formula involves an integral, which can be approximated by a sum of a finite number of terms.

There are 2 ways of computing the average spectral distortion: root-mean-square(RMS) and mean-root square(MRS).

RMS is defined by

$$\overline{SD}_{RMS} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} SD_n^2} \qquad (2.3)$$

and MRS is defined by

$$\overline{SD}_{MRS} = \frac{1}{N}\sum_{n=1}^{N} SD_n \qquad (2.4)$$

where SD is the log spectral distortion of the LP coefficients and N is the number of points used to approximate the spectral distortion. In this work, the MRS definition is used as it is most commonly used in other works.

The cost of computing the LSD is much higher than computing the MSE as can be seen by comparing (2.4) and (2.1).

For speech signals, since the spectrum has low power at high-frequencies, spectral distortion will be greatly increased for those regions. Therefore, for an 8kHz sampling rate, spectral distortion is often calculated from 0–3000Hz, as is done in this work, rather than 0–4000Hz keeping in mind that the signal is band-limited [17].

2.2.3 Peak Signal-To-Noise Ratio

The *PSNR* is the distortion measure that will be used for image coders. The *PSNR* is closely related to the MSE and thus inherits its properties and therefore does not relate directly with human perception, but it is a simple measure to compute and gives a good idea of the quality of the reconstructed image. Typical *PSNR* values used for reasonable image quality are in the range of +25dB to +35dB.

The *PSNR* is the measure to be used in image coding. Its definition is

$$PSNR = 10 \log_{10} \left(\frac{(\max_i(x_i))^2}{MSE} \right) \tag{2.5}$$

where $\max_i (x_i)$ is the maximum value of x.

2.3 Transform

There are a few properties that a good transform should have. The first one is invertibility: one wants to use the transform at the encoder and use the inverse for the decoder. The second property is preserving distortion. A unitary transform is usually chosen because it preserves the MSE distortion. A typical way of using a transform coder is shown in Figure 2.4, where T is the transform and T^{-1} is the inverse transform.

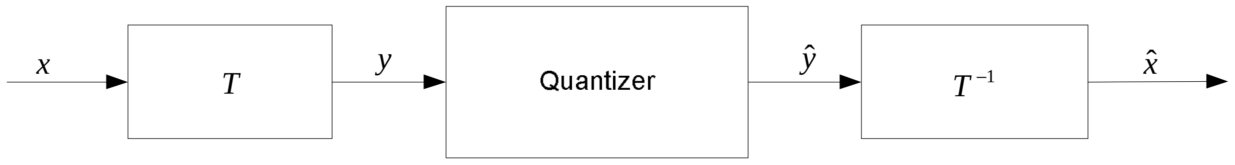


Fig. 2.4 Block Diagram of a Transform Coder

A popular transform is the Karhunen-Loève transform, which is used to decorrelate a set of data through an orthogonal transform. Its benefits and derivation are outlined in Appendix B. The KLT is the transform used in this work.

2.4 Bit Allocation

The bit allocation problem is the core interest of this work. The methods that will be presented in the following chapters will be compared to a few more traditional bit allocation formulas.

2.4.1 Traditional Optimal Bit Allocation formula for MSE

The bit allocation will be done in such a way that the MSE over all dimensions is minimized. The derivation can be found in [2].

Assuming $D_i = A_i(b_i)2^{-2b_i} = A_i(b_i)e^{-2b_i \ln 2}$ to be the distortion over dimension i, where D_i is the MSE distortion i, $A_i(b_i)$ is a factor weighting the benefit of adding a bit to dimension i, b_i is the number of bits allocated to dimension i. Let us rewrite $A_i(b_i)$ as $K_i(b_i)\lambda_i$, and approximate $K_i(b_i)$ by K, which is a constant number, and λ_i is the variance of dimension i, the total distortion can be written as

$$D \approx \sum_{i=1}^{d} \lambda_i K e^{-2b_i \ln 2} \tag{2.6}$$

And now, using a Lagrange multiplier, we minimize the distortion under the constraint that $\sum_{i=1}^{d} b_i = b_{tot}$.

$$\frac{\partial \left[(1/N) \sum_{i=1}^{N} \lambda_i K e^{-2b_i \ln 2} + \beta \sum_{i=1}^{N} b_i \right]}{\partial b_j} = 0$$

$$-(1/N)\lambda_j K 2 \ln 2 e^{-2b_j \ln 2} + \beta = 0$$

then

$$\lambda_j e^{-2b_j \ln 2} = \frac{N\beta}{K 2 \ln 2} = C \tag{2.7}$$

for $j = 1, 2, ...N$.

And now, solving for b_j,

$$\begin{aligned} b_j &= \bar{b} + \frac{1}{2 \ln 2} \left[\ln \lambda_j - (1/N) \ln \left(\prod_{i=1}^{N} \lambda_i \right) \right] \\ &= \bar{b} + \frac{1}{2 \ln 2} \left[\ln \lambda_j - (1/N) \ln \left(|M_x| \right) \right] \\ &= \bar{b} + 1/2 \log_2 \frac{\lambda_j}{(|M_x|)^{(1/N)}} \end{aligned} \tag{2.8}$$

where $\bar{b}$ is the average bit rate M_x is the covariance matrix of x and $|M_x|$ is the determinant of M_x.

The formula (2.8) can be shown to be equivalent to requiring the ratio of levels allocated to dimension i and dimension j to be equal the ratio of the standard deviation of dimension i and dimension j.

$$\begin{aligned}
b_i &= \bar{b} + 1/2 \log_2 \frac{\lambda_i}{(|M_x|)^{(1/N)}} \\
2_i^b &= 2^{\bar{b}+1/2\log_2 \frac{\lambda_i}{(|M_x|)^{(1/N)}}} \\
2_i^b / 2_j^b &= \frac{2^{\bar{b}+1/2\log_2 \frac{\lambda_i}{(|M_x|)^{(1/N)}}}}{2^{\bar{b}+1/2\log_2 \frac{\lambda_i}{(|M_x|)^{(1/N)}}}} \\
L_i/L_j &= \frac{2^{\bar{b}}}{2^{\bar{b}}} \frac{2^{1/2\log_2 \frac{\lambda_i}{(|M_x|)^{(1/N)}}}}{2^{1/2\log_2 \frac{\lambda_j}{(|M_x|)^{(1/N)}}}} \\
L_i/L_j &= \frac{2^{\log_2 \frac{\lambda_i^{1/2}}{(|M_x|)^{(1/2N)}}}}{2^{\log_2 \frac{\lambda_j^{1/2}}{(|M_x|)^{(1/2N)}}}} \\
L_i/L_j &= \frac{\lambda_i^{1/2}}{\lambda_j^{1/2}} \\
L_i/L_j &= \frac{\sigma_i}{\sigma_j}
\end{aligned} \tag{2.9}$$

where L_i is the number of levels allocated to dimension i and σ_i is the standard deviation of dimension i, which is also $\lambda_i^{1/2}$.

This provides a closed-form expression for the optimal bit allocation but suffers two problems: some dimensions may get negative bit allocation resulting from the $\log_2$ term for small variances and fractional bit rates because the bit allocation does not guarantee an integer number of levels for all dimensions.

2.4.2 Intuitive Derivation of Bit Allocation Formula

There has been many derivations of the optimal bit allocation formula. Huang and Schultheiss [2] derived it using an approximation of the distortion and assuming the total distortion to be the sum of distortions among the dimensions. The derivation is a standard optimization procedure involving one constraint: the sum of bits must be equal to the desired bit rate. This derivation is an elegant mathematical derivation and yields the desired formula. However, the assumption of the variable $A_i(b_i)$ in the distortion formula for dimension i is very hard to integrate into a more general formula.

Here we present a derivation that is simple and intuitive, yielding the same result yet easily generalized to introduce the variable $K_i(b_i)$ term.

Assuming $D_{\text{tot}} = \sum_{i=1}^{d} D_i$ and $D_i = A_i(b_i)2^{-2b_i}$ [2]

$$D_{\text{tot}} = \sum_{i=1}^{d} A_i(b_i)2^{-2b_i} \tag{2.10}$$

where $2^{-2b_i} = 1/L_i^2$ and d is the number of dimensions.

Let us rewrite $A_i(b_i)$ as $K_i(b_i)\lambda_i$, and approximate $K_i(b_i)$ by K, which is a constant number, we get

$$\begin{aligned} D_{\text{tot}} &= K \sum_{i=1}^{d} \frac{\lambda_i}{L_i^2} \\ &\leq K d \max_i \frac{\lambda_i}{L_i^2} \end{aligned} \tag{2.11}$$

with equality when all the λ_i/L_i^2 are equal. It is then obvious that to minimize the total distortion, one needs to minimize the bound, which means minimizing $\max_i \lambda_i/L_i^2$ and therefore making the terms equal as it was also seen in section 2.4.1 in (2.9).

So we want to make all the λ_i/L_i^2 terms equal to a constant. Let

$$\frac{\lambda_i}{L_i^2} = 1/\gamma. \tag{2.12}$$

We know furthermore that

$$\begin{aligned}\prod_{i=1}^{d} L_i &= 2^{b_{tot}} \\ \prod_{i=1}^{d} L_i^2 &= 2^{2b_{tot}}.\end{aligned} \tag{2.13}$$

Combining (2.13) and (2.12), one gets

$$\begin{aligned}\prod_{i=1}^{d} L_i^2 &= \gamma \prod_{i=1}^{d} \lambda_i \\ \gamma &= \left(\frac{2^{2b_{tot}}}{\prod_{i=1}^{d} \lambda_i} \right)^{\frac{1}{d}} \\ &= \frac{L_i^2}{\lambda_i}.\end{aligned} \tag{2.14}$$

Then use this γ in the original equation for levels

$$\begin{aligned}L_i^2 &= \lambda_i \gamma \\ L_i^2 &= \lambda_i \left(\frac{2^{2b_{tot}}}{\prod_{i=1}^{d} \lambda_i} \right)^{\frac{1}{d}} \\ 2\log_2 L_i &= \log_2 \frac{\lambda_i}{\left(\prod_{i=1}^{d} \lambda_i \right)^{\frac{1}{d}}} \left(2^{\frac{2b_{tot}}{d}} \right) \\ b_i &= \frac{b_{tot}}{d} + 0.5 \log_2 \frac{\lambda_i}{\left(\prod_{j=1}^{d} \lambda_j \right)^{\frac{1}{d}}}\end{aligned} \tag{2.15}$$

which is the well-known bit allocation formula.

We can simply reintroduce the $K_i(b_i)\lambda_i$ term following the derivation as it was approximated as $K\lambda_i$. And then, we would get

$$b_i = \frac{b_{tot}}{d} + 0.5\log_2 \frac{\lambda_i K_i(b_i)}{\left(\prod_{j=1}^{d} \lambda_j K(b_j)\right)^{\frac{1}{d}}} \tag{2.16}$$

which is an equation that can be solved iteratively.

The function $K_i(b_i)$ is distribution dependent. Lloyd [3] develops a few examples for the uniform, Gaussian and Laplacian distributions. In our problem, data is assumed to be Gaussian distributed. Thus, for our coder, one could assume $K_i(b_i)$ to be constant and thus, yield the well-known bit allocation formula. From Lloyd [3]

$$\begin{aligned} A_i(b_i) &= E_q \cdot L_i^2 \\ K_i(b_i) &= \frac{E_q \cdot L_i^2}{\lambda_i} \end{aligned} \tag{2.17}$$

where E_q is the quantization noise of the optimal quantizer for dimension i and L_i is the number of levels allocated to dimension i.

However, even for a Gaussian distribution, at low bit rates, $K_i(b_i)$ is dependent on b_i [4].

Let us now confirm the fact that the bit allocation formula makes all $\frac{\lambda_i}{L_i^2}$ terms equal to each other. We will enter hypothetical variances and see what the bit allocation formula gives us

Table 2.1 Bit Allocation Versus Variances

σ_i^2	14	3	2	1	0.1
σ_i	3.74	1.73	1.41	1.00	0.32
b_i	21.60	20.49	20.19	19.69	18.03
L_i	3.17×10^6	1.47×10^6	1.20×10^6	8.48×10^5	2.68×10^5
σ_i/L_i	1.18×10^{-6}	1.18×10^{-6}	1.18×10^{-6}	1.18×10^{-6}	1.18×10^{-6}

where σ_i^2 is the variance of dimension i, σ_i is the standard deviation of dimension i, b_i is the number of bits allocated to dimension i and L_i is the number of levels allocated to dimension i.

From this table of level allocation, one can see indeed that following the original bit allocation formula (2.15), the resulting bit allocation will make all the σ_i/L_i or λ_i/L_i^2 equal. Moreover, scaling down the total number of bits would scale down equally the average bit rate component in the bit allocation formula (2.15), but the ratio between the dimensions would be kept constant, if there are enough bits and it will be shown in Chapter 4 what happens when it is not the case, to preserve the λ_i/L_i^2 equality.

2.5 Greedy Bit Allocation Algorithm

2.5.1 Motivation and Background on Greedy Bit Algorithm

The optimal bit allocation formula derived by Huang and Schultheiss [2] has two problems: it does not guarantee that all dimensions have non-negative bit allocation and it does not guarantee that each dimension is allocated a number of bits resulting in an integer number of levels (2^{b_i}).

Segall [6] derived an algorithm which solves the problem of negative bits allocation. This algorithm, furthermore, can take care of low bit rate bit allocation.

The Greedy bit allocation algorithm is based on the procedure given by Fox [7] on discrete optimization via marginal analysis and is proposed to avoid these two problems. The Greedy bit allocation algorithm is optimal for low-rate integer bit allocation because it allocates bits to the dimensions with biggest need under the condition that distortion-rate functions are concave and assumption that the total distortion D_{tot} is the sum of individual distortions D_i.

The additive nature of the MSE is essential for the Greedy bit allocation algorithm, it permits one to separately determine which dimension has the largest need without affecting other dimensions. For the LSD, the error in one dimension might affect the total distortion more than the error in another dimension, therefore the LSD is not an additive distortion measure, the Greedy algorithm would perform quite poorly since the assumption is violated.

2.5.2 The Greedy Algorithm Based on Discrete Optimization via Marginal Analysis

Here is the procedure that Fox [7] gives for allocating resources:
Algorithm 2.5.1: DISCRETE OPTIMIZATION($c(x), \phi(x)$)

```
comment: Initialize all allocations to 0
1.$x^0 = \mathbf{0}$
2.$k = 1$
3.$x^k = x^{k-1} + e_i$
4.if $C(x^k) > M$
  then done,
  else $k \leftarrow k + 1$ and go to step 3
```

where e_i is the ith unit vector and i is the index for which

$$\frac{\phi_j(x_j^{k-1}+1) - \phi_j(x_j^{k-1})}{c_j(x_j^{k-1}+1) - c_j(x_j^{k-1})} \tag{2.18}$$

is maximum, where $c_j(x_j^k)$ and $\phi_j(x_j^k)$ are respectively the cost and error associated with dimension j at allocation k, $C(x^k)$ is the total cost over all dimensions.

The Greedy algorithm is due to Fox [7]. In [7], it is said that the criterion to be maximized is (2.18) in which j is the dimension and k is the kth allocation and in which $\phi_j(x_j^{k-1})$ must be concave and strictly increasing and $c_j(x_j^{k-1})$ is convex and strictly increasing guaranteeing that the allocations are *undominated*.

Allocations x satisfying

$$\begin{aligned} \phi(y) > \phi(x) \rightarrow C(y) > C(x) \\ \phi(y) = \phi(x) \rightarrow C(y) \geq C(x) \end{aligned} \tag{2.19}$$

for all y are called undominated or efficient, i.e. no other allocation gives a smaller distortion for the same cost.

2.5.3 Application to Bit Allocation

Equation (2.18) has the form of a ratio of benefit-to-cost. The benefit is defined by the reduction in MSE and the cost is the cost in bits of such an allocation. This is done by setting $-\phi_j(x_j^{k-1})$ to be the distortion-rate function, which is, for log-convex distribution functions, concave and stricly increasing and $c_j(x_j^{k-1})$ to be the functions which converts levels to bits, i.e. $\log_2(x)$ which is a convex and strictly increasing function. Therefore, the conditions for (2.18) to be *undominated* are satisfied.

Replacing the variables used by Fox to the variables applicable in the bit allocation context, where x_j^{k-1} is the number of levels L_j^k allocated to dimension j at allocation $k-1$ and $-\phi_j(x_j^{k-1})$ is the MSE distortion $D_j(L_j^k)$, the equation is then rewritten as

$$\frac{D_j(L_j^k+1) - D_j(L_j^k)}{\log_2(L_j^{k-1}+1) - \log_2(L_j^{k-1})}. \tag{2.20}$$

2.5.4 The Greedy Bit Allocation algorithm

The concept is intuitive: allocate bits to dimensions that most need them. It is actually an iterative procedure where at each iteration, one bit is distributed to the dimension with largest demand, i.e. variance in the MSE case, thus reducing the distortion of the dimension, and consequently, the total distortion. In this way, the Greedy algorithm only sees the benefit of the next allocation and not any further.

The Greedy algorithm has the advantage of being very simple and optimal under the assumption that $\sum D_i = D_{tot}$ and the distortion in a dimension decreases for every bit given to that dimension. It avoids the problem of negative bits allocated to dimensions with low variance and of non-integer bits.

However, the Greedy algorithm inserts another constraint: to each dimension must be allocated an integer number of bits. This yields a number of levels which is a power of 2 and each dimension can be coded independently. This creates its own problem as it makes the allocation farther from the ratio found using the bit allocation formula and thus less optimal as seen in the Chapter 1.

Here is an outline of the Greedy algorithm from [9]:

Algorithm 2.5.2: GREEDY($bits, D_i(b_i)$)

```
comment: Initialize variables
for each $i \in 1$ to $N$
  do $b_i = 0$
$bits_{remain} = bits$

comment: Distribute bits 1 by 1
repeat
  $j = arg\max_i (D_i(b_i) - D_i(b_i + 1))$
  $b_j \leftarrow b_j + 1$
  $bits_{remain} \leftarrow bits_{remain} - 1$
until $bits_{remain} = 0$
```

where b_j is the number of bits allocated to dimension j, $bits_{remain}$ is the number of bits remaining after each allocation and $D_i(b_i)$ is the distortion of dimension i with b_i bits.

The $D_i(b_i)$ can be stored in a look-up table. For Gaussian distribution, values can be taken from Max [4].

2.5.5 Further Approximations

The values of decrease in distortion per bit can be approximated to analytic functions to avoid the use of a look-up table. Segall presented an algorithm which uses an analytical formula to approximate the benefit of adding one bit [6].

The formula for distortion is written as

$$D(b_i) = \sum_{i=1}^{d} \sigma_i^2 A_i(b_i) \tag{2.21}$$

where σ_i is the standard deviation, σ_i^2 is equal to the variance λ_i and $A_i(b_i)$ depends on the distribution of the dimension to be coded.

Segall develops $A_i(b_i)$ for a Lloyd-Max quantizer used on a Gaussian random variable and $A_i(b_i)$ is written as

$$A_i(b_i) = \begin{cases} 2^{-1.57b_i}, & \textbf{if } b_i \leq 2.32 \\ 2.73\dfrac{2^{b_i}}{(2^{b_i}+0.853)^3}, & \textbf{if } b_i \geq 2.32 \end{cases} \tag{2.22}$$

Huang and Schultheiss [2] made a further assumption in their derivation of the bit allocation formula which assumes the benefit to be 6dB per bit. This assumption corresponds to assuming high-rate quantization as one can see from Segall's formula (2.22) when using a large b_i. When assuming high-rate and thus 6dB per bit, the Greedy bit algorithm reduces to a much simpler form where for every bit, the distortion reduces by a factor 4. Then the algorithm only needs to find the dimension which currently has largest distortion and allocates a bit.

2.6 Level Allocation

One can observe that allocating bits adds another constraint to the optimization: the number of bits allocated to a dimension must be an integer. Assuming Gaussian distribution, for best MSE, the bit allocation must match most closely what is given by the optimal bit allocation formula (2.15) and therefore the ratio between levels allocated to each dimension as seen in the previous derivation, by constraining the number of bits to integers, we may get quite far away from the bit allocation formula (2.8) as it can be unfeasible to match fractional number of bits given by the formula. The magnitude of this problem can be reduced by using level allocations instead, in this way, only levels have to be integer numbers and the optimal bit allocation can be more closely matched. For the bit allocations to be integers, one needs to round down the bits. The extreme example in Table 2.2, where the bit allocation formula 2.15 has been used to assign bits, shows that one cannot round up neither simply round or one might over allocate bits.

In this case, rounding down respects the maximum bit rate but ends up wasting all the bits. Of course, there exists clever algorithms to distribute the remaining bits after the allocation such as the Greedy Algorithm. The example in Table 2.3 illustrates the problem of integer bit allocation that occurs even when using the Greedy Algorithm, showing that the Greedy algorithm does not get close enough to the optimal bit allocation formula.

From this table, one can see that using levels instead of bits will ultimately lose fractions

Table 2.2 Rounding the Bits

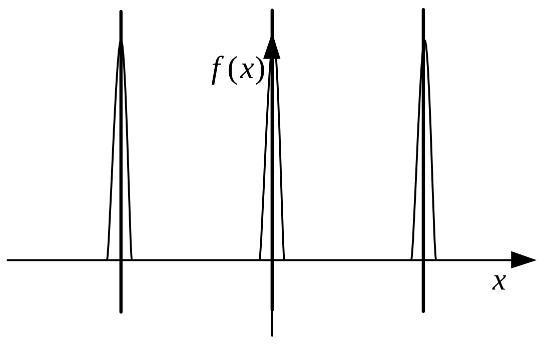
Dim	Bits from formula	Round up	Round down	Round
dim 1	0.75	1	0	1
dim 2	0.75	1	0	1
dim 3	0.75	1	0	1
dim 4	0.75	1	0	1
total bits	3	4	0	4

Table 2.3 Level Allocation Versus Bit Allocation

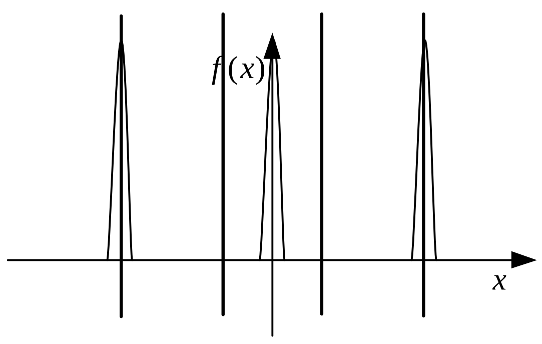
Dim	Bits from formula	Greedy	Levels	Resulting Bits
dim 1	5.75	6	53	5.7279
dim 2	5.75	6	53	5.7279
dim 3	5.75	6	53	5.7279
dim 4	5.75	5	53	5.7279
total bits	23	23	22.9117	22.9117

of a bit. However, with proper algorithms such as the Greedy Level Allocation algorithm presented later, one can get an allocation reaching 22.9926 bits (54,54,54,53 levels) in this case. The match with respect to the bit allocation formula however shows clearly that using levels is a much closer approximation and may give lower total distortion.

There are a few inconveniences associated with using levels instead of bits. A problem is the symmetrical uniform scalar quantizer suboptimality for certain distribution functions, for instance, having 4 levels might give worse distortions than having 3 levels. Considering the simple example in Figure 2.5, one can see that the quantizer with 3 levels is more suited to this distribution than the quantizer with 4 levels and this means that the Distortion-Rate ($D(R)$) curve may not be a strictly decreasing function. The $D(R)$ curve maps a distortion for representative values of bit rate, this way, one can compare two algorithms based on their $D(R)$ curves as it is only fair to compare two algorithms by looking at the resulting distortions at the same bit rate, or seeing how many bits each require to achieve a same distortion. However, when allocating bits, adding a bit simply adds a level between the current levels, with the current levels remaining intact, therefore, the $D(R)$ curve would be strictly decreasing.



(a) Symmetrical Uniform Quantizer With 3 Levels



(b) Symmetrical Uniform Quantizer With 4 Levels

Fig. 2.5 3 Levels Versus 4 Levels Symmetrical Uniform Quantizers

2.7 Compandor

Trying to benefit from both scalar quantizer types advantages, one can build a non-uniform quantizer from a uniform quantizer and a compandor. The block diagram of a typical compandor is shown in Figure 2.6.

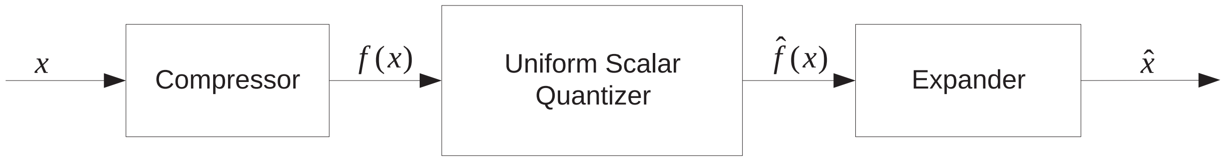


Fig. 2.6 Block Diagram of a Compandor

The idea is to compress and expand the signal so that the uniform quantizer levels would map to the regions of interest such as the region around the mean rather than the tails of the probability density function of a Gaussian distribution function, thus expanding the region where most data points would fall in as opposed to the rest which is a larger interval yet with fewer occurrences. The expander reverses the mapping to get the original scale as shown in Figure 2.6. The optimal high-rate compressor is developed in [9] and is given by

$$C(x) = \frac{\displaystyle\int_{-\infty}^{x} f(y)^{1/3} dy}{\displaystyle\int_{-\infty}^{\infty} f(y)^{1/3} dy} \tag{2.23}$$

where $f(y)$ is the probability density function (PDF) of the variable to be compressed and $C(x)$ is the compressed variable.

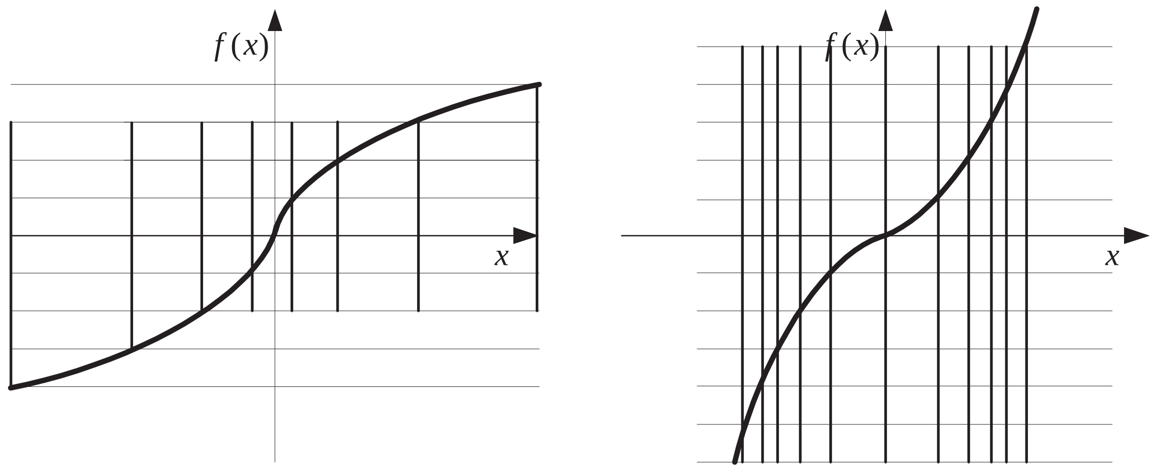
A typical compressor-expander pair is shown in Figure 2.7. Note that the resulting quantizer becomes a non-uniform quantizer as expected as seen on Figure 2.7(a).

2.8 GMM Coder Design

The GMM coder designed by Subramaniam is the basis of all experiments with speech for the course of this work. It is used here to code LSF speech parameters.

The complete coder is shown in Figure 2.8 and for each Gaussian mixture, the branch labeled "mixture" is implemented as seen in Figure 2.9, where μ and σ are the parameters representing respectively the averages and standard deviations of the Gaussian mixture or cluster of the branch and Q is the eigenvector matrix used as the KLT decorrelating the input data.

This coder is simply a transform coder. The signal is being processed by several transform branches simultaneously and the branch giving the smallest distortion is considered



(a) Compressor (b) Expander

Fig. 2.7 Simple Compressor-Expander Pair

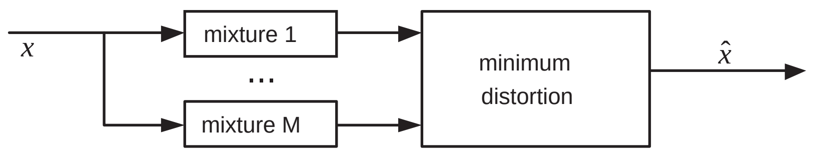


Fig. 2.8 Overall GMM Coder Designed by Subramaniam

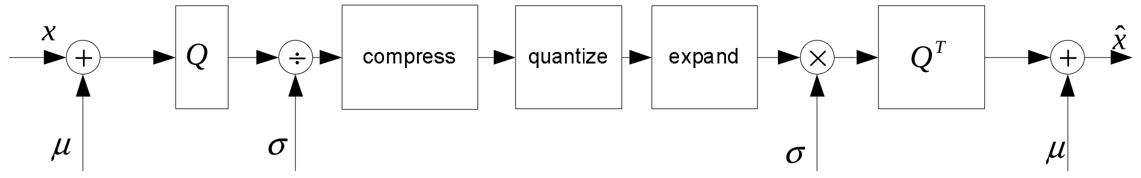


Fig. 2.9 A Branch of the GMM Coder Designed by Subramaniam

the Gaussian source generating the signal. The transformation Q on each branch is the KLT associated to the covariance matrix of the corresponding gaussian mixture.

For the purpose of this thesis, all training and testing was performed on a GMM-based LSF quantizer with 64 mixtures with full covariance matrices. For the GMM-based LSF quantizer, the training set was a set of 1000 speech files and the test set was 200 speech files when the error criterion was the MSE and 50 when the error criterion was the LSD. All speech files were from the McGill TSP lab speech sample database. These speech files were recorded at 8000Hz, from there, 10th order linear predictive (LP) coefficients were computer from frames of 32milliseconds (ms) with a 50% overlap. Finally, 10th order line spectral frequencies (LSF) are derived from the LP coefficients as they are simply a representation of the LP coefficients.

2.9 Image Coder Design

The image coder is a simple transform coder using the Karhunen-Loève Transform (KLT) as the transformation matrix.

The coder is used to compress a 512x512 pixels standard image such as the popular "Lena" and the "gold hill". These pictures have broad frequency content, it has both low, medium and high frequency content and therefore form a good test set.

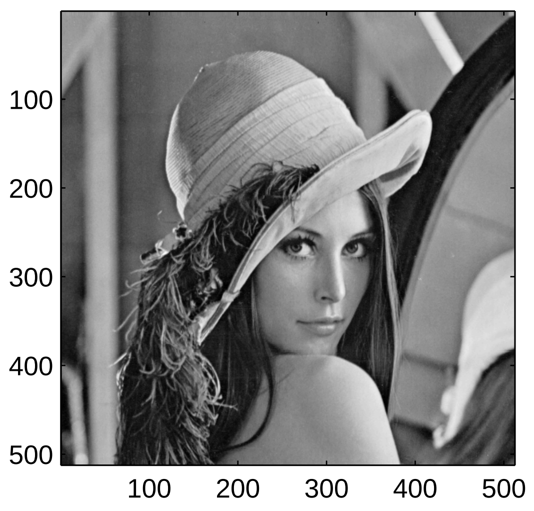
In order to separate the training set from the test set to obtain more fair results, training is performed on one image and testing on another image, as it is not good practice to perform training and testing on the same set. In this case, training is performed on the "gold hill" image and testing is performed on the "Lena" image.

The image is first split into 8x8 pixels blocks. These blocks form the training data set for the KLT to be used to decorrelate the blocks of the image. Each block is rewritten as a 64-dimensional vector. Next, the vectors are used to train the KLT by finding the covariance matrix of the data and getting its eigenvectors matrix which is the KLT. This completes the training for the transformation matrix.

Now each block of the test image is decorrelated using the newly found KLT matrix. Then the transformed coefficients are quantized using previously optimized quantizers for each dimensions for Gaussian data. One such quantizer takes the range as a multiple of the variance and divides the range by the number of levels allocated by the levels allocation algorithms. The implemented quantizer is a scalar uniform quantizer.

Finally, the quantized block is recorrelated and recomposed into the 512x512 image. The distortion on the original image and reconstructed image is computed.

Here are the two images that will be used for training and testing.



(a) Lena

(b) Gold Hill

Fig. 2.10 Images Used for Testing

Chapter 3

A Greedy Level Allocation Algorithm

3.1 Problem of the Greedy Bit Allocation Algorithm

The Greedy bit algorithm allocates integer bits. If one wants to allocate integer number of levels to each dimension and code dimensions as a vector, then one needs to develop a new greedy algorithm. This problem is only made worse because of the fractional bit nature of the coder designed by Subramaniam [1]. In the coder in question, there is first a cluster bit allocation which allocates the $2^{b_{tot}}$ levels to the M Gaussian mixtures according to the variances in the mixture and the prior probability of the mixture component. The cluster bit allocation formula is written as

$$2^{b_j} = 2^{b_{tot}} \frac{(\alpha_j c_j)^{\frac{d}{d+2}}}{\sum_{i=1}^{d} \alpha_i c_i^{\frac{d}{d+2}}} \tag{3.1}$$

where d is the number of dimensions, α_j is the prior probability of a Gaussian mixture j and $c_j = \prod_{i=1}^{d} {\lambda_{j,i}}^{1/d}$ in which $\lambda_{j,i}$ is the variance of dimension i of cluster j.

From (3.1), it is obvious that the number of bits given to each cluster has no guarantee of being an integer, but if one requires integer bits, then the fractional number of bits will have to be truncated, hence the big loss of bits due to integer bit allocation. Then, the fractional number of bits is allocated among the dimensions. For levels allocation, the algorithm works well and there is but small losses due to integer levels,

Moreover, the Greedy algorithm indeed does not have the problem of negative bit allocation and is extremely simple to compute but lacks the precise fractional bits allocation that can further match the bit allocation formula.

3.2 Greedy Level Allocation Algorithm

The proposed algorithm allocates level-by-level to each dimension, trying to reduce the distortion of each dimension and by spending the least amount of bits.

3.3 Criteria To Optimize For

3.3.1 Cost

When allocating a level, there is an associated cost. In the Greedy bit allocation algorithm, it was easy to find the cost: it would be simply 1 bit. When allocating a level, the cost depends on the previous number of levels the dimension already had. If the dimension had 1 level and we increment it to 2, 1 bit is used whereas if the dimension had 10 levels, incrementing it to 11 doesn't cost much, in fact only $\log_2(11/10)$bits.

3.3.2 Benefit

The benefit is defined as the difference in total distortion before and after the allocation

$$benefit = D_{old} - D_{new}. \tag{3.2}$$

Fox used the difference of objective functions as the benefit as well in [7].

3.3.3 Benefit-to-Cost ratio

Assume the total distortion is simply the sum of all individual dimension distortions. The ratio to optimize is written as:

$$\begin{aligned} ratio_{i,j} &= \frac{benefit_{i,j}}{cost_{i,j}} && (3.3) \\ benefit_{i,j} &= D_{\text{tot}}(i, j-1) - D_{\text{tot}}(i, j) && (3.4) \\ cost_{i,j} &= \log_2 \frac{L_{tot}(j)}{L_{\text{tot}}(j-1)} && (3.5) \end{aligned}$$

where $L_{\text{tot}}(j)$ is the product of the levels L_i at allocation j and $D_{\text{tot}}(i,j)$ is the distortion of dimension i with allocation j.

This criteria makes sense because when allocating a level to a certain dimension, we must evaluate how beneficial it is and also the cost of doing so, which was not useful for the Greedy bit allocation algorithm because when allocating one bit, the cost would be one bit. Here allocating one level can be more or less bits depending on the number of levels already allocated to the dimension.

3.3.4 Intuitive Understanding of the GRLA

To further understand the principle through the GRLA, the following graphical example will be used.

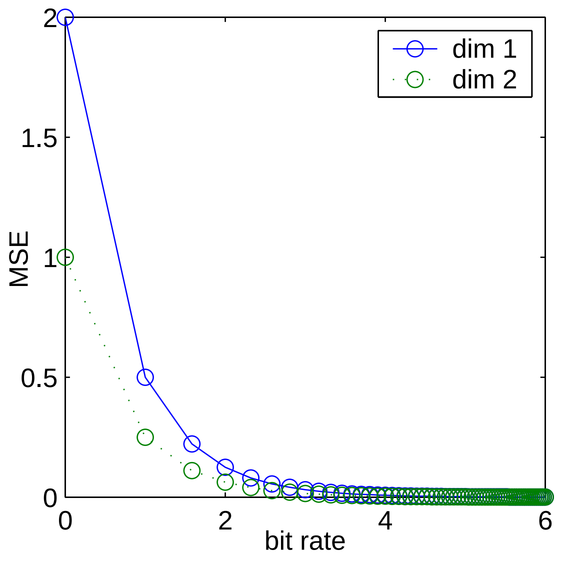


Fig. 3.1 MSE Versus Bit Rate for 2 Components, one with variance equal to 1, and one with variance equal to 2

From Figure 3.1, one can see the theoretical Lloyd-Max quantizer MSE distortion versus the given bit rate. From the graph, one can call Δx_i the cost in bits and Δy_i the benefit in MSE. Then the slope $\frac{\Delta y_i}{\Delta x_i}$ is simply the previously described benefit-to-cost ratio.

From the previous definitions, it is also obvious that a level should be given to the curve with the allocation yielding the largest slope, i.e. benefit-to-cost ratio. The first level would be given to dimension 1. Then, the current slope of dimension 2 becomes steeper than the current slope of dimension 1, thus, a level is allocated to dimension 2. This procedure continues, each time looking at the current slope. This is the intuitive understanding of the GRLA.

So far, the definition of the benefit for the GRLA is quite similar to that of the Greedy bit allocation algorithm. One could again make some approximations by assuming high-rate and Gaussian distribution. Then the distortion can simply be computed analytically as

$$D_{tot} = \sum_{i=1}^{d} K \frac{\sigma_i^2}{L_i^2} \tag{3.6}$$

where one can simply compute $\frac{\sigma_i^2}{L_i^2}$ as a comparison value for each dimension.

3.3.5 Level Allocation Flattening

There are many possible variations of the benefit and cost functions. For instance, one can ignore the cost totally and only use the benefit in the benefit-to-cost ratio. The effect is a flattening of the level allocations, which can be observed in this simple example. Consider the initial variance $\lambda_1 = 1 = \sigma_1^2$ and $\lambda_2 = 1024 = \sigma_2^2$ and we have 5 bits to allocate. The right bit allocation will make $\sigma_1/L1_i = \sigma_2/L2_i$ (from the bit allocation formula derived in section 2.4.2), where $L1$ and $L2$ denote respectively the number of levels allocated to dimensions 1 and 2. Therefore, it is obvious that the level allocation is $L1 = 1$ and $L2 = 32$. However, when not taking the cost into account, the level allocation in Table 3.1 occurs.

Table 3.1 Level Allocation without Cost

$L1_i$	$L2_i$	$\sigma_1/L1_i$	$\sigma_2/L2_i$
1	1	1	32
$\cdots$ 1	13	1	32/13
1	14	1	32/14
2	14	1/2	32/14
2	15	1/2	32/15
2	16	1/2	2

The problem occurs at the last line where the algorithm decides to allocate a level to dimension 1 because $\sigma_1/L1_i$ would be reduced by $1^2 - (1/2)^2 = 0.75$ whereas $\sigma_2/L2_i$ would only have been reduced by $(32/14)^2 - (32/15)^2 = 0.673$. Therefore, only using the benefit to allocate bits would result in adding a bit to dimension 1 and this in turn causes the algorithm to never reach the optimal ratio of 1 : 32 levels equal to the ratio of standard deviations. This flattens the level allocation among the dimensions

In general, flattening the level allocation gives bad performance since one gets farther from the allocation corresponding to the ratio of standard deviations. This effect is in fact easy to observe as a dimension which has never had a level allocated to it will react much more to getting one more level, increasing from 1 to 2 levels, decreasing the distortion by 1/2 whereas a dimension with larger variance would already have 1024 levels and an addition would only be a change of variance of 1/1024. It is true that giving the level to the dimension with fewer levels would decrease the total distortion more, but the opportunity cost is something that is important to take into account as now: by allocating a level to the dimension with a lower number of levels, one prevents the dimension with 1024 levels to go from 1024 to 2048. Therefore, cost is important.

3.3.6 Level Allocation Unflattening

There is yet another variation to the Greedy Level Allocation (GRLA) algorithm, it consists of changing the benefit function to emphasize variance. Emphasizing variance has an obvious effect when one looks at the bit allocation formula derived in Chapter 4 repeated in (3.7), it will unflatten the level allocation.

As in the modified bit allocation formula

$$b_i = \frac{b_{tot}}{d} + k \log_2 \frac{\lambda_i}{\left(\prod_{j=1}^{d} \lambda_j\right)^{\frac{1}{d}}}. \tag{3.7}$$

One can notice that the factor k which will be explained to be a factor correcting the low bit rate benefit of adding a bit to a dimension. It will be shown that $k = 0.5$ is analogous to setting the benefit to 6dB per bit. From Max [4], one can read that the benefit is lower for low bit rates, being actually closer to 5dB per bit. It will be shown in Chapter 4 that 5dB per bit benefit is achieved by setting $k = 0.6$. Rewriting the above in a more explicit way, one gets

$$\begin{aligned} b_i &= \frac{b_{tot}}{d} + 0.5\alpha \log_2 \frac{\lambda_i}{\left(\prod_{j=1}^{d} \lambda_j\right)^{\frac{1}{d}}} \\ &= \frac{b_{tot}}{d} + 0.5 \log_2 \frac{\lambda_i^{\alpha}}{\left(\prod_{j=1}^{d} \lambda_j^{\alpha}\right)^{\frac{1}{d}}} \end{aligned} \tag{3.8}$$

where $\alpha = k/0.5$.

It is observed that $k = 0.6$ gives better results for our bit rates of interest and therefore, the corresponding α is $0.6/0.5 = 1.2$. This means that the variances should be raised to the power 1.2 to yield a benefit of 5dB per bit.

The variances used in the benefit can be raised to different powers. The effect is the same as changing k in the bit allocation formula (3.7). Figure 3.2 shows the MSE performance of different values of k on the GMM-based LSF quantizer with 64 Gaussian mixtures. Although the performance gain is small, it does show that the optimal α is not 1, and therefore, that the optimal k is not 0.5. This result supports the hypothesis that the assumptions that the data to be coded is Gaussian distributed and high-rate are not valid.

The power of the flattening is associated to the actual benefit, i.e. reduction on distortion, of adding one bit. One may even overunflatten by powering the variances by an α which is too big. The result would be the same as with increasing k in the next chapter too much and therefore there is a point where k becomes too big and performance decreases. Therefore, there exists an optimal value and for the the GMM-based coder, $\alpha = 1.2$ or $k = 0.6$.

As seen from Figure 3.2, the optimal α is 1.2, which, as will be seen later, corresponds to having a benefit of 5dB per bit which is around the benefit of adding a bit to a dimension at low rate.

Overunflattening can be used to counter flattening if one wishes to. Some tests show that combining the two effects can yield results which are better than when using the proper benefit and cost. It is however hard to predict the level of flattening caused by ignoring the cost and therefore, most of the time, one would refrain from using this flattening-unflattening pair.

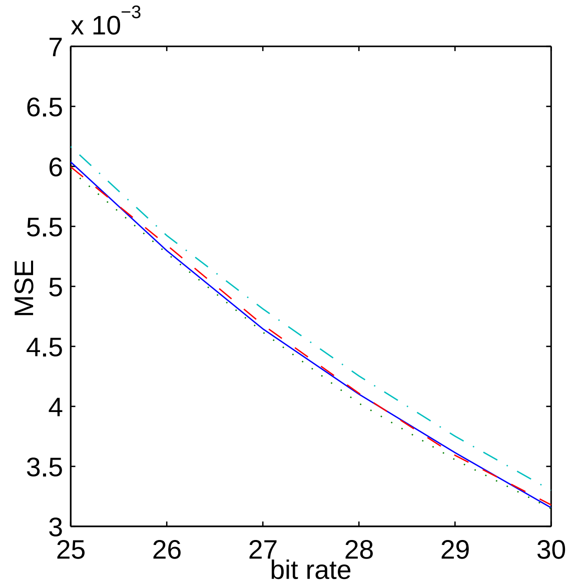


Fig. 3.2 Performance Using Different Value of α, Solid Line: $\alpha = 1$, Dotted Line:$\alpha = 1.2$, Dashed Line:$\alpha = 1.4$, Dash-dotted Line:$\alpha = 1.6$

Table 3.2 Level Allocation with Flattening and Unflattening for 27 Bits

dimension	1	2	3	4	5	6	7	8	9	10
levels for version 1	22	12	10	7	3	4	4	6	5	5
levels for version 2	16	10	9	7	3	4	5	6	6	6
levels for version 3	85	22	17	7	1	2	3	5	4	5
levels for version 4	35	14	13	7	2	3	4	5	5	5

Table 3.2 shows a typical bit allocation pattern for 4 versions of the Greedy Level Allocation algorithm, varying the definition of the ratio. The first version is the original definition of benefit and cost; the second one is when only considering benefit, thus flattening the level allocation; the third one is when squaring the variance in the benefit definition to unflatten the level allocation; and finally, the fourth one is when squaring the variance to unflatten the level allocation but not considering cost, giving thus a somewhat balanced flattening-unflattening pair. One can see that indeed, the second version gives more flat number of bits throughout the dimensions. The third version gives a very unflattened bit allocation, where the dimensions with high variances get nearly all the bits. The first and fourth versions have similar distribution as a result of balanced flattening-unflattening. These indeed confirm the aforementioned effects.

3.4 Greedy Level Allocation Algorithm

Here is an outline of the Greedy Level Allocation Algorithm (GRLA)

Algorithm 3.4.1: Greedy Level Allocation(*bits*, *variances*)

```
comment: Initialize variables
for each $i \in 1$ to $d$
  do $L_i = 1$
repeat
  comment: Initialize variables
  for each $i \in 1$ to $d$
    do $eligible_i = 1$

  comment: Prune dimensions
  for each $i \in 1$ to $d$
    do $L_i = L_i + 1$
  if $\prod_{i=1}^{d} L_i > 2^{bits}$
    then $eligible_i = 0$

  comment: Find dimension with highest benefit-to-cost ratio
  for each $i \in 1$ to $d$
    do
      $tempL_i \leftarrow L_i + 1$
      $cost_i = \log_2 \prod_{j=1}^{d} tempL_j - \log_2 \prod_{j=1}^{d} L_j$
      $benefit_i = (\lambda_i / L_i^2) - (\lambda_i / tempL_i^2)$
      $ratio_i = benefit_i / cost_i$
      $tempL_i \leftarrow tempL_i - 1$
  $j = arg \max_i ratio_i$
  $L_j = L_j + 1;$
until no dimensions can take another level
```

There are a few variants to this algorithm. One variant consists of restricting the number of levels in each dimension to be an odd number. This restriction comes from the observation that for random variables that do not satisfy the log-concavity condition [11], even numbers of levels yield optimal quantizers which are asymmetric that are not feasible under the current compandor-uniform quantizers. In order to ensure a quantization level on the mean of the density function, one needs to restrict to odd numbers of levels. This is easy to do with the GRLA since it is synonymous to increasing the number of levels in steps of 2 levels rather than 1 level at a time.

3.5 Experimental Results

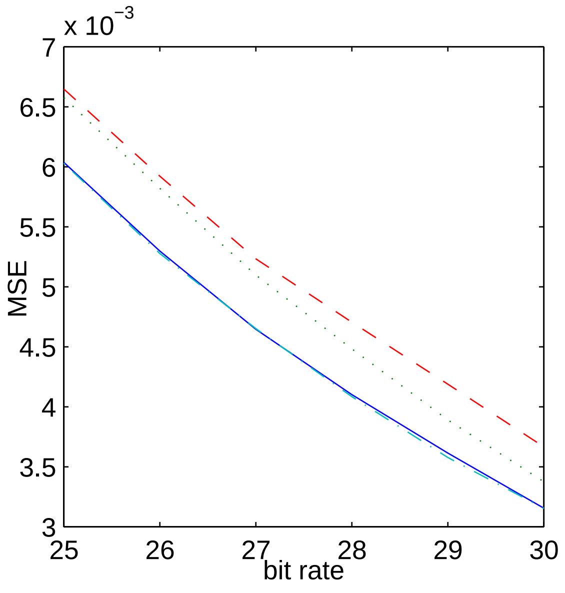
Since the performance of a bit allocation scheme depends a lot on its assumptions, the various bit allocation schemes will be tested on two test cases: the first one is the LSF quantizer described in [1] on a set of speech files from McGill TSP lab where the resulting mixture components approach Gaussian distributions as described in Chapter 2. The GRLA will not be tested on the Lena image as the image coder has both high dimensionality and high bit rate making the algorithm very computationally expensive and impractical. The GRLA will be used as a topping off algorithm to complement the allocation algorithms described in Chapter 4.

3.5.1 Simulations and Results on LSF Quantizer

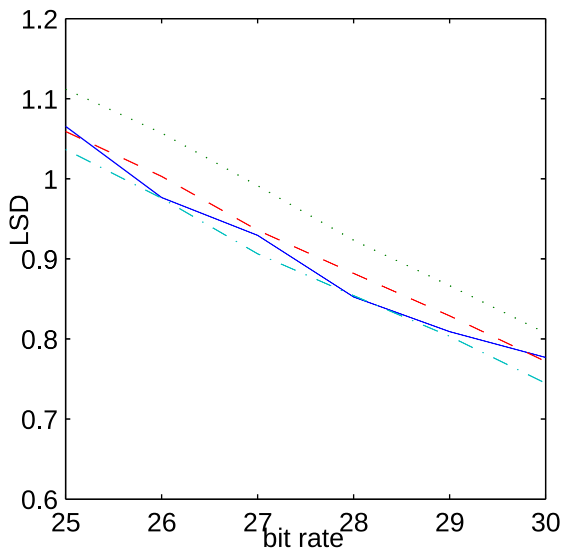
As was described by the theory, looking at Figure 3.3, when ignoring the cost, the distribution is flattened, and yielding in the case of this GMM-based coder a worse performance. Squaring the variance unflattens the levels and in this situation, unflattens too much as the best exponent was found to be $\alpha = 1.2$ in Figure 3.2, therefore, performance is also worse. When combining the flattening and unflattening by not taking the cost into account and squaring the variance, the two effects cancel out and leave a slightly unflattened distribution, which happens to be better suited for this particular situation.

It is observed on Figure 3.4 that the GRLA always performs as well or better than the Greedy bit allocation algorithm. This is only due to the more loose constraint of integer levels rather than integer bits. It is observed that when using the MSE as a distortion measure, the GRLA outperforms the Greedy bit algorithm by around 0.6 bit.

From Figure 3.4, one can compare the GRLA with the Greedy bit algorithm. The GRLA approaches better the optimal ratio of levels between the dimensions. Moreover, it allocates maximally the available levels to all dimensions, so that the used bits get close to the given bit rate as could be seen in Table 4.3. The GRLA can attain the average LSD of 1dB using about 25.8 bits whereas the Greedy Bit algorithm needs around 26.7 bits.



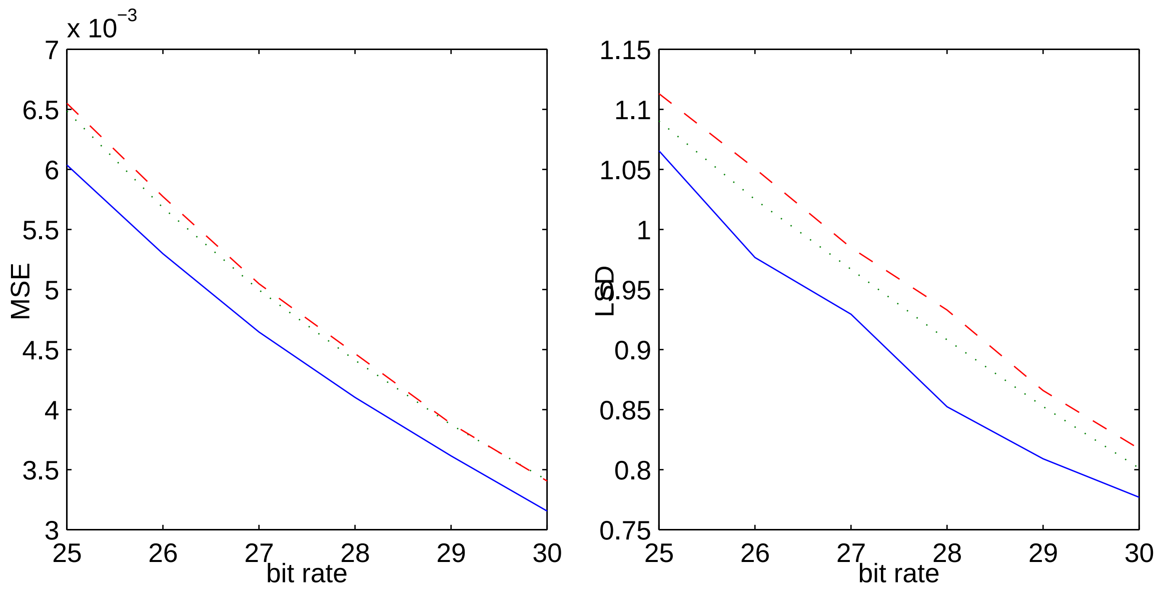
(a) MSE performance



(b) LSD performance

Fig. 3.3 Using the Different Flattening-Unflattening Definitions for Cost and Benefit on GMM-Based Coder; Solid Line: Original Definition, Dotted Line: No Cost, Dashed Line: Variance Squared, Dash-Dotted Line: Variance Squared, No Cost

However, the GRLA has very high computational cost: the Greedy bit algorithm simply allocates bits at a time therefore, the total computation time is proportional to the number of bits and dimensions. The GRLA allocates levels, therefore, the computational time is anywhere between that of the Greedy bit algorithm and $2^{b_{\text{tot}}}$, depending on the variances, e.g. if one dimension has variance dominating the other dimensions, the GRLA will add one level at a time for $2^{b_{tot}}$ times. Therefore, the GRLA has unpredictable execution time. For this reason, the GRLA is only used as a filling algorithm, where the bulk of the work is done through a faster algorithm, such as Segall's algorithm or the algorithms described in Chapter 4.



(a) MSE performance (b) LSD performance

Fig. 3.4 The GRLA Versus the Pruning and the Greedy Bit Algorithm on GMM-Based Coder; Solid Line: GRLA, Dotted Line: Greedy Bit, Dashed Line: Pruning

3.6 Conclusion

The Greedy Level Allocation algorithm yields good performance under the same assumptions as the original bit allocation formula, that is Gaussian distribution, high bit rate, additive distortions and concave distortion functions. This algorithm has the advantage that it can get very close to the required bit rate, even when it is fractional, while keeping a good ratio of between the levels allocated to each dimension. However, for high-rate and high dimensions, the Greedy Level Allocation algorithm becomes very computationally intensive as its computational complexity increases linearly with the number of dimensions and exponentially with the bit rate. Therefore, using the Greedy Level Allocation algorithm for image coding can be very expensive and impractical.

The true usefulness of the Greedy Level Allocation algorithm is its ability to be coupled with an algorithm that does not approach well the required bit rate such as the TSIBA and the Generalized TSIBA described in Chapter 4. In such cases, the TSIBA first gets close to the required bit rate while keeping its ratios and then the GRLA is only used to get even closer to the required bit rate, thus forming a more solid and less wasteful algorithm by topping off those dimensions which can take more levels.

The topping off procedure is generally done on high variance dimensions which already have a lot of levels, therefore that have benefit closer to 6dB. Thus, the α-factor is chosen to be 1 instead of 1.2 when the GRLA is used for topping dimensions.

Chapter 4

Improved Bit Allocation Formula

4.1 Introduction

4.1.1 Traditional Bit Allocation Formula For High-Rate KLT

Huang and Schultheiss [2] derived the bit allocation formula that is now commonly used in transform coders and seen as the optimal bit allocation formula. The derivation made assumptions on high-rate and Gaussianity of the data set. They indeed consider the bit allocation to be at least 3 bits per dimensions [2]. However, Lloyd [3] had previously worked on the effect of PCM quantization and showed that at lower rate, the distortion is not simply a function of the step size but of a variable $A_i(b_i)$, which could vary from 1 to about 2.72[3]. When assuming high-rate, the derivation assumes a constant value of $A_i(b_i) = K\lambda_i$ where K is a constant.

In nowadays speech and image applications, the bit rate is rarely set to be above 3 bits per dimension. For instance, in [1], the bit rate of interest is around 2 bits per dimension. Therefore, the formula needs to be modified to reflect today's low-rate needs.

As seen in section 3.3.6, by changing the exponent applied to the variance, one could change the benefit of adding one bit and get bit allocations yielding lower distortions. The same idea can be applied to the bit allocation formula derived by Huang and Schultheiss and will be done in section 4.3.

4.2 Non-Negative Bit Allocation

As seen in section 2.4.2, it is seen that the ratio of the number of levels given to each dimension must be kept constant for optimality to be kept. Now imagine the case where the total bit rate is reduced by more than a certain limit, thus that some L_i would need to be smaller than 1, and consequently negative bits, in order to preserve the ratio. The algorithm then would allocate negative bits to dimensions and from there arises the problem of negative bit allocations.

There are many ways of dealing with negative bit allocations, however, the important characteristic of any algorithm is to preserve the λ_i/L_i^2 ratio, if not for all dimensions, then at least for dimensions with a large enough variance so that they should get bits. Dimensions with small variances are then suitable for an average mean square distortion around the value of their variances, which is still smaller than the distortion of large variance dimensions.

Segall [6] described an algorithm which solves the problem of negative bit allocation. The algorithm sets a threshold θ of variance σ_i^2 or λ_i below which a dimension is pruned.

The pruning is done through the following step (see Appendix E for more details):

$$S(\theta) = \sum_{j:\sigma_j^2 \geq \theta} h\left(\frac{\theta}{\sigma_j^2} k'(0)\right) \tag{4.1}$$

where h is the inverse function of k', where k' is the derivative of $k(b_i)$, and $k(b_i)$ is the decrease of distortion per bit and $S(\theta)$ is the total number of bits used, which is not to exceed the given number of bits. θ can be seen as the threshold of variance below which no bits are allocated to the dimension.

The two-stage iterative bit allocation (TSIBA) algorithm also takes care of the negative bit allocation problem by a similar philosophy, yet another algorithm.

The TSIBA, as its name states, is an iterative procedure. The first stage deals with pruning the unwanted dimensions and the second stage adds fractions of bits to each dimension in order to get as close to the given number of bits when the number of levels per dimension is restricted to be an integer.

To prune the dimensions with small variances, one simply distributes bits according to a standard formula, e.g. (2.15), and looks at the bit allocations. If a dimension is allocated a negative number of bits, it is pruned. Then the allocation is performed again on the remaining dimensions until no dimensions get negative bits allocated to them. The bit rate is then rounded so that the number of levels per dimension is an integer.

Most likely, the number of used bits is smaller than the given number of bits at this point. Then, the average bit rate is increased by a slight amount and the allocation done again.

These two iterative loops are done until the number of used bits exceeds the given number of bits.

The TSIBA is not only about solving the problem of negative bit allocations, it also deals with the problem of non-Gaussian and low bit rates as will be seen shortly.

4.3 A Study on Low Rate Effect of Level Allocation

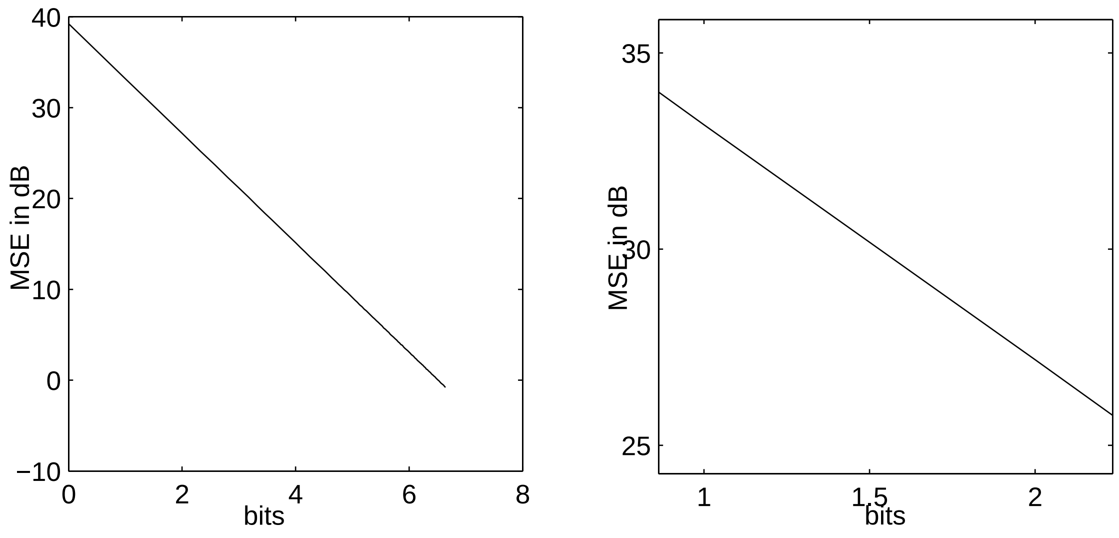
As seen in Chapter 2, the variable $A_i(b_i)$ is assumed to be $K\lambda_i$ with K being constant, which will be shown to not be the case. This indeed yields the well-known bit allocation formula. This assumption is in fact synonymous with the assumption that for a given number of incremental bits, the distortion of the dimension is decreased by 6dB.

The 6dB per bit rule is true for a scalar quantizer for uniform distribution which is the uniform scalar quantizer. It is easy to derive the MSE for the uniform optimal quantizer as the MSE is the following formula

$$\sigma_{e_b}^2 = \frac{2^{-2b}}{12} \tag{4.2}$$

where b is the number of bits in the uniform scalar quantizer and $\sigma_{e_b}^2$ is the variance of the error signal, i.e. the quantization noise power.

From this definition of the MSE distortion, one can derive that for every additional bit allocated to the quantizer, the error is reduced by 6dB, i.e. the variance of the error is divided by 4 as seen in the (4.3) and clearly seen from Figure 4.1.



(a) 6dB per bit (b) zoomed-in

Fig. 4.1 MSE of Uniform Quantizer on a Uniform Data Set Using a Lloyd-Max Quantizer

$$\begin{aligned}\sigma^2_{e_{b+1}} &= \frac{2^{-2(b+1)}}{12}\\ \sigma^2_{e_{b+1}} &= \frac{2^{-2b}2^{-2}}{12}\\ \sigma^2_{e_{b+1}} &= \frac{\sigma^2_{e_b}}{2^2}.\end{aligned} \tag{4.3}$$

Taking $10\log_{10}$ on each side, one gets $\approx$ 6dB.

The same can be done for a Gaussian distributed variable using a Lloyd-Max optimal quantizer. The distortion is written as

$$D_i = A_i(b_i)2^{-2b_i} \tag{4.4}$$

which is once again the 6dB per bit rule that can be derived as in (4.3).

The bit allocation formula is in fact also closely related to the 6dB per bit assumption.

Looking at the optimal bit allocation formula

$$b_i = \frac{b_{tot}}{d} + 0.5 \log_2 \frac{\lambda_i}{\left(\prod_{j=1}^{d} \lambda_j\right)^{\frac{1}{d}}}. \tag{4.5}$$

If one multiplies λ_i by 4, i.e. 6dB, the following expression is given

$$\begin{aligned} b'_i &= \frac{b_{tot}}{d} + 0.5 \log_2 \frac{4\lambda_i}{\left(\prod_{j=1}^{d} \lambda_j\right)^{\frac{1}{d}}} && (4.6) \\ b'_i &= \frac{b_{tot}}{d} + 0.5 \log_2 \frac{\lambda_i}{\left(\prod_{j=1}^{d} \lambda_j\right)^{\frac{1}{d}}} + 0.5 \log_2 4 && (4.7) \\ b'_i &= \frac{b_{tot}}{d} + 0.5 \log_2 \frac{4\lambda_i}{\left(\prod_{j=1}^{d} \lambda_j\right)^{\frac{1}{d}}} + 1. && (4.8) \end{aligned}$$

Therefore, one gets once again the well-known 6dB per bit rule.

However, since most coding is done at low rate, the Gaussian does not reach the 6dB/bit yet as seen in Max [4], but more around 5dB/bit. Considering changing the 0.5 factor in front of the $\log_2$ term to 0.6

$$\begin{aligned} b'_i &= \frac{b_{tot}}{d} + 0.6 \log_2 \frac{4\lambda_i}{\left(\prod_{j=1}^{d} \lambda_j\right)^{\frac{1}{d}}} && (4.9) \\ &= \frac{b_{tot}}{d} + 0.5 \log_2 \frac{\lambda_i}{\left(\prod_{j=1}^{d} \lambda_j\right)^{\frac{1}{d}}} + 1.2 && (4.10) \end{aligned}$$

which means that for a 6dB difference, one would need 1.2 bits, or 5 dB/bit.

Since the behaviour of the Gaussian yields less than 6dB per bit at low rate, then matching the actual benefit per bit may yield a better bit allocation. One can do so by varying the factor in front of the logarithm term of the bit allocation formula.

4.4 The TSIBA Algorithm

The TSIBA deals with several problems encountered by the traditional bit allocation formula. First, there is no guarantee on getting non-negative bits for all dimensions. The logarithmic term can, in fact, yield many negative number of bits depending on how some variances are small compared to others. Second, there is no guarantee that the number of bits, or at least levels allocated to each dimension is an integer. This problem is easy to deal with however, it will cause the total bits allocated to be smaller than the desired bit rate. Third, the bit allocation formula assumes high rate. This assumption, as seen in the derivation of the bit allocation formula, permits a simplification and yield the formula (2.15). However, in nowadays applications, it is often the case that the high-rate assumption is violated. In the LSF quantizer described by Subramaniam [1], we indeed see that the target bit rate is around 2.2 bits per dimensions. The high-rate assumption is further violated in image coding, where each pixel is now coded with less than 1 bit.

The third part of the TSIBA is in modifying the bit allocation formula (2.8). In fact, the bit allocation formula is modified into a similar form:

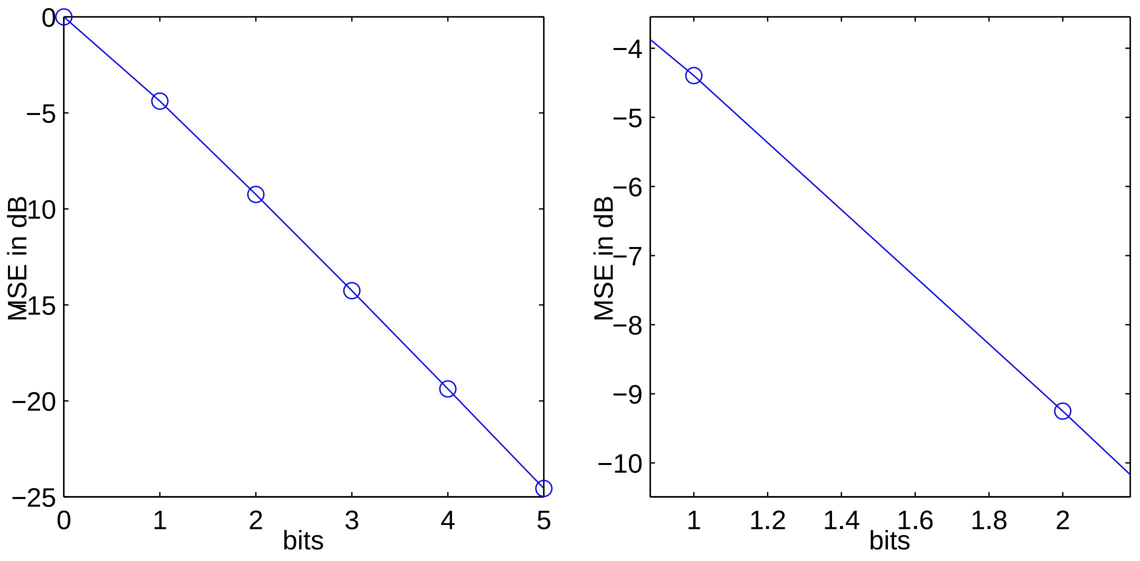
$$b_i = \bar{b} + k \log_2 \frac{\lambda_i}{(\prod_{j=1}^{d} \lambda_j)^{(1/d)}} \tag{4.11}$$

where $\bar{b}$ is the average bit rate per dimension, λ_i is the variance of dimension i, d is the number of dimensions and k is a parameter which is used to tune the benefit per bit.

This equation adds a free variable to the bit allocation formula. This variable k indeed controls the importance of the variance term. By making k larger, one effectively weights the variance more than the average bits per dimension.

When relating back to the previous subsection, one can also view this new variable k as placing a line of slope 5dB/bit on the Gaussian benefit versus bit rate curve. This however changes k globally for all dimensions at the same time, therefore, a dimension which has many bits and that would normally follow the 6dB per bit rule would still be following a

5dB per bit if it is the optimal k. Figure 4.2 and 4.3 show that for a Gaussian distribution, even the Lloyd-Max quantizer does not achieve a performance gain of 6dB per bit but more in the 5dB/bit region.



(a) 6dB per Bit (b) zoomed-in

Fig. 4.2 MSE Calculated by Max for Lloyd-Max Quantizer on Gaussian Data

Here is a brief outline of the algorithm:

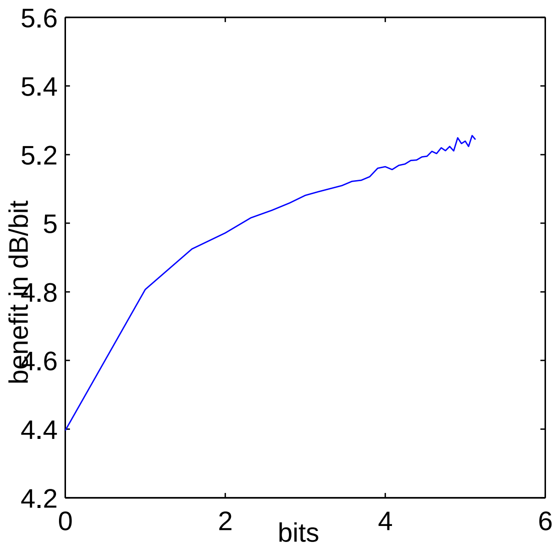


Fig. 4.3 Benefit per Bit for Lloyd-Max Quantizer on Gaussian Distribution

Algorithm 4.4.1: TSIBA-INNER LOOP($bits_{given}, variances$)

```
$\bar{b} = b_{tot}/d$
repeat
    Calculate the bit allocations using a bit allocation formula
    for each $i \in 1 \to d$
    do
        if $b_i < 0$
            $\sigma_i = 1.$
            Tag the i-th dimension as pruned.
        d = number of untagged dimensions.
        $\bar{b} = b_{tot}/d$
        Update the geometric mean of coefficient variances.
until $b_i > 0, \forall i \in 1 \to d$

repeat
    Increment desired bit rate by a step.
    Calculate the $b_i$'s using (2.16) only for untagged coefficients.
    $L_i = \lfloor (2^{b_i}) \rfloor$
    $bits_{used} = \log_2 \prod_{i=1}^{d} L_i$
until $bits_{used} > bits_{given}$
backtrack 1 step
$L_i = \lfloor (2^{b_i}) \rfloor$
```

There are three parts to the Two-Stage Iterative Bit Allocation Algorithm (TSIBA), one which looks for an optimal k, another that prunes insignificant levels before allocating bits, and the last part, that allocates the maximum integer number of levels without going over the desired bit rate.

The outer loop which finds the optimal k simply iterates through all the values of k that have to be considered (depending on the precision we want on the optimal k), using each value of k in the bit allocation formula (2.16).

The procedure that takes the most computational effort is finding the optimal k because the optimal k may lie anywhere, typically from 0.5 to 1, and that choosing k is a post-processing decision, the algorithm simply iterates through the values of k and picks the one resulting in smallest distortion. The computational complexity would be the time taken to code all test data times the number of values of k that one desires to consider. Thus, the computational complexity depends mostly on the precision to which one wants to the best value of k.

The Armijo [8] algorithm can be used to make the search faster. The Armijo increases the value of k geometrically until there is no longer improvement, the value of k is then decreased geometrically again to fine tune k. This method indeed can get to a more precise solution much faster than using regular step sizes. The only drawback of this method of finding k is that it can easily get trapped in a local optimum. For smooth convex functions, it is not a problem. Figure 4.4 shows the curve of the *PSNR* versus the value of k, it is observed that the resulting curve is not smooth. Any disturbance in the k curve would cause the Armijo algorithm to get stuck in a local optimum.

There are many possible fixes for this problem, such as changing search direction only when one is sure the function is changing direction, i.e. the function has steadily been decreasing for a few values of k.

4.4.1 Finding the Low Rate Bit Allocation More Optimal Formula

The bit allocation formula can be modified to yield more optimal bit allocations for low rate.

Note that (2.16) should theoretically yield a better allocation for low bit rates for non-uniformly distributed data.

Let us start by treating the Gaussian case. Max [4] had previously shown an optimal

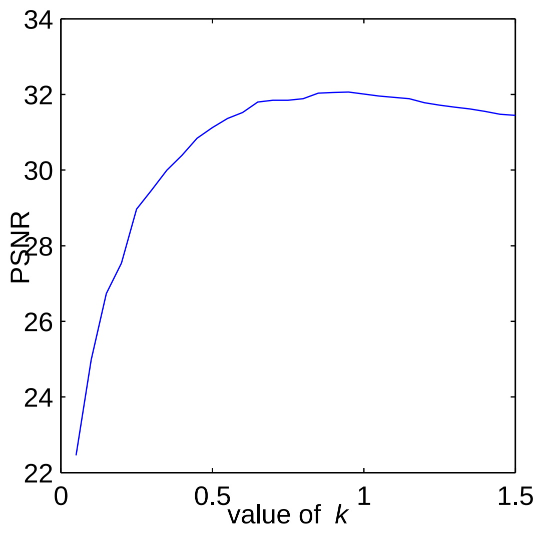


Fig. 4.4 PSNR for Different k for Lena Image

vector quantizer for a normally distributed random variable. His results gave the following table:

Table 4.1 Gaussian: Optimum MSE of Quantization Noise for Various Values of L_i

L_i	MSE	$L_i^2 MSE$
2	0.36	1.45
4	0.12	1.88
8	3.46×10^{-2}	2.21
16	9.50×10^{-3}	2.43
...		
∞	0	2.72

Although these numbers were computed for a vector quantizer (unequal cell-size), these numbers should be lower than when using a scalar quantizer, since MSE would be larger than for a vector quantizer.

These numbers have been recomputed for a scalar quantizer. There are two cases of interest: uniform distribution and Gaussian distribution.

Most distribution can be modelled by a mixture of Gaussian variables, therefore, computing the value of $K_i(b_i)$ for Gaussian distribution will prove fruitful.

One should use the two yielded $K_i(b_i)$ according to the expected distribution of the test set.

From the values of $K_i(b_i)$ trained as shown in the next section, it is simply a question of using it in the bit allocation procedure yielded by (2.16). The final bit allocation is simply found by computing b_i, updating $K_i(b_i)$, recomputing b_i and so on until b_i does not change much anymore.

This method can easily be combined with any other method using the original bit allocation formula and yield a better performance. Since this method is a generalization of the TSIBA, it will so be called and can be used in the original TSIBA to replace the bit allocation formula.

4.5 Training the Generalized TSIBA

The Generalized TSIBA requires a set of $K_i(b_i)$. This parameter can be approximated to the set of $K_i(b_i)$ associated with the Gaussian distribution, however, the performance yielded by such an approximation does not reach the performance of the simple TSIBA. The parameter $K_i(b_i)$ is in fact distribution dependent, and bit dependent, just like the $K_i(b_i)$ found by Lloyd [3], but it will be trained for each dimension in order to match the distribution of the dimension rather than assuming Gaussianity. In fact, for the case of the decorrelated Lena image, it is observed as shown in Figure 4.5 that the dimension with largest variance has distribution that is far from being Gaussian, hence the huge gain that the TSIBA yields. The same goes for the decorrelated LSF coefficients that are to be coded in Subramaniam's coder [1], the effect however being less dramatic because of the usage of the GMM.

For the transform coder used for images, the training of $K_i(b_i)$ is not complicated. The image is decorrelated as usual and the resulting data reorganized into a matrix. Then, for each dimension, a $D(R)$ curve is computed. From the $D(R)$ curve, one gets the $K_i(b_i)$ by using the equation

$$K_i(b_i) = \frac{D(R) \cdot L_i^2}{\lambda_i}. \tag{4.12}$$

For the Subramaniam coder [1], there is an additional problem: one needs to first classify the vectors into the proper cluster before computing the $D(R)$ of each dimension,

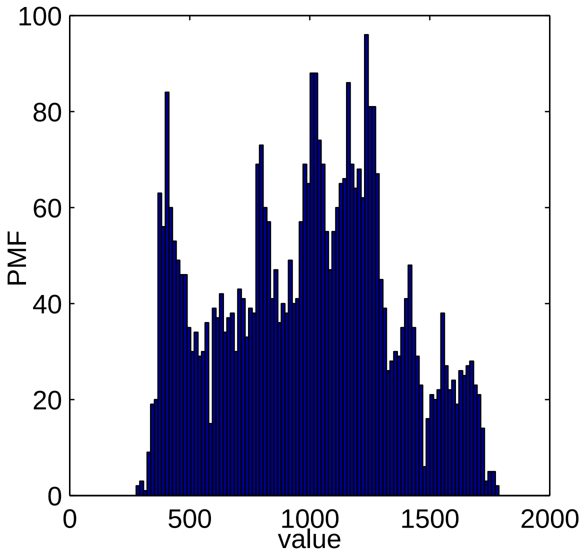


Fig. 4.5 Distribution of data for the dimension associated with the largest variance for the decorrelated Lena image

this is the purpose of having a GMM in the first place. Note that the training of $K_i(b_i)$ can and should be done offline since it may require a lot of time. The training data set is sent through the coder described by Subramaniam [1] with a certain bit rate, where vectors are classified with their respective clusters. Then, in each cluster is computed the $D(R)$ curves for each decorrelated dimension using the *MSE* as the distortion measure. Note that the *LSD* cannot be used since the distortion is computed on the resulting frequency spectrum from the LSF, and therefore the simple *MSE* is used. Each dimension for each cluster get a $D(R)$ curve and therefore a $K_i(b_i)$ curve.

The computed $K_i(b_i)$ can now be used in the bit allocation formula (2.16).

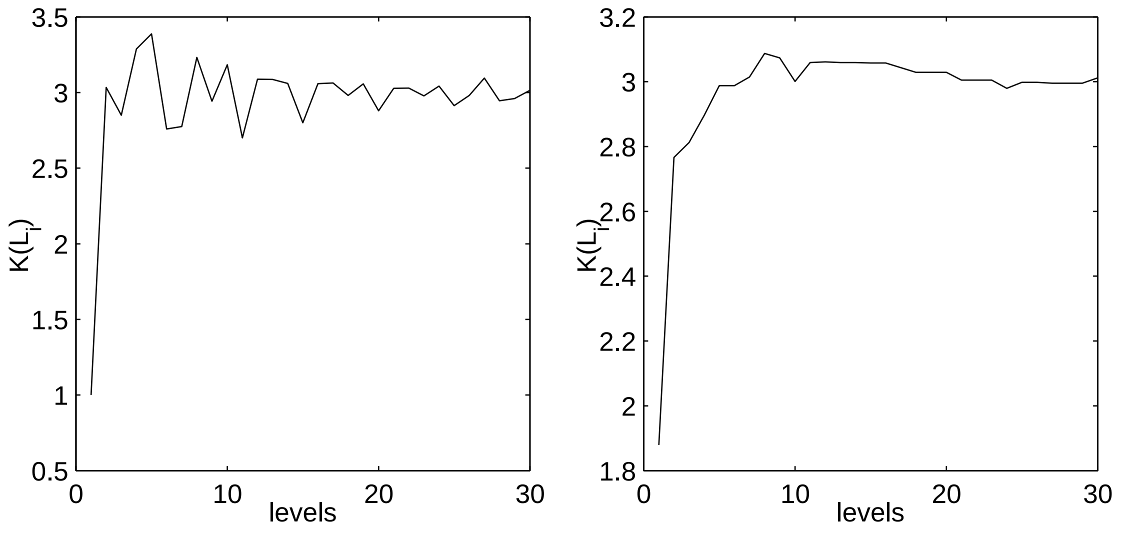
The $K_i(b_i)$ curves do not show a very smooth relationship as seen in Figure 4.6. The $K_i(b_i)$ curves need to be smoothen in order to give a reasonable convergence. To see this problem, let us imagine this hypothetical situation, two variances are fairly close together, one gets an initial number of levels of 5 allocated to it, the other gets 4, but the value of $K_i(b_i)$ rather than being smooth goes down at that special point, say 4 and 5 as shown in Table 4.2. The next iteration would yield a level allocation of 4 and 5. The level allocation would follow this infinite loop.

As one can see from this simple example of the possible infinite loop, a condition for success of the Generalized TSIBA is smoothness of the $K_i(b_i)$ curves. This can be done

Table 4.2 Level Allocation Infinite Loop

	$L(t=i)$	$K(t=i)$	$L(t=i)$	$K(t=i)$	$L(t=i)$	$K(t=i)$	$\cdots$
dimension 1	4	5	5	4	4	5	$\cdots$
dimension 2	5	4	4	5	5	4	$\cdots$

using several methods as long as the curves become smooth and the general form is kept intact. The median filter or a low-pass filter is used to smoothen the curves. The resulting curves look like Figure 4.6.



(a) Unsmoothened $K_i(b_i)$ Curve (b) Smoothened $K_i(b_i)$ Curve

Fig. 4.6 Smoothening the $K_i(b_i)$ Curve

4.6 Simulations and Results

Our interest here is speech coding and image coding. Therefore, simulations will compare the Greedy bit allocation algorithm, the pruning algorithm used by Subramaniam [1] and the TSIBA on a speech LSF vector database and a few standard images.

4.6.1 TSIBA on LSF quantizer

MSE as a Distortion Measure

Since these bit allocation formulas were derived for minimizing the MSE, it makes sense to start by testing the TSIBA using the MSE. The test bench is the Subramaniam speech coder [1] described in chapter 2, using 4 different bit allocation schemes: the TSIBA, the pruning algorithm used by Subramaniam, the algorithm designed by Segall and the Greedy bit allocation algorithm. Note that the pruning algorithm used by Subramaniam does indeed use the original bit allocation formula and thus, it can be optimized for best k search, yielding a better performance.

Figure 4.9a) clearly shows that the Greedy bit allocation algorithm yields a performance which is much poorer than the three other methods, with the TSIBA performing best.

LSD as a Distortion Measure

The second part of the tests are the same as the first ones but the LSD is used as a distortion measure, which is more meaningful for measuring performance of an LSF quantizer.

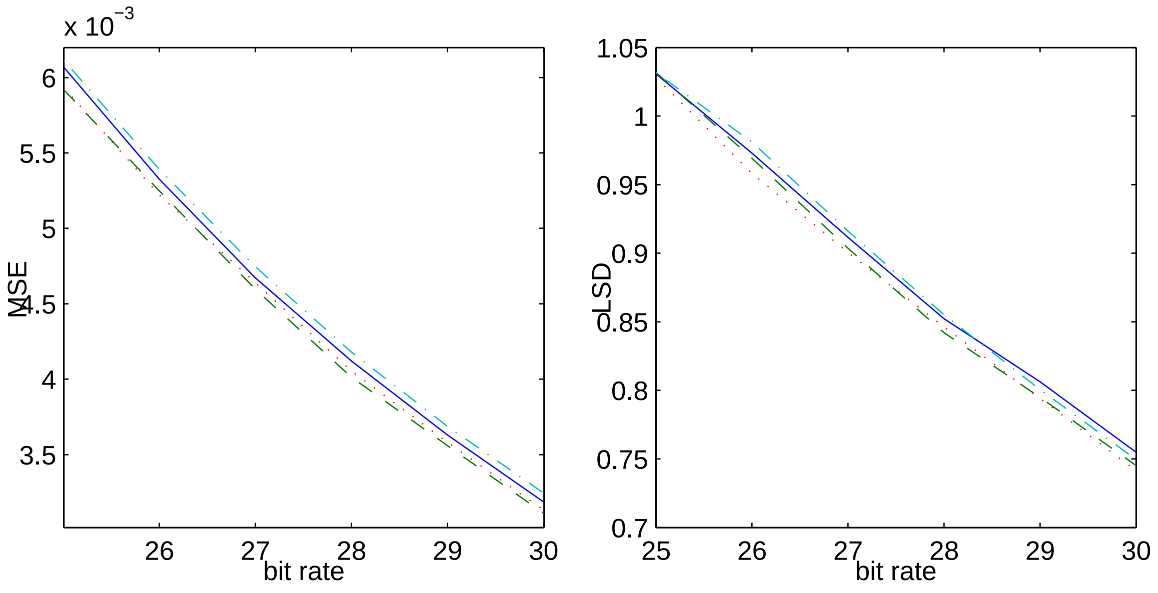
When tested on 64 Gaussians, it is shown in Figure 4.9b) that the TSIBA consistently yields the best performance out of the 4 algorithms (Greedy bit allocation, Segall,TSIBA and pruning), that with or without best k search.

Figure 4.7 shows the $D(R)$ curves for different values of k. It is seen from the plot that the optimal k is not 0.5 as seen in the bit allocation formula derived by Huang and Schultheiss, it is instead 0.6, which is the same as with the GRLA varying α in section 3.3.6.

Figure 4.8 shows the $D(R)$ curve for different values of k in the bit allocation formula used by the pruning algorithm. The optimal value of k is once again 0.6.

On all tests, the TSIBA and the GTSIBA do best, followed by the pruning Algorithm Subramaniam used, the Segall's algorithm, and the worst performance is the Greedy bit allocation algorithm. This may be only due to the Greedy bit allocation algorithm allocating bits rather than levels, which constrains the bit allocation furthermore and thus yields the worst performance.

In fact, the only advantage the Greedy bit algorithm has is that the exact desired bit rate is used rather than a number close to it. This only matters when the approximation yields a large error, i.e. at very low bit rates, but even at those rates, the D(R) curves show that the difference is very thin. And in those cases of very low bit rates, the GRLA



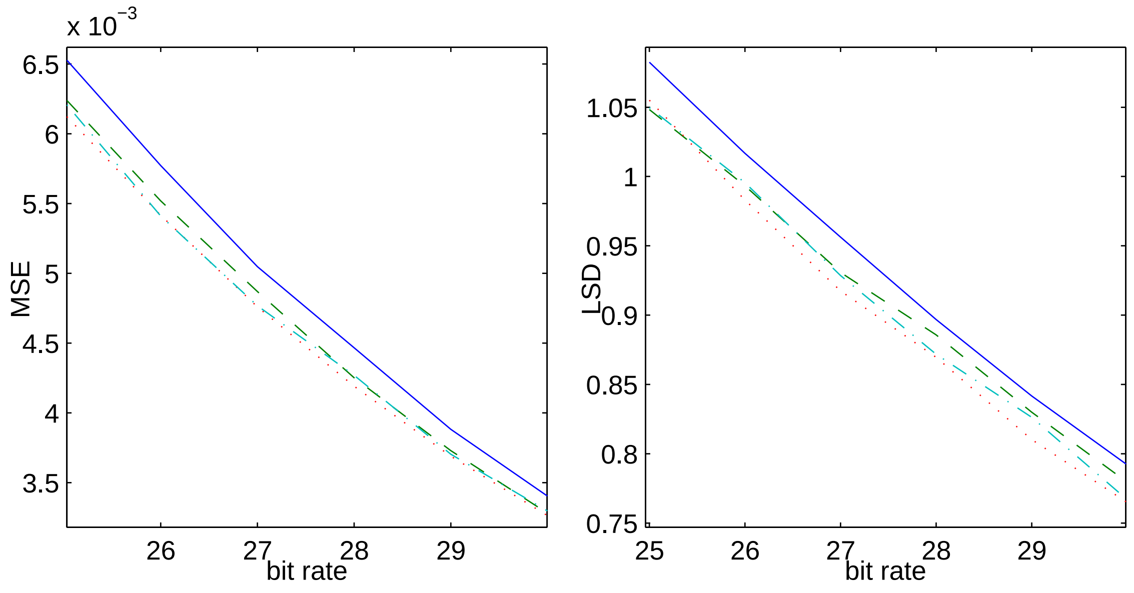
(a) MSE Performance (b) LSD Performance

Fig. 4.7 Using the TSIBA with Different Values of k for the GMM LSF Quantizer; Solid Line: $k = 0.5$, Dotted Line: $k = 0.6$, Dashed Line: $k = 0.7$, Dash-Dotted Line: $k = 0.8$

performs slightly better than the Greedy bit allocation algorithm (see chapter 3 for the GRLA). The GRLA yields performance very close to that of the TSIBA.

Another point to observe is how close the algorithm gets to the required bit rate. This measures how wasteful the algorithm is. The TSIBA by itself is among the more wasteful algorithms.

Referring to Table 4.3, when compared to the pruning algorithm used by Subramaniam [1], we see clearly that the TSIBA wastes about 0.1 bit more. This shows that the algorithm can get an even better performance if it gets closer to the required bit rate. When topping off the bit allocations of the TSIBA using the GRLA, both algorithms use a number of bits close to the required bit rate both when using one mixture and when using 64 mixtures. This is in contrast with the Greedy bit allocation algorithm which used all bits when using one mixture, but wasted a lot more bits when using 64 mixtures, as the number of bits allocated to each cluster (or mixture) was a fractional number. When doing so, the TSIBA reaches very reasonable bit usage. Since the Greedy Level Allocation algorithm is only adding levels to each dimension, the average distortion of the coder results in a lower



(a) MSE Performance (b) LSD Performance

Fig. 4.8 Using the Pruning Algorithm with Different Values of k for the GMM LSF Quantizer; Solid Line: $k = 0.5$, Dotted Line: $k = 0.6$, Dashed Line: $k = 0.7$, Dash-Dotted Line: $k = 0.8$

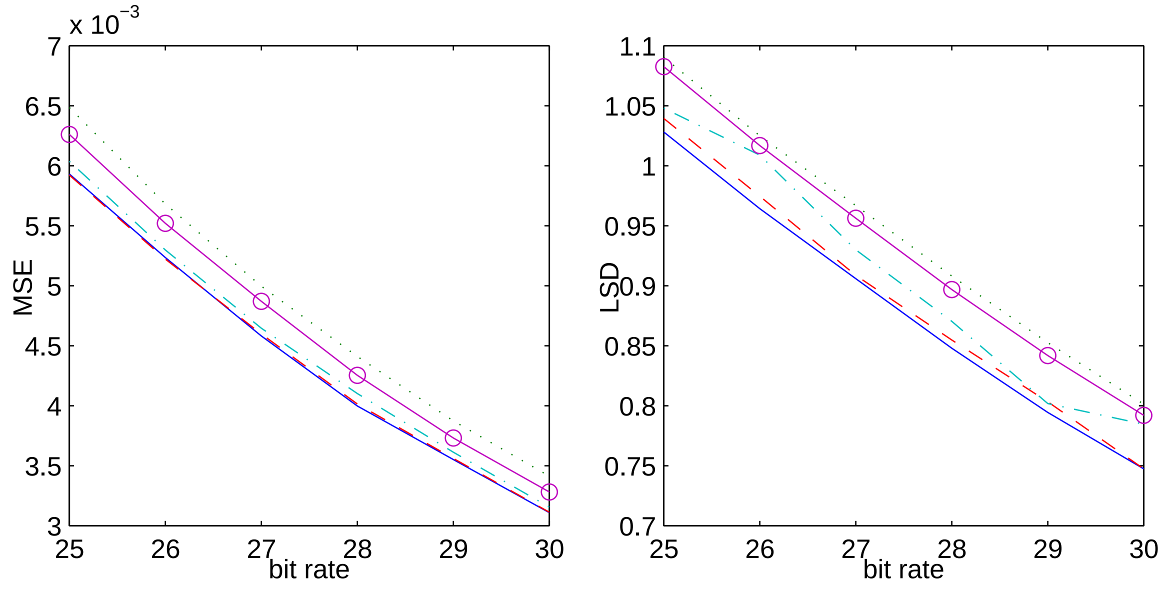
Table 4.3 Table of Used Bits for Pruning and TSIBA with $k = 0.5$

Required bit rate	Pruning	TSIBA	TSIBA + Greedy Level
1 Gaussian Mixture			
25	24.9350	24.8355	24.9875
27	26.9730	26.8355	26.9509
29	28.9501	28.8090	28.9769
64 Gaussian Mixtures			
25	24.9297	24.8189	24.9588
27	26.9378	26.8582	26.9525
29	28.9479	28.8645	28.9663

distortion.

4.6.2 GTSIBA on LSF quantizer

The GTSIBA finds the weighting functions $K_i(b_i)$ before using them to compress the data. It will give similar results to the TSIBA algorithm as they both treat the problem of different benefit per bit at low rate.



(a) MSE Performance (b) LSD Performance

Fig. 4.9 Using the Different Algorithms for the GMM LSF Quantizer; Solid Line: GTSIBA, Dotted Line: Greedy Bit, Dashed Line: TSIBA Optimal k, Dash-Dotted Line: Segall's Algorithm, Solid with Circle: Pruning Optimal k

Table 4.4 shows the compiled results from Figure 4.9.

A good performance gain is also observed on the coder designed by Subramaniam [1] in Figure 4.4. The Generalized TSIBA algorithm clearly outperforms all the other algorithms. The GTSIBA uses 1 bit less than the Greedy algorithm, 0.8 bit less than the pruning algorithm used by Subramaniam and 0.7 bit less than the Segall algorithm. The additional advantage of using the Generalized TSIBA is its ability to compute the $K_i(b_i)$ curves offline at training time. Therefore, provided that the $K_i(b_i)$ curves have been properly smoothened, the algorithm will find the optimal level allocations within a reasonable though

Table 4.4 Table of Bit Rates for an LSD of 1dB

Bit allocation algorithm	bits for average LSD 1dB
GTSIBA	25.5
TSIBA	25.7
Pruning	26.3
Segall	26.2
Greedy	26.5

unpredictable time period as this depends on the smoothness of the $K_i(b_i)$ curves and the precision to which one wishes to get the allocations close to the formula.

Computationally, the Generalized TSIBA usually performs in the same amount of time as the TSIBA, however, as dimensionality increases, it is harder to iteratively refine the levels and thus, it will require more time to converge to an acceptable solution. However, setting the wanted precision to reasonable bounds will prevent the Generalized TSIBA to take too much time to estimate the proper level allocations. The Generalized TSIBA is more suited for the Subramaniam LSF quantizer [1] than the image coder in the sense that it deals with lower bit rate and dimensionality.

Table 4.5 Table of Used Bits for Pruning and Generalized TSIBA

Required bit rate	Pruning	Generalized TSIBA	Generalized TSIBA + Greedy Level
64 Gaussian Mixtures			
25	24.9297	24.8189	24.9588
27	26.9378	26.8582	26.9525
29	28.9479	28.8645	28.9663

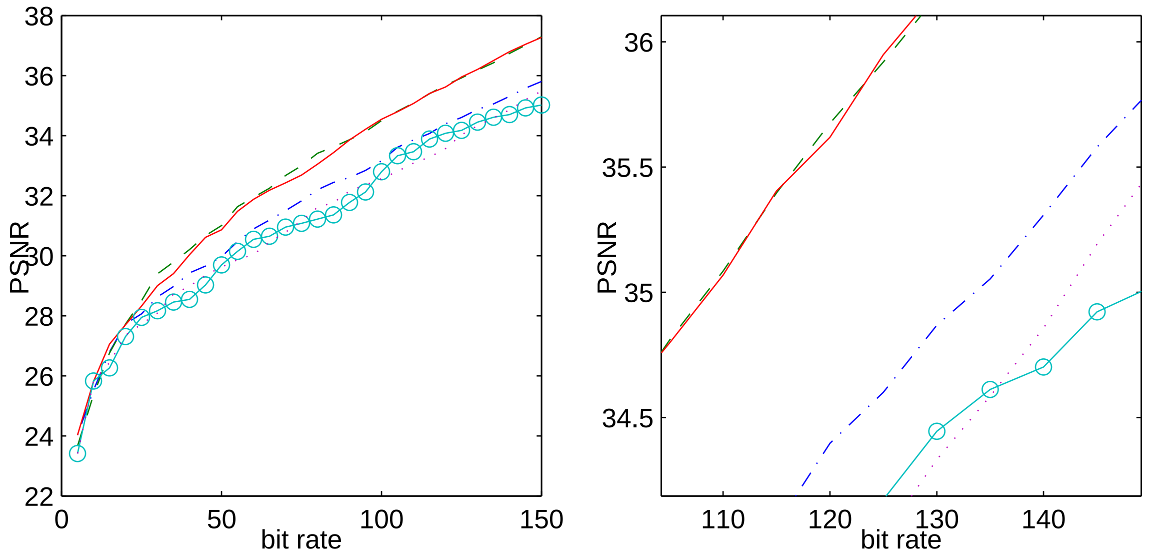
Looking at Table 4.5, it is observed that the same problem as the TSIBA is encountered here, that is the wasting of bits. And therefore, a version of the GRLA algorithm is again used to get closer to the required bit rate.

4.7 Simulations and Results on Image Coder

It makes sense that the TSIBA and Generalized TSIBA will yield a better performances when tested on an image coder as the data is much less Gaussian, at least when using the *MSE* or *PSNR* as the distortion measure.

Test on Lena image

The trainings of the GTSIBA and the KLT matrix were performed on the "gold hill" image and the tests were conducted on the Lena image in order to separate the training set from the test set.



(a) Test on Lena Image (b) Zoom-In Region of Interest

Fig. 4.10 Test of KLT Image Coder on Lena Image; Solid Line: GTSIBA, Dotted Line: Segall's Method, Dashed Line: TSIBA with $k = 0.9$, Dash-Dotted Line: TSIBA with $k = 0.5$, Solid with Circles: Greedy Bit Algorithm

For the tests on the Lena image, the Generalized TSIBA and TSIBA perform very well. A gain of nearly 28% bits is met over the Greedy bit allocation algorithm assuming 6dB per bit benefit for a *PSNR* of 35dB as seen in Figure 4.10.

Table 4.6 Table of Bit Rates for a *PSNR* of +35dB on Lena Image

Bit allocation algorithm	bits for *PSNR* 35dB
GTSIBA	108
TSIBA $k = 0.5$	133
TSIBA $k = 0.9$	108
Segall	142
Greedy	150

4.8 Conclusion

The TSIBA takes advantage of the non-gaussian distribution of the test data set. It does so by emphasizing the high variances with respect to the lower variance terms by powering the variance terms (λ_i^k), effectively changing the benefit of adding a bit. This indeed yields a generally optimal k for all dimensions. The TSIBA is very computationally intensive as it recodes the data set for as many values of k as there is and picks the best afterwards. This *a-posteriori* algorithm is useful in the sense that for a certain type of data set, a new optimal value of k does not need to be computed and can be assumed to be a certain precomputed value.

The Generalized TSIBA takes the idea to another level. Since it was observed that the value of k for the simple TSIBA changes for different bit rates and for different distributions, it makes sense to use different values of k for each dimension since different dimensions are likely to have different numbers of levels allocated to them and each dimension has a different distribution function. The values of k were trained offline and used in the new bit allocation formula (2.16) with iterative refinement to attain very good performance.

Both these algorithms could not get very close to the required bit rate, and therefore, the Greedy Level Allocation algorithm (Chapter 3) is used to distribute a maximal amount of levels to each dimension. The performance, however, only gains a small amount from this topping off procedure.

Chapter 5

Conclusion

This thesis is focused on the design of bit allocation schemes for GMM-based coders tested on speech LSF and for a simple transform coder tested on a small image set.

5.1 Summary of Work

5.1.1 Greedy Level Allocation Algorithm

Based on the philosophy of the Greedy bit allocation algorithm, where a single bit is distributed at a time, choosing the dimension which yields the largest decrease in distortion. The Greedy level allocation algorithm (GRLA) distributes a single level at a time, choosing the dimension which yields the largest benefit-to-cost ratio. The benefit is defined in terms of the decrease in *MSE* distortion of the dimension in question and the cost is defined as the difference in the number of bits before and after the allocation because giving a level does not correspond to giving a constant number of bits.

The given algorithm agrees with the procedure derived by Fox [7], which was also used to derive the Greedy bit allocation algorithm.

The GRLA has the advantage over the Greedy bit allocation algorithm because it can distribute levels in a ratio matching the optimal ratio of levels distributed to dimensions corresponding to the ratio of standard deviations. The Greedy bit allocation algorithm was constrained to giving full bits at a time, making the number of levels for each dimension a power of two. By having such a constraint, it is hard to get to the proper ratio of levels.

Because the GRLA distributes levels rather than bits, it also has some disadvantages over the Greedy bit allocation algorithm. First, by distributing levels rather than bits, one is nearly certain to not use all the given bits. The GRLA typically uses a number of bits that is very close to the given bit rate but still wastes a small fraction. Using levels rather than bits has its own intrinsic disadvantages: one now needs to send data as a vector and thus cannot code the dimensions independently. The third problem is computational complexity: since for every level, one needs to check which dimension yields the best benefit-to-cost ratio, all those ratios need to be computed at every iteration. There are many more levels than there are bits, and furthermore, since the KLT is used, a few dimensions usually dominate making the number of levels very large for those dimensions, and thus, the number of ratios to be computed increases.

The computational complexity is not as big a problem as it sounds because one can first allocate levels using an analytic algorithm such as the TSIBA or the GTSIBA and then use the GRLA to fill up the dimensions until no more bits are available.

5.1.2 Two-Stage Iterative Bit Allocation Algorithm

The TSIBA algorithm is a bit allocation algorithm solving the problem of negative bit allocations and non-integer number of levels for dimensions. It uses a modified version of the optimal bit allocation formula derived by Huang and Schultheiss [2] where the benefit of adding one bit can be varied. The algorithm simply allocates bits using a bit allocation formula, then every dimension which were allocated a negative number of bits are pruned. The formula is then computed again with only the remaining dimensions, until all dimensions have a non-negative number of bits allocated to them. Then, the number of levels is rounded down to be an integer. This rounding process may cause the bit rate to be much under the given rate, and thus, an offset is added to all dimensions. This whole process is repeated until the number of used bits is close enough to the given bit rate.

The benefit of adding one bit to a dimension depends on the number of bits already allocated and the distribution function of the dimension. Max [4] calculated that at lower bit rates, even Gaussian distribution functions do not yield 6dB per bit, but more around the 5dB per bit. Therefore, the bit allocation formula is modified to make the benefit per bit a variable that can be set when coding.

Finding the optimal benefit which minimizes the overall distortion can be very computationally expensive since one needs to recode with each value of benefit and see which gives the lowest overall distortion. The optimal value can be assumed to be a certain value however, then the formula does not have a high computational cost.

5.1.3 Generalized Two-Stage Iterative Bit Allocation Algorithm

Since the benefit of adding one bit to a dimension depends on the number of bits already allocated and the distribution function of the dimension, the previously explained concept can be generalized to have a different coefficient for each dimension. By again modifying the formula, one can add a weighting factor to each dimension variance which depends on the current number of levels and the distribution function of the dimension.

The GTSIBA is an iterative algorithm, it will use the formula for allocating bits with the weighting factors, then compute the new weighting factors according to the current number of bits. The iterative procedure goes on until convergence and therefore, the need for a nice weighting factor function is crucial.

The weighting factors are trained prior to coding. This permits an *a-priori* estimation which is done offline and therefore does not add to the computational cost of the bit allocation algorithm.

5.1.4 Comparison Between the Methods

Overall, the GTSIBA usually performs best. This is to be expected as the GTSIBA takes into account the benefit depending on the current number of bits allocated and distribution function of each dimension separately. The TSIBA usually performs well too, though it finds an overall benefit for all dimensions.

The GRLA is applied on top of both algorithms and adds a number of levels to each dimension, making the bit wasting very small.

Both the TSIBA and GTSIBA with GRLA can gain around one bit over the Greedy bit allocation algorithm when used on the GMM-based coder. The gain is even more impressive when the bit allocation algorithms are used on the image coder. A gain of 28% is achieved when using the GTSIBA and 28% when using the TSIBA with the optimal k over the Greedy bit allocation algorithm.

5.2 Future Work

5.2.1 Generalizing the TSIBA by Powering the Variance of Each Dimension

The generalization of the TSIBA to the GTSIBA did not emphasize the benefit of each dimension, but simply weighted the variance of the dimension in order to bring the given bit rate to the required level. One can in fact purely extend the concept of the TSIBA towards powering the variance, where the powering is different for each dimension depending on the distribution and the number of bits that has been allocated.

The generalization would work even better if the benefit was calculated in the GRLA. The GRLA allocates one level at a time, and for each new level allocated, the decrease in distortion is different. Therefore, using the proper benefit would yield the best possible allocation.

5.2.2 An Analytic GTSIBA

Another direction to study is to make the GTSIBA analytic rather than iterative since an iterative algorithm does not guarantee an ending time. The algorithm would then be similar to Segall's algorithm.

5.2.3 The Sensitivity Matrix

Since the quality of an LSF quantizer is measured using the *LSD*, it is obvious that one should find a method of alloating bits which is weighted towards components which affect the *LSD* more rather than optimized for the *MSE*.

The use of the sensitivity matrix can be introduced into the bit allocation algorithms so that the bit allocation depends on *LSD*.

References

[1] A. D. Subramaniam and B. D. Rao, "PDF optimized parametric vector quantization of speech line spectral frequencies," *IEEE Transactions on Speech and Audio Processing,* vol. 11, no. 2, pp. 130–142, March 2003.

[2] J. Huang and P. Schultheiss, "Block quantization of correlated Gaussian random variables," *IEEE Transactions on Communications,* vol. 11, no. 3, pp. 289–296, September 1963.

[3] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory,* vol. 28, no. 2, pp. 129–137 , March 1982.

[4] J. Max, "Quantizing for minimum distortion," *IEEE Transactions on Information Theory,* vol. 6, no. 1, pp. 7–12 , March 1960.

[5] K. Sayood, *Introduction to data compression,* 2nd edition, Morgan Kaufmann, 2000.

[6] A. Segall, "Bit allocation and encoding for vector sources," *IEEE Transactions on Information Theory,* vol. 22, no. 2, pp. 162–169, March 1976.

[7] B. Fox, "Discrete optimization via marginal analysis," *Management Science,* vol. 13, no. 3, pp. 210–216, November 1966.

[8] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific Journal of Mathematics,* vol. 16, no. 1, pp. 1–3, 1966.

[9] A. Gersho and R. M. Gray, *Vector quantization and signal compression,* Kluwer Academic Publishers, 1992.

[10] K. K. Paliwal and B. S. Atal, "Efficient vector quantization of LPC parameters at 24 bits/frame," *IEEE Transactions on Signal and Audio Processing,* vol. 1, no. 1, pp. 3–14, January 1993.

[11] P. Kabal, "Quantizers for the gamma distribution and other symmetrical distributions," IEEE Transactions on Acoustic, Speech and Signal Processing, vol. 32, no. 4, pp. 836–841, August 1984.

[12] D. O'Shaughnessy, *Speech communications : human and machine,* Wiley-IEEE Press, 2nd edition, November 1999.

[13] P. F. Panter and W. Dite, *"Quantization distortion in pulse count modulation with nonuniform spacing of levels,"* Proc. IRE, vol. 39, no. 1, pp. 44, January 1951.

[14] J. A. Bilmes, "A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models" Computer Science Division, University of California Berkeley, April 1998.

[15] F. Itakura, "Line spectrum representation of linear predictive coefficients of speech signals," *Journal of Acoustic Society of America,* vol. 57, no. 1, pp. 535, April 1975.

[16] F. K. Soong and B.-H. Juang, "Line spectral pair (LSP) and speech data compression" *IEEE International Conference on ICASSP '84,* vol. 9, no. 1, pp. 37–40, May 1984.

[17] P. Hedelin and J. Skoglund, "Vector quantization based on Gaussian mixture models" *IEEE Transactions on Speech and Audio Processing,* vol. 8, no. 4, pp. 385–401, July 2000.

[18] T. D. Lookabaugh and R. M. Gray, "High-resolution quantization theory and the vector quantizer advantage," *IEEE Transactions on Information Theory,* vol. 35, no. 5, pp. 1020-1033, September 1989.

Appendix A

Linear Predictive Coefficients

A.1 Linear Combination

The LPC coefficients are the coefficients a_j in $\hat{X}_i = -\sum_{j=1}^{p} a_j X_{i-j}$ which would minimize the mean squared error (MSE)

$$\begin{aligned} E[E_i^2] &= E[(X_i - \hat{X}_i)^2] && \text{(A.1)} \\ &= E[(X_i + \sum_{j=1}^{p} a_j X_{i-j})^2] && \text{(A.2)} \\ &= E[(\boldsymbol{a}^T \boldsymbol{X_{i:i-j}})^2] && \text{(A.3)} \\ &= \boldsymbol{a}^T E[\boldsymbol{X_{i:i-p}} \boldsymbol{X_{i:i-p}}^T] \boldsymbol{a} && \text{(A.4)} \\ &= \boldsymbol{a}^T \boldsymbol{R} \boldsymbol{a} && \text{(A.5)} \end{aligned}$$

where $\boldsymbol{a}^T = [1, a_1, ..., a_p]$, $\boldsymbol{R}$ is the covariance matrix $\boldsymbol{X_{i:i-p}} \boldsymbol{X_{i:i-p}}^T$ and $\boldsymbol{X^T_{i:i-p}} = [X_i, ..., X_{i-p}]$.

Now minimizing the average error under the constraint that $a_0 = 1$, i.e. that for $\boldsymbol{b} = [1, 0, ..., 0]^T$, $\boldsymbol{b}^T \boldsymbol{a} = 1$, differentiating the following cost function and setting the derivative to zero, one gets

$$\begin{aligned} \frac{\partial \nu}{\partial \boldsymbol{a}} &= \frac{\partial}{\partial \boldsymbol{a}} \left[\boldsymbol{a}^T \boldsymbol{R} \boldsymbol{a} - \lambda \boldsymbol{b}^T \boldsymbol{a} \right] \\ 0 &= \boldsymbol{R} \boldsymbol{a} - \lambda \boldsymbol{b} \\ \boldsymbol{R} \boldsymbol{a} &= [\lambda, \boldsymbol{0}]^T. \end{aligned} \qquad \text{(A.6)}$$

When looking at the frequency contents of a speech signal, one can observe many interesting properties, the most important one is the presence of peaks called formants. These peaks in the frequency-domain are modelled by the LPC in the time-domain. Two poles are needed to modelled each formant and, typically, an additional 2-4 poles for the zeros and general spectral shaping [12]. Higher-order LPC would give an even better representation of the spectral envelope, and when the order tends to infinity, the estimated spectral envelope tends to the actual spectral envelope.

A.2 Line Spectral Frequencies

One can pass from LPC to LSF by the following relation [16]

$$\begin{aligned} A(z) &= 1/2[P(z) + Q(z)] && \text{(A.7)} \\ P(z) &= A(z)\left[1 + z^{-(m+1)}\frac{A(z^{-1})}{A(z)}\right] && \text{(A.8)} \\ Q(z) &= A(z)\left[1 - z^{-(m+1)}\frac{A(z^{-1})}{A(z)}\right]. && \text{(A.9)} \end{aligned}$$

The LSF have three important properties:

1. all zeros of P(z) and Q(z) are on the unit circle
2. zeros of P(z) and Q(z) are interlaced with each other
3. the minimum phase property of A(z) is easily preserved after quantization of the zeros of P(z) and Q(z).

LSFs are thus more robust and can replace LPC as data to be sent.

Appendix B

Karhunen-Loève Transform

B.1 KLT as a Transform

The Karhunen-Loève Transform (KLT) is the eigenvector matrix of the covariance matrix of a dataset. It fulfills the requirements of a good transform: it is invertible since the inverse is simply its transpose, and it conserves distortion since all unitary transforms conserve the mean-square distortion and the KLT is unitary. For simplicity, X is assumed to be a zero-mean random variable.

$$MSE = ||\boldsymbol{UX}||_2^2 = (\boldsymbol{UX})^T(\boldsymbol{UX}) = \boldsymbol{X}^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{X} = \boldsymbol{X}^T\boldsymbol{X} = ||\boldsymbol{X}||_2^2 \tag{B.1}$$

where $\boldsymbol{U}$ is the KLT matrix, i.e. eigenvectors matrix.

B.2 Optimality of the KLT

The KLT is an optimal orthogonal block transform for Gaussians because of 3 aspects.

First, the KLT provides maximal energy compaction, it minimizes the number of coefficients required to get an accurate representation of the data. Defining the measure of flatness as

$$log(\prod_{i=0}^{k-1} \sigma_{Y_i}^2) = \sum_{i=0}^{k-1} log(\sigma_{Y_i}^2) \tag{B.2}$$

where σ_{y_i} is the standard deviation of variable y_i.

To make this measure maximally flat, i.e. maximize (B.2), one would need all σ_{Y_i}

to be equal. Thus, minimizing it, i.e. making it minimally flat, would optimize energy compaction.

Second, the KLT maximizes the coding gain $G(A) = \frac{\sigma_x^2(A)}{\left(\prod_{k=1}^{d} \sigma_k^2(A)\right)^{1/d}}$. Since the numerator stays constant with unitary transformations, this can be seen as a result of the maximal energy compaction which minimizes the denominator.

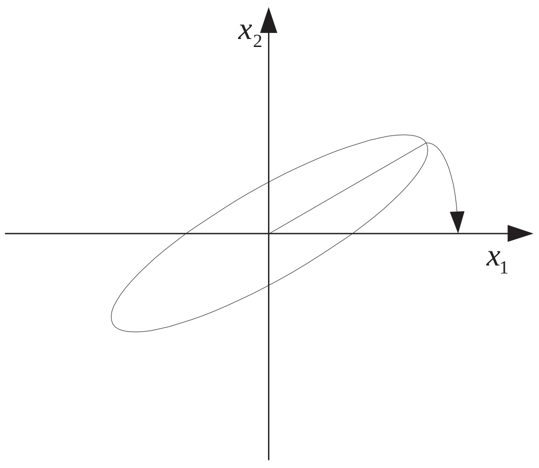
Third, the KLT decorrelates the random variables, and assuming Gaussian distribution, decorrelation also means independence. Say $Y = \boldsymbol{U}^T \boldsymbol{X}$, then the covariance matrix R_{YY} is given by

$$\begin{aligned} E[\boldsymbol{Y}\boldsymbol{Y}^{\boldsymbol{T}}] &= E[\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{U}] && \text{(B.3)} \\ &= \boldsymbol{U}^{\boldsymbol{T}} E[\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}}]\boldsymbol{U} && \text{(B.4)} \\ &= \boldsymbol{U}^{\boldsymbol{T}} \boldsymbol{R}_{\boldsymbol{X}\boldsymbol{X}} \boldsymbol{U} && \text{(B.5)} \\ &= diag \lambda_i && \text{(B.6)} \end{aligned}$$

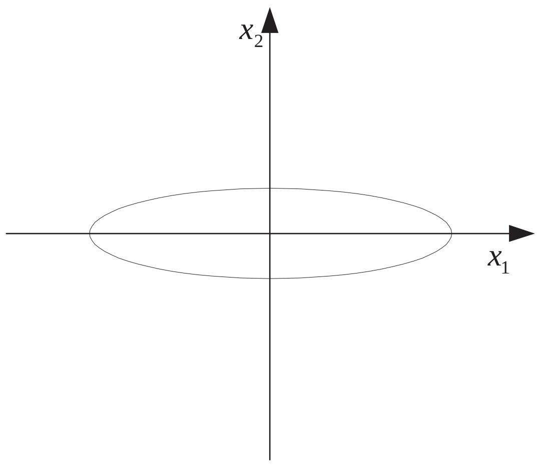
where X is a set of dependent Gaussian variables and $\boldsymbol{U}$ and λ_i are respectively the eigenvectors and eigenvalues of its autocorrelation, or, since $\boldsymbol{X}$ is zero-mean, covariance matrix $\boldsymbol{R}_{\boldsymbol{X}\boldsymbol{X}}$ [9].

The KLT can also be seen as a rotation which aligns the axis and the distribution function as seen in Figure B.1 which shows a two-dimensional case in which the KLT rotates the data by using an orthogonal transform.

Training the KLT comes directly from the definition of it: one simply computes the eigenvectors of the covariance matrix $E[\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}}] - E[\boldsymbol{X}]^2$, where, usually, $E[\boldsymbol{X}]$ is assumed to be $\mathbf{0}$.



(a) Data Distribution Before Decorrelation



(b) Data Distribution After KLT Decorrelation

Fig. B.1 Effect of KLT

Appendix C

Expectation-Maximization Algorithm

The expectation-maximization (EM) algorithm is a general training algorithm mostly used for estimating parameters of a Hidden Markov Model (HMM) and Gaussian Mixture Model (GMM) described in section 2.1. The EM algorithm is an iterative algorithm which refines the parameter set so that at each iteration, the likelihood of the parameter set generating the training data does not decrease.

GMMs have the following probability density function:

$$p_l(x_i|\mu_l, \Sigma_l) = \frac{1}{(2\pi)^{d/2}|\Sigma_l|^{1/2}} e^{(-\frac{1}{2}(x_i-\mu_l)^T \Sigma_l^{-1}(x_i-\mu_l))}. \tag{C.1}$$

Let us denote Φ as the parameter set for the GMM containing the mixture weights α_l, and the means μ_l and covariance matrices Σ_l denoted as ϕ_l and Φ' as the parameter set of the previous iteration. Here is a derivation of the GMM update equations from [14]

First, start by defining the log likelihood function.

$$P(x, y|\Phi) = \sum_{i=1}^{N} log(P(x_i|y_i))P(Y) = \sum_{i=1}^{N} log(\alpha_{y_i} p_{y_i}(x_i|\phi_{y_i})) \tag{C.2}$$

where x_i is the observation at time i, y_i is the unobserved data at time i, i.e. the Gaussian mixture generating x_i and α_{y_i} is the mixture y_i prior probability.

Then, denoting y_i by l, one defines the Q-function as

$$\begin{aligned} Q(\Phi, \Phi') &= \sum_{l=1}^{M}\sum_{i=1}^{N} log(\alpha_l p_l(x_i|\phi_l))p(l|x_i, \Phi') && \text{(C.3)} \\ &= \sum_{l=1}^{M}\sum_{i=1}^{N} log(\alpha_l)p(l|x_i, \Phi') + \sum_{l=1}^{M}\sum_{i=1}^{N} log(p_l(x_i|\phi_l))p(l|x_i, \Phi') && \text{(C.4)} \end{aligned}$$

which gives one part only dependent on α_l and the another term on ϕ_l. Then, one differentiates the Q-function with respect to the variables to be optimized.

Starting with the weights of the mixtures under the constraint that $\sum_{l=1}^{M} \alpha_l = 1$, one gets

$$\begin{aligned} \frac{\partial Q(\Phi, \Phi')}{\partial \alpha_l} &= \frac{\partial}{\partial \alpha_l}[\sum_{l=1}^{M}\sum_{i=1}^{N} log(\alpha_l)p(l|x_i, \Phi') + \lambda(\sum_{l=1}^{M} \alpha_l - 1)] && \text{(C.5)} \\ 0 &= \sum_{i=1}^{N} \frac{1}{\alpha_l} p(l|x_i, \Phi') + \lambda. && \text{(C.6)} \\ & && \text{(C.7)} \end{aligned}$$

Solving for the constraint gives $\lambda = -N$, the resulting equation is

$$\alpha_l = \frac{1}{N}\sum_{i=1}^{N} p(l|x_i, \Phi'). \quad \text{(C.8)}$$

For the other two parameters of the GMM, the mean and the covariance matrix, we need to expand the definition of the Q-function by expanding the $p(x_i|\phi)$ for mixture l to

$$p_l(x_i|\mu_l, \Sigma_l) = \frac{1}{(2\pi)^{d/2}|\Sigma_l|^{1/2}} e^{(-\frac{1}{2}(x_i-\mu_l)^T \Sigma_l^{-1}(x_i-\mu_l))} \quad \text{(C.9)}$$

and

$$Q(\Phi, \Phi') = \sum_{l=1}^{M}\sum_{i=1}^{N} log\left(\frac{1}{(2\pi)^{d/2}|\Sigma_l|^{1/2}} e^{(-\frac{1}{2}(x_i-\mu_l)^T \Sigma_l^{-1}(x_i-\mu_l))}\right) p(l|x_i, \Phi'). \quad \text{(C.10)}$$

Now, differentiate with respect to the means of the mixtures.

$$\frac{\partial Q(\Phi,\Phi')}{\partial \mu_l} = \frac{\partial}{\partial \mu_l}[\sum_{l=1}^{M}\sum_{i=1}^{N}\left(-\frac{1}{2}log(|\Sigma_l|)-\frac{1}{2}(x_i-\mu_l)^T\Sigma_l^{-1}(x_i-\mu_l)\right)p(l|x_i,\Phi')] \quad (C.11)$$

$$0 = \sum_{i=1}^{N}\Sigma_l^{-1}(x_i-\mu_l)p(l|x_i,\Phi'). \quad (C.12)$$

Setting the above to 0 and solving for μ_l, we obtain

$$\mu_l = \frac{\sum_{i=1}^{N}x_i p(l|x_i,\Phi')}{\sum_{i=1}^{N}p(l|x_i,\Phi')}. \quad (C.13)$$

Now, we need to rewrite the Q-function again and deriving with respect to Σ_l^{-1}

$$Q(\Phi,\Phi') = \left[\sum_{l=1}^{M}\left[\frac{1}{2}log(|\Sigma_l^{-1}|)\sum_{i=1}^{N}p(l|x_i,\Phi')-\frac{1}{2}\sum_{i}^{N}p(l|x_i,\Phi')tr(\Sigma_l^{-1}N_{l,i})\right]\right] \quad (C.14)$$

$$\frac{\partial Q(\Phi,\Phi')}{\partial \Sigma_l^{-1}} = \frac{1}{2}\sum_{i=1}^{N}p(l|x_i,\Phi')(2\Sigma_l-diag(\Sigma_l))-\frac{1}{2}\sum_{i=1}^{N}p(l|x_i,\Phi')(2N_{l,i}-diag(N_{l,i})) \quad (C.15)$$

$$= \frac{1}{2}\sum_{i=1}^{N}p(l|x_i,\Phi')(2M_{l,i}-diag(M_{l,i})) \quad (C.16)$$

$$= 2S-diag(S) \quad (C.17)$$

where $N_{l,i}=(x_i-\mu_l)(x_i-\mu_l)^T$, $M_{l,i}=\Sigma_l-N_{l,i}$ and $S=\frac{1}{2}\sum_{i=1}^{N}p(l|x_i,\Phi')M_{l,i}$

Setting the derivative to zero, we obtain that $S=0$ and solving for Σ_l, we get

$$0=\sum_{i=1}^{N}p(l|x_i,\Phi')(\Sigma_l-N_{l,i}) \quad (C.18)$$

or

$$\Sigma_l = \frac{\sum_{i=1}^{N} p(l|x_i, \Phi')(x_i - \mu_l)(x_i - \mu_l)^T}{\sum_{i=1}^{N} p(l|x_i, \Phi')}. \tag{C.19}$$

The EM algorithm directly comes from the previous 3 update equations. Here is the algorithm for the estimation of GMM parameters

Algorithm C.0.1: EM(x, Φ')

```
repeat
    comment: Expectation step (E-Step):
    for each $j \in 1$ to $N$
    do  $p(x_j|\Phi) = \sum_{l=1}^{M} \alpha_l p_l(x_j|\phi_l)$
        for each $l \in 1$ to $M$
        do  $p(l|x_j, \phi_l) = \frac{\alpha_l p_l(x_j|\phi_l)}{\sum_{k=1}^{M} \alpha_k p_k(x_j|\phi_k)}$

    comment: Maximization step (M-Step):
    for each $l \in 1$ to $M$
    do  $\alpha_l = \frac{1}{N} \sum_{j=1}^{N} p(l|x_j, \phi_l)$
        $\mu_l = \frac{\sum_{j=1}^{N} x_j p(l|x_j, \phi_l)}{\sum_{j=1}^{N} p(l|x_j, \phi_l)}$
        $\Sigma_l = \frac{\sum_{j=1}^{N} p(l|x_j, \phi_l)(x_j - \mu_l)(x_j - \mu_l)^T}{\sum_{j=1}^{N} p(l|x_j, \phi_l)}$
until $\log(L(\Phi|X, Y)) > THRESHOLD$
```

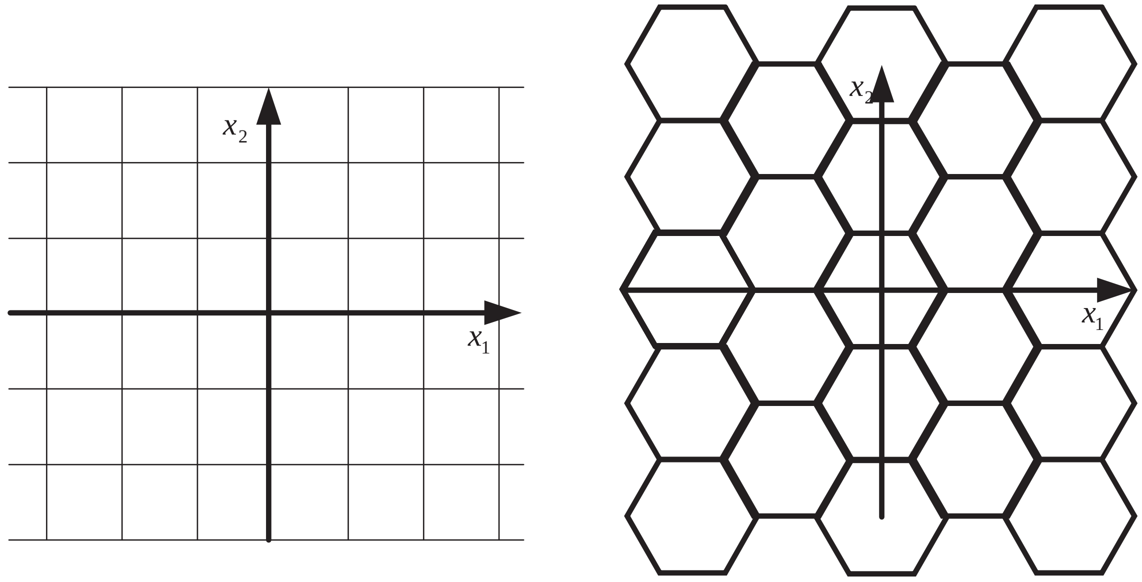
where M is the number of mixtures, N is the number of training vectors, α_l is the prior probability of cluster l, x_j is vector j of the training set, ϕ_l is the set of parameters μ_l and Σ_l of cluster l, μ_l is the mean vector of cluster l, Σ_l is the covariance matrix of cluster l and $(L(\Phi|X, Y))$ is the likelihood of the parameters with the observed data.

Appendix D

VQ vs SQ advantages

A vector quantizer has three advantages over a uniform scalar quantizer.

The first advantage is space-filling advantage shown in Figure D.1. With hypercubes, filling the plane results in larger average distortion than with hexagons. The best shape (distortion wise) would be hyperspheres, but they are not realizable for obvious reasons.



(a) Space-Filling Using a Uniform Scalar Quantizer(Hypercubes) (b) Space-Filling Using a Vector Quantizer(Hexagonal Shape)

Fig. D.1 Space-Filling Advantage

The second advantage is the shape advantage as illustrated on Figure D.2. For non-uniform distribution functions, the SQ yields suboptimal quantization cells when applied to many dimensions on certain non-uniform PDF such as the Gaussian distribution. As one can see, the region of importance has less quantization points than in the VQ case and the less important region has more, which yields a worse average distortion.

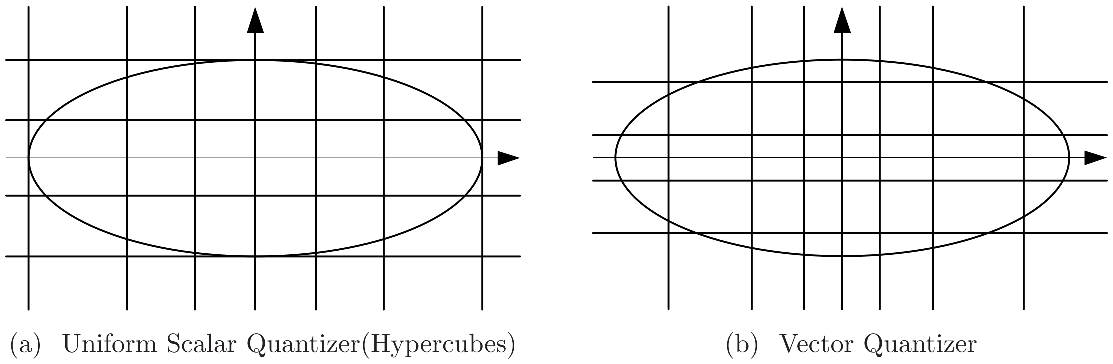
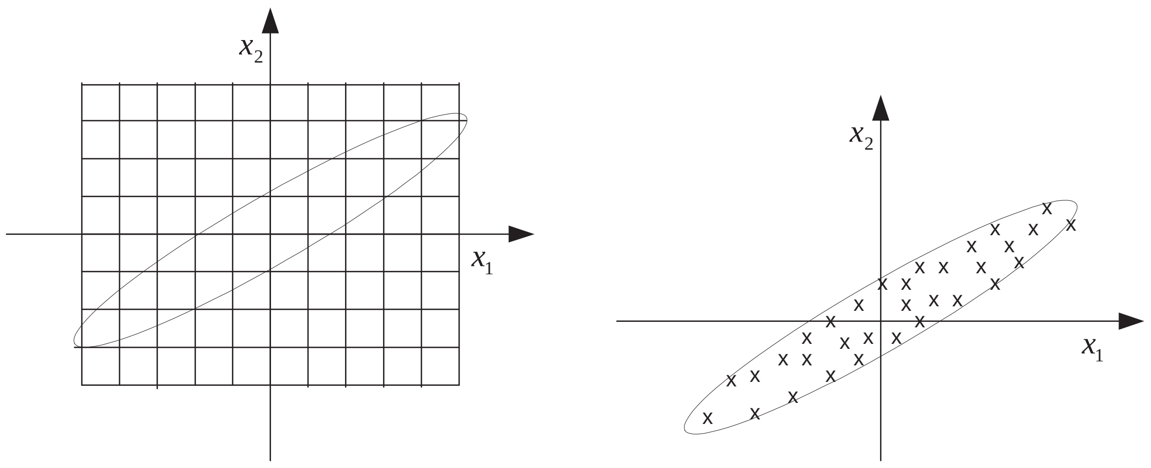

(a) Uniform Scalar Quantizer(Hypercubes) (b) Vector Quantizer

Fig. D.2 Shape Advantage

The third advantage is the memory advantage. It is best illustrated in the situation where there is correlation between the dimensions as shown on Figure D.3 in a two-dimensional Gaussian variable. As one can see, when using hypercubes, many codepoints are wasted because hardly any data points would map to these codepoints. The vector quantizer effectively distributes codepoints where they are useful.



(a) Uniform Scalar Quantizer(Hypercubes) (b) Vector Quantizer

Fig. D.3 Memory Advantage

Appendix E

Other Bit Allocation Algorithms

E.1 Pruning Used by Subramaniam

Subramaniam used another method in [1] to allocate levels. The algorithm in question is a pruning algorithm which is not optimal but is low-complexity. The idea is to over-allocate levels to dimensions and then remove some. Here is the algorithm taken from [1]
Algorithm E.1.1: PRUNING($bits$)

```
$L_{i,j} \leftarrow \lceil 2^{b_{i,j}} \rceil$
$k \leftarrow d$
while $k \geq 1$
    do {
        $P_i = \prod_{j=1}^{d} L_{i,j}$
        $N_{i,k} = \lfloor L_{i,k}(1 - 2^{b_i}/P_i) \rfloor$
        if $k = 1$
          then $N_{i,k} \leftarrow N_{i,k} + 1$
        $L_{i,k} \leftarrow L_{i,k} - N_{i,k}$
        $k \leftarrow k - 1$
```

where $L_{i,k}$ are the number of levels allocated to cluster i for dimension k.

As will be seen later, this algorithm does use very effectively the given bits but does not reach the performance of the presented algorithms.

E.2 Segall's Bit Allocation

Segall's algorithm [6] is an analytic algorithm which goal is to solve for the optimal bit allocations using the different benefits of adding 1 bit at different number of bits.

It moreover has a routine to constrain the total number of bits to the desired number of bits, and guarantees non-negative bit allocations.

The algorithm will minimize

$$D = \sum_{j=1}^{m} \sigma_j^2 k(B_j) \tag{E.1}$$

subject to

$$\sum_{j=1}^{m} B_j = C \tag{E.2}$$

and

$$B_j \geq 0, j = 1, 2, ..., m. \tag{E.3}$$

Optimizing for minimum distortion, one obtains

$$B_j = \left\{ \begin{array}{r} B_j^* = h\left(\frac{\theta^*}{\sigma_j^2} k'(0)\right) \text{ if } 0 \leq \theta^* < \sigma_j^2 \\ 0, \text{ if } \theta^* \geq \sigma_j^2 \end{array} \right\} \tag{E.4}$$

where θ^* is the unique root of the equation

$$S(\theta) = \sum_{j:\sigma_j^2 \geq \theta} h\left(\frac{\theta}{\sigma_j^2} k'(0)\right) \tag{E.5}$$

where h is the inverse function of k'.