5.3

Fundamentals of Vector Quantization

Mohammad A.U. Khan
COMSATS Institute of Information Technology, Pakistan
Mark J.T. Smith
Purdue University, West Latayette

1 Introduction

In this age of information, we see an increasing trend toward the use of digital representations for audio, speech, images, and video. Much of this trend is being fueled by the exploding use of computers and multimedia computer applications. The high volume of data associated with digital signals, particularly digital images and video, has stimulated interest in algorithms for data compression. Many such algorithms are discussed elsewhere in this book.

At the heart of all these algorithms is quantization, a field of study that has matured over the last few decades. In simplest terms, quantization is a mapping of a large set of values to a smaller set of values. The concept is illustrated in Fig. 1A, which shows on the left a sequence of unquantized samples with amplitudes assumed to be of infinite precision, and on the right that same sequence quantized to integer values. Obviously, quantization is an irreversible process, because it involves discarding information. If done wisely, the error introduced by the process can be held to a minimum. The generalization of this notion is called *vector quantization*, commonly denoted VQ. It too is a mapping from a large set to a smaller set but involves quantizing blocks of samples together. The conceptual notion of VQ is illustrated in Fig. 1B. Blocks of samples, which we view as vectors, are represented by code vectors stored in a codebook—a process called encoding. The codebook is typically a table stored in a digital memory, where each table entry represents a different code vector. A block diagram of the encoder is shown in Fig. 2. The output of the encoder is a binary index that represents the compressed form of the input vector. The reconstruction process, which is called decoding, involves looking up the corresponding codevector in a duplicate copy of the codebook, assumed to be available at the decoder.

The general concept of VQ can be applied to any type of digital data [36–41]. For a one-dimensional (1D) signal as illustrated in Fig. 1B, vectors can be formed by extracting contiguous blocks from the sequence. For two-dimensional signals (i.e., digital images) vectors can be formed by taking two-dimensional (2D) blocks, such as rectangular blocks, and unwrapping them to form vectors. Similarly, the same idea can be applied to three-dimensional (3D) data (i.e., video), color and multispectral data, transform coefficients, and so forth.

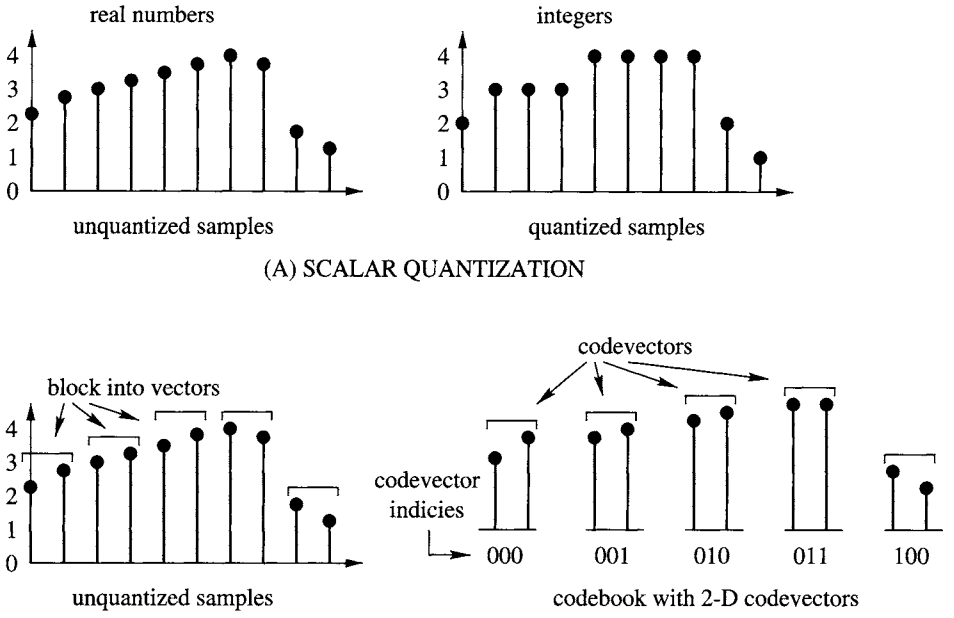


FIGURE 1 Comparison of scalar quantization with vector quantization.

2 Theory of Vector Quantization

Although conceptually simple, there are a number of issues associated with VQ that are technically complex and relevant for an in-depth understanding of the process. To address these issues, such as design and optimality, it is convenient to treat VQ in a mathematic framework. Toward this end, we can view VQ as two distinct operations—encoding and decoding—shown explicitly in Fig. 2. The encoder $\mathcal{E}$ performs a mapping from k-dimensional space $\mathcal{R}^k$ to the index set $\mathcal{I}$, and the decoder $\mathcal{D}$ maps the index set $\mathcal{I}$ into the finite subset $\mathcal{C}$, which is the codebook. The codebook has a positive integer number of code vectors that defines the codebook size. In this chapter, we will use N to denote the codebook size and $\mathbf{y}_i$ to denote the code vectors, which are the elements of $\mathcal{C}$. The bit rate R associated with the VQ depends on N (the number of code vectors in the codebook) and the vector dimension k. Since the bit rate is the number of bits per sample,

$$R = (\log_2 N)/k. \tag{1}$$

For simple scalar quantizers, such as the one illustrated in Fig. 1A, bit rates are typically integer numbers. However, for VQ it is natural to have fractional bit rates such as 1/2, 3/4, 16/3. The structure of VQ in this regard inherently provides greater flexibility in terms of rate granularity.

The operation associated with the VQ decoder is extremely simple, involving no arithmetic at all. Conversely, the encoding procedure is complex, because a best-matching vector decision must be made from among many candidate code vectors. To select a best matching codevector, we use a numerically computable distortion measure $d(\mathbf{x}, \mathbf{y}_i)$, where low values of $d(\cdot, \cdot)$ imply a good match. There are many distortion measures that can be considered for quantifying the "quality of match" between two vectors $\mathbf{x}$ and $\mathbf{y}$, the most common of which is the *squared error* given by

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &= (\mathbf{x} - \mathbf{y})^t (\mathbf{x} - \mathbf{y}) \\ &= \sum_{\ell=1}^{k} (x[\ell] - y[\ell])^2, \end{aligned}$$

where $x[\ell]$ and $y[\ell]$ are the elements of the vectors $\mathbf{x}$ and $\mathbf{y}$, respectively. To encode a vector $\mathbf{x}$, distortions are computed

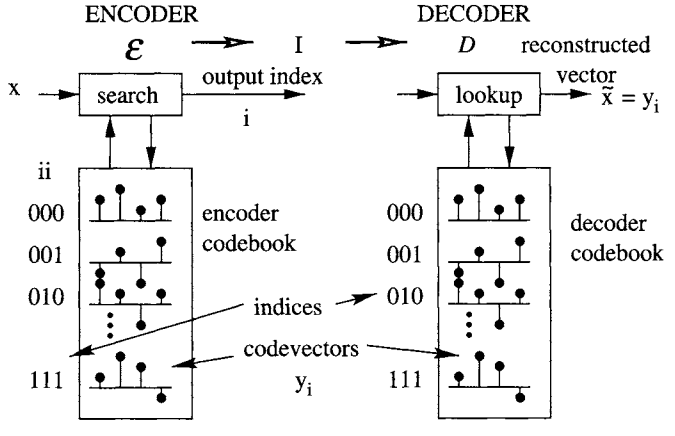
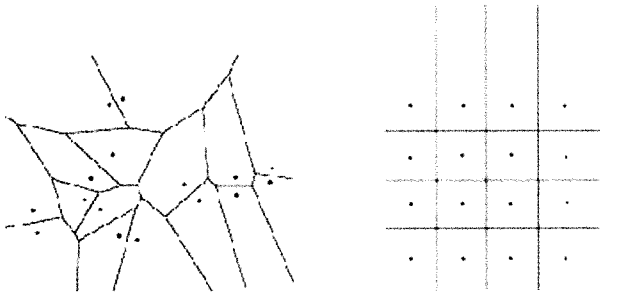


FIGURE 2 Block diagram of a vector quantization encoder and decoder.



(A) Partition cells for a 2-D VQ (B) Partition cells corresponding to scalar quantization

FIGURE 3 Illustration of the partition cells associated with VQ and scalar quantization.

between it and each codevector $\mathbf{y}_i$ in the codebook. The code vector producing the smallest distortion is selected as the best match and the index associated with that code vector is used for the representation.

This process of encoding has an interesting and useful interpretation in the k-dimensional space. The set of codevectors defines a *partition of* $\mathcal{R}^k$ into N cells V_i, where $i = 1, 2, \ldots, N$. If we let $\mathcal{Q}(\cdot)$ represent the encoding operator, then the ith cell is defined by

$$V_i = \{\mathbf{x} \in \mathcal{R}^k : \mathcal{Q}(\mathbf{x}) = \mathbf{y}_i\}. \tag{2}$$

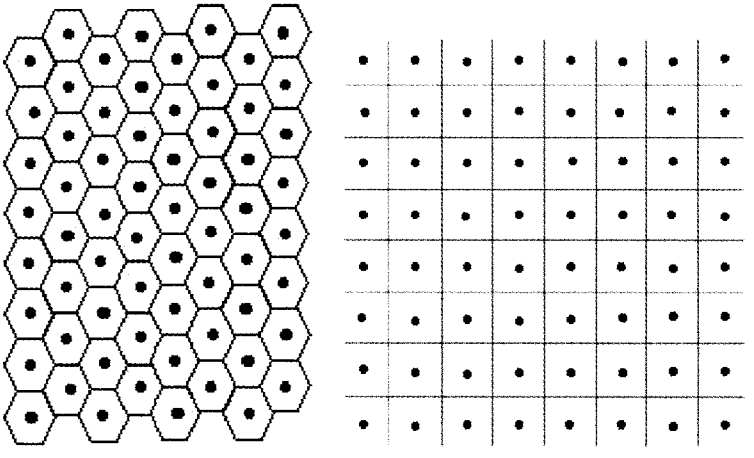
Partitions of this type that are formed uniquely from the codebook and a nearest neighbor distortion metric such as the squared error distortion are called *Voronoi partitions*. The notion of partitioning can be visualized easily in two-dimensions, an illustration of which is shown in Fig. 3A. Here, each vector has two elements (x_1, x_2) and consequently is a point in the 2D space. That is, both the input vectors and codevectors are points in this space. The encoding procedure defines a unique partitioning of the space as shown in the figure, where the black dots denote code vectors.

The quality of performance of a VQ is typically measured by its average distortion for a given input source. In practice, sources are typically signal samples, image pixels, or some other data output associated with a signal that is being compressed. Whatever the source, average distortion measures are typically used to quantify the performance of a vector quantizer—the smaller the average distortion the better the performance.

Vector quantizers are of interest because they can be designed to yield better performance than a scalar quantizer [31]. In fact, VQs can generally contribute three types of performance gain: cell shape gain, density shape gain, and source correlation gain. All three can be explained in a straightforward way using 2D vectors and partition diagrams.

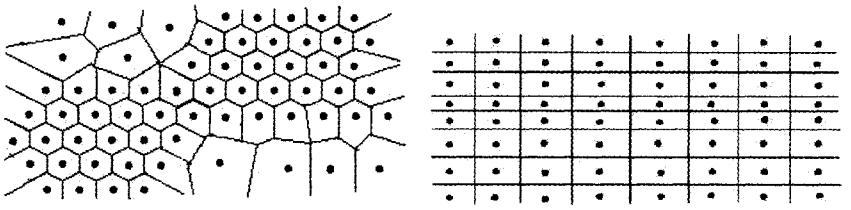
Consider first an input source that is independent (i.e., no dependencies among the samples) and uniformly distributed over the range. Figure 4 depicts the 2D partition diagram for such a source, in this case with 64 cells. What we see is that VQ is able to realize hexagonal-shaped cells because the partitions are defined in 2D via equation (2). If we consider scalar quantization, the effective partitioning in 2D is rectangular. The performance of a quantizer for a particular source, as we said before, can be judged by its average distortion. From the partition diagram, this can be viewed in terms of the average distance to the center of the cell or equivalently the packing efficiency of the cells. Viewing the average quantization distortion in terms of cell shape geometry makes it easy to see the gains achieved by VQ. Clearly hexagonal cells will have lower overall distortion than rectangular cells. Furthermore, it can be shown that the hexagonal geometry provides the minimum distortion. This type of gain is called *shape gain* and the magnitude of this gain increases with dimension.

Now let's consider a bimodal density source. Figure 5 illustrates a 2D partition diagram for both a VQ and scalar



(A) (B)

FIGURE 4 Uniform density source: Illustration of partition diagram with 64 cells associated with (A) vector quantization and (B) scalar quantization.



(A) (B)

FIGURE 5 Vector quantization density gain: Illustration of partition diagram with 64 cells associated with (A) vector quantization, and (B) scalar quantization.

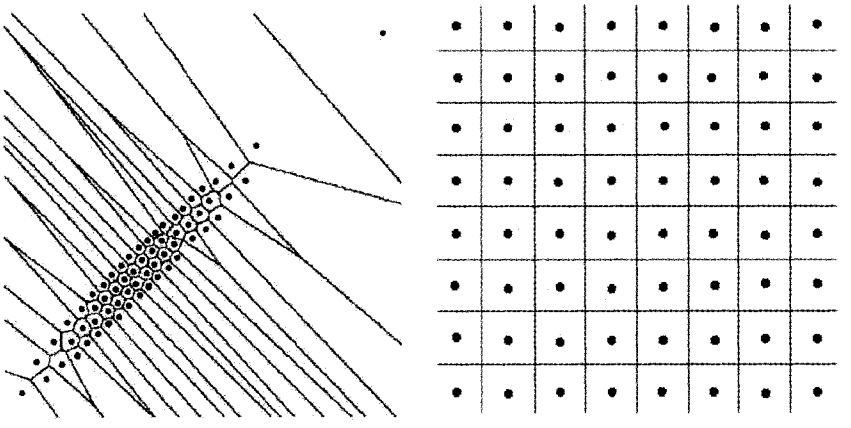
quantizer. Notice that in the VQ case, the cell shapes are hexagons and are placed in the area where the source has high density. For the scalar quantization case, the cells are rectangular in shape and some cells are placed in an area where the source does not have significant density. Thus, we see that VQ achieves a density-related advantage.

Up until now, we have observed that VQ provides better cell geometry and places cell according to varying source density. However, we have not explored the performance of VQ for nonindependent signal sources (i.e., sources that have dependencies among the samples), which is often the case for many signals. To understand this issue, let us consider a first-order Gauss-Markov source that has a strong linear correlation among source samples. High correlation among samples implies that each vector will have elements that are close in value. For the 2D case as illustrated in Fig. 6, this results in points that cluster along a diagonal in 2D space. VQ when applied directly can exploit this distribution as shown in the figure by creating cells that minimize the average distance to the centroid. Scalar quantization when applied directly has no way to capture sample-to-sample dependencies. Rather, it tends to cover the broader space defined by mapping the scalar quantized values to two dimensions as shown in Fig. 6B. Thus we see geometrically that VQ (unlike scalar quantization) can directly exploit signal dependencies.

3 Design of Vector Quantizers

The key element in designing a VQ is determining the codebook for a given input source. In practice, the input source is represented by a large set of representative vectors called a *training set*. Over the years, there have been many algorithms proposed for VQ design. The most widely cited is the classic iterative method attributed to Linde, Buzo, and Gray, a.k.a. the LBG algorithm [2]. The LBG algorithm is fashioned around certain necessary conditions associated with the distinct encoder and decoder operations implicit in VQ. The first of these conditions states that for a fixed decoder codebook, an optimal encoder partition of $\mathcal{R}^k$ is the one that satisfies the *nearest neighbor rule*, which says that we map each input vector to the cell V_i producing the smallest distortion. By that measure, we are selecting the code vector that is nearest to the input vector.

The second optimality condition is the *centroid condition*. It states that for a given encoder partition cell, the optimal decoder codeword is the centroid of that cell, where the centroid of cell V_i is the vector $\mathbf{y}^*$ that minimizes $E\{d(\mathbf{x}, \mathbf{y})|\mathbf{x} \in V_i\}$, the average distortion in that cell. The centroid is a function of the distortion measure and is different for different distortion measures. For the popular



(A) (B)

FIGURE 6 Gauss-Markov source: Illustration of partition diagram with 64 cells associated with (A) vector quantization and (B) scalar quantization.

squared error distortion, the centroid is simply the arithmetic average of the vectors in cell V_i, for example

$$\mathbf{y}^* = \frac{1}{||V_i||} \sum_{\ell \in V_i} \mathbf{x}_\ell,$$

where $||V_i||$ denotes the number of vectors in cell V_i.

It can be shown that local optimality can be guaranteed by upholding these conditions, subject to some mild restrictions [1].

3.1 The Linde, Buzo, and Gray Design Algorithm

The necessary conditions for optimality provide the basis for the classic LBG VQ design algorithm. The LBG algorithm is a generalization of the scalar quantization design algorithm introduced by Lloyd, and hence is also often called the generalized Lloyd algorithm or GLA. Interestingly, this algorithm was known earlier in the pattern recognition community as the k-means algorithm.

The steps of the LBG algorithm for the design of an N vector codebook are straightforward and intuitive. The basics of the algorithm are illustrated in Fig. 7. Starting with a large training set (much larger than N), one first selects N initial codevectors as shown in Fig. 7. Initial code vectors can be selected randomly from the training set. There are two basic steps in the algorithm: encoding of the training vectors and computation of the centroids. To begin, we first encode all the training vectors using the initial codebook. This process assigns a subset of the training vectors to each cell defined by the initial code vectors as illustrated by the partitions in Fig. 7B. Next, the centroid is computed for each cell shown in Fig. 7C. The centroids are then used to form an updated codebook. The process then repeats iteratively with a recoding of the training vectors and a new computation of the centroids to update the codebook. Fig. 7D illustrates the centroid locations after a few iterations. Ideally, at each iteration, the average distortion is reduced until convergence.

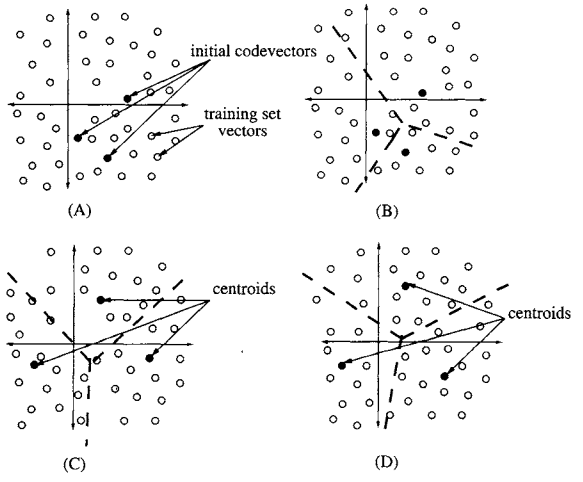


FIGURE 7 Illustration of Linde, Buzo, and Gray (LBG) algorithm.

In practice, convergence is often slow near the point of convergence. Hence in the interest of time, one often terminates the iterative algorithm when the codebook is very close to the local optimum. There are many stopping criteria that can be considered for this purpose. One approach in particular is to compute the average distortion $\mathcal{D}^{(\ell)}$ between the training vectors and the codevectors periodically during the design process, where the superscript ℓ denotes the ℓth iteration. If the normalized difference in distortion from one iteration to the next falls below a prespecified threshold, the design process can be terminated. For example, one could evaluate $\mathcal{D}$ at each iteration and compute the normalized difference

$$\frac{(\mathcal{D}^{(\ell)} - \mathcal{D}^{(\ell-1)})}{\mathcal{D}^{(\ell)}},$$

where forced termination is imposed when this normalized difference becomes less than the stopping threshold.

Often convergence proceeds smoothly. On occasion, the encoding stage of a given iteration may result in one or more cells not being populated by any of the training vectors. This situation, known as the "empty cell" problem, effectively reduces the codebook size. This problem, when detected, can be addressed in any one of a number of ways, one in particular consisting of splitting the cell with the greatest population in two to replace the lost empty cell.

3.2 Other Design Methods

Many methods of VQ design have appeared in the literature in recent years. Some focus on finding a good initial set of codevectors, which are then passed on to a classic LBG algorithm. By starting with a good initial codebook, one not only converges to a good solution, but generally converges in fewer iterations. Randomly selecting the initial codevectors from the training set is the easiest approach. This approach often works well, but sometimes does not provide sufficient diversity to achieve a good locally optimal codebook. A simple variation that can be effective for certain sources is to select the N vectors (as initial code vectors) from the training set that are farthest apart in terms of the distortion measure. This tends to ensure that the initial code vectors are widely distributed in the k-dimensional space.

Alternatively, one can apply the splitting algorithm, which is a data-dependent approach that systematically grows the initial codebook. The method, introduced in the original paper by Linde, Buzo, and Gray [2] and illustrated in Fig. 8, starts with a codebook consisting of the entire training set. First the centroid of the training set is computed as shown in

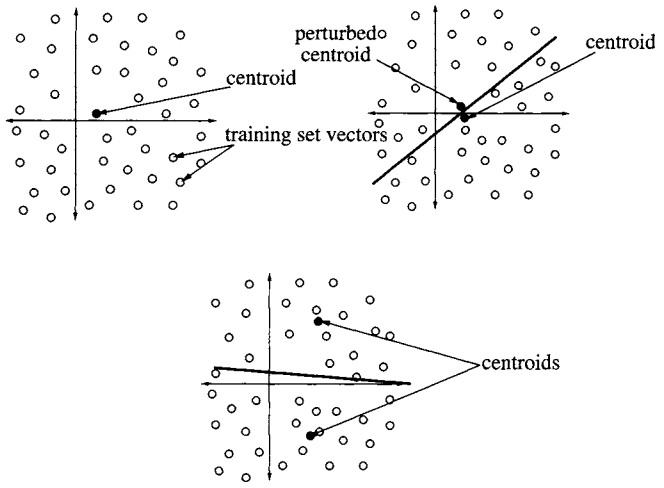


FIGURE 8 Initilization of Linde, Buzo, and Gray (LBG) algorithm using splitting algorithm.

Fig. 8 (top left). This centroid is then split into two codewords by perturbing the elements of the centroid (Fig. 8, top right). For instance, this could be done by adding some small value to each element. The original centroid and the perturbed centroid are used to encode the training set, after which centroids are computed to form a new initial codebook as shown in Fig. 8 (bottom). These new centroids can then be perturbed and used to encode the training set. After centroids are computed, we have four code vectors in the codebook. The process can be repeated until N code vectors are obtained. At this point, the LBG algorithm can be applied as described earlier.

These approaches are intended primarily as a way to obtain initial codebooks for the LBG algorithms. Other methods have been proposed that attempt to find good codebooks directly, which may be optimized further by the LBG algorithm if so desired. One such algorithm in particular is the pairwise nearest neighbor algorithm or PNN [3]. In the PNN algorithm, we start with the training set and systematically merge vectors together until we arrive at a codebook of size N. The idea is to identify pairs of vectors that are closest together in terms of the distortion measure, and replace these two vectors with their mean. This operation reduces the codebook size by effectively merging those partitions that would result in the smallest increase in distortion. In this way the PNN algorithm can be used to merge a large training set into an N vector codebook. As described, this algorithm is computationally demanding. However, a fast implementation may be used, the details of which can be found in an article by Equitz [3].

Codebooks designed by the PNN algorithm can be used directly for VQ or as initial codebooks for the LBG algorithm. It has been observed that using the PNN algorithm as a front end to the LBG algorithm (i.e., in lieu of the random selection or splitting methods) can lead to better locally optimal solutions.

It is impossible to discuss all the design algorithms that have been proposed. However, it is appropriate to mention a few others in closing this section. There are a number of modifications to the LBG algorithm that can lead to an order of magnitude speed up in design time. One approach involves transforming all the training vectors into the discrete cosine transform domain and performing the VQ design in that domain. Because many of the transform coefficients are close to zero and hence can be neglected, codebook design can be performed effectively with lower dimensional vectors. Although there is overhead associated with performing the transform, it is offset by the efficiency of performing iterations in a reduced dimensional space.

Neural nets have also been considered for VQ design. A number of researchers have successfully used neural nets to generate VQ codebooks [5, 6]. Neural net algorithms can have advantages over the classic LBG algorithm, such as less sensitivity to the initialization of the codebook and faster convergence.

The ultimate design algorithm is one that finds the global optimal. Several attempts at this have been reported, such as design by simulated annealing, by stochastic relaxation, and by genetic algorithms [7–10]. Algorithms of this type are perhaps the best in terms of performance, but tend to have high computational complexity. Interestingly, amid all of these choices, the LBG algorithm still remains one of the most popular.

4 Vector Quantization Implementations

VQ is attractive because it has a performance advantage over scalar quantization. However, like all things in life, quality comes with a price. For VQ, that price comes in the form of increased encoder complexity and codebook memory. The number of code vectors that must be stored in a codebook grows exponentially with increasing bit rate. For example, a 16-dimensional VQ at a rate of 0.25 bits per sample requires a codebook of size 16, whereas the same VQ at a rate of 1 bit/sample requires 65,536 code vectors. Codebook memory also grows exponentially with vector dimension. For example, an 8-dimensional VQ at a rate of 1 bit/sample (with 1-byte code vector elements) would occupy 2048 bytes. Increasing the dimension to 32 causes the memory storage requirement to jump to more than 34 Gbytes. Similarly, the same kind of exponential dependence exists for encoder complexity. Unlike scalar quantization, careful attention should be given to dimension and rate, because memory and complexity requirements can easily become prohibitively large. As a general rule, VQs that are used in practice have dimension of 16 or less, because complexity, memory, and performance tradeoffs are generally most attractive in this range.

A host of fast search methods have been reported for VQ that can be grouped into two general types. The first

can be called *fast optimal search* methods, which are optimal in the sense that they guarantee that the encoder will find the best matching codevector for each input vector [1, 11, 12, 34].

One of the simplest methods of this type is known as the *partial distortion* method. Consider the VQ encoder where in the conventional paradigm the input vector $\mathbf{x}$ is compared to each of the code vectors by explicit computation of $d(\mathbf{x}, \mathbf{y}_i)$ for $i = 1, 2, \ldots, N$. The partial distortion method involves keeping track of the lowest distortion calculation to date as the codebook is being searched. To understand how complexity is reduced, assume that we have searched $N/4$ of the code vectors in the codebook and that the minimum distortion found thus far is d[min]. For the next distortion calculation, we compute

$$d[i] = \sum_{\ell=1}^{k} (x[\ell] - y_i[\ell])^2,$$

where $x[\ell]$ and $y_i[\ell]$ are the elements of $\mathbf{x}$ and $\mathbf{y}_i$, respectively. If during the process of evaluating the summation above, the value of $d[i]$ exceeds d[min], then we can terminate the calculation since we know that this vector is no longer a candidate. The net result of applying this procedure for encoding is that many of the vectors will be eliminated from further consideration prior to the full evaluation of the distortion calculation.

In addition, the triangle inequality can be used to reduce complexity, the idea being to use some reference points from which the distance to each code vector is precomputed and stored [11]. The encoder then computes only the distance between the input vector and each reference point. Using these less complex comparisons in conjunction with precomputed data, a reduction in complexity can be achieved. The speed improvement realized by techniques of this type are clearly dependent on the codebook and input source; however, in general, one can expect a modest speed up.

Although every little bit helps, the complexity gains realized by optimal fast-search algorithms fall short of addressing the exponential complexity growth associated with VQ. In this regard, efficient structured VQ encoding algorithms are attractive.

5 Structured Vector Quantization

A class of time-efficient methods has been studied extensively that sacrifice performance for substantial improvement in speed. The approach taken is to impose efficient structural constraints on the VQ codebook. These constraints are often formulated to make encoding complexity and/or memory *linearly* or *quadratically* dependent on the rate and dimension rather than *exponentially* dependent. The price paid, however, is usually inferior performance for the same rate and dimension. Nonetheless, the substantial reduction in complexity usually more than offsets the degradation in performance [32]. To begin, we consider the most popular structured VQ of this class, tree-structured vector quantization (TSVQ).

5.1 Tree-Structured Vector Quantization

TSVQ consists of a hierarchic arrangement of code vectors, that allows the codebook to be searched efficiently. It has the property that search time grows linearly with rate instead of exponentially. Binary trees are often used for TSVQ because they are among the most efficient in terms of complexity. The concept of TSVQ can be illustrated by examining the binary tree shown in Fig. 9. As shown, the TSVQ has a root node at the top of the tree with many paths leading from it to the bottom. The codevectors of the tree,

$$\mathbf{y}_{000}, \mathbf{y}_{001}, \cdots, \mathbf{y}_{111},$$

are represented by the nodes at the bottom. The search path to reach any node (i.e., to find a code vector) is shown explicitly in the tree. In our particular example there are $N = 8$ codebook vectors and $N = 8$ paths in the tree, each leading to a different code vector. To encode an input vector $\mathbf{x}$, we start at the top and move to the bottom of the tree. During that process, we encounter $\nu = 3$ (or $\log_2 N$) decision points (one at each level). The first decision (at level $\nu = 1$) is to determine whether $\mathbf{x}$ is closer to vector $\mathbf{y}_0$ or $\mathbf{y}_1$ by performing a distortion calculation. After a decision is made at the first level, the same procedure is repeated for the next level until we have identified the codeword at the bottom of the tree. For a binary tree, it is apparent that $N = 2^{\nu}$, which means that for a codebook of size N, only $\log_2 N$ decisions have to be made.

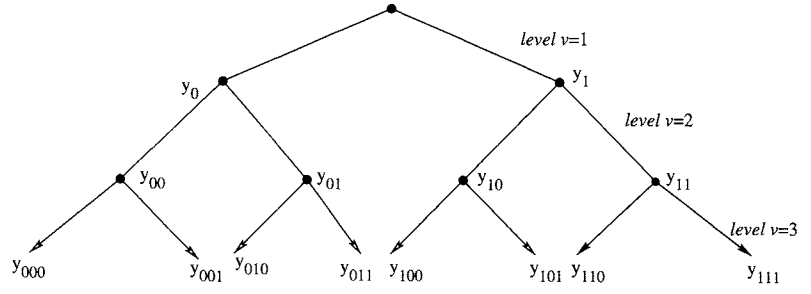


FIGURE 9 Tree-structured vector quantization diagram showing a three-level balance binary tree.

As presented, this implies the computation of two vector distortion calculations, $d(\cdot,\cdot)$, for each level, which results in only $2\log_2 N$ distortion calculations per input vector.

Alternatively, one can perform the decision calculation explicitly in terms of hyperplane partitioning between the intermediate code vectors. The form of this calculation is the inner product between the hyperplane vector and input vector, where the sign of the output (+ or −) determines selection of either the right or left branch in the tree at that node. Implemented this way, only $\log_2 N$ distortion calculations are needed.

For the 8-vector TSVQ example above, this results in three instead of eight vector distortion calculations. For a larger (more realistic) codebook of size $N=256$, the disparity is 8 versus 256, which is quite significant.

TSVQ is a popular example of a constrained quantizer that allows implementation speed to be traded for increased memory and a small loss in performance. In many coding applications, such tradeoffs are often attractive.

6 Mean-Removed Vector Quantization

Mean-removed VQ is another popular example of a structured quantizer that leads to memory-complexity-performance tradeoffs that are often attractive in practice. It is a method for effectively reducing the codebook size by extracting the variation among vectors due specifically to the variation in the mean and coding that extracted component separately as a scalar. The motivation for this approach can be seen by recognizing that a codebook may have many similar vectors differing only in their mean values.

A functional block diagram of mean-removed VQ is shown in Fig. 10. First the mean of the input vector is computed and quantized with conventional scalar quantization. Then the mean-removed input vector is vector quantized in the conventional way using a VQ that was designed with mean-removed training vectors. The outputs of the overall system are the VQ codewords and the mean values.

At the decoder, the mean-removed vectors are obtained by table loop-up. These vectors are then added to a unit amplitude vector scaled by the mean, which in turn restores the mean to the mean-removed vector. This approach is really a hybrid of scalar quantization and VQ. The mean values, which are scalar quantized, effectively reduce the size of the VQ, making the overall system less memory and computation intensive.

One can represent the system as being a conventional VQ with codebook vectors consisting of all possible codewords obtainable by inserting the means in the mean-removed vectors. This representation is generally called a super codebook. The size of such a super codebook is potentially very large, but clearly is also very constrained. Thus, better performance can always be achieved, in general, by using a conventional unconstrained codebook of the same size instead. However, since memory and complexity demands are often costly, mean-removed VQ is attractive.

6.1 Gain-Shape Vector Quantization

Gain-shape VQ is very similar to mean-removed VQ but involves extracting a gain term as the scalar component instead of a mean term. Specifically, the input vectors are decomposed into a scalar *gain* term and a gain normalized vector term, which is commonly called the *shape*. The gain value is the Euclidean norm given by

$$g = ||\mathbf{x}|| = \sqrt{\sum_{i=1}^{k} x^2[i]}, \tag{3}$$

and the shape vector $\mathbf{S}$ is given by

$$\mathbf{S} = \frac{\mathbf{x}}{g}. \tag{4}$$

The gain term is quantized with a scalar quantizer, while the shape vectors are represented by a shape codebook designed specifically for the gain normalized shape vectors.

Perhaps not surprisingly, gain-shape VQ and mean-removed VQ can be combined effectively together to capture the complexity and memory reduction gains of both. Similarly, the implicit VQ could be designed as a TSVQ to achieve further complexity reduction if so desired.

To illustrate the performance of VQ in a printed medium such as a book, it is convenient to use image coding as our application. Comparative examples are shown in Fig. 11.

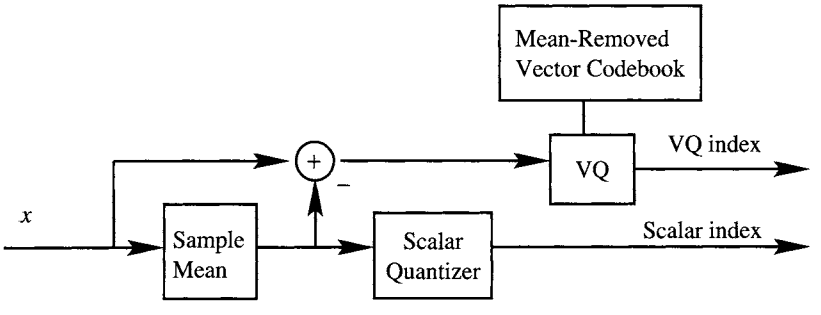
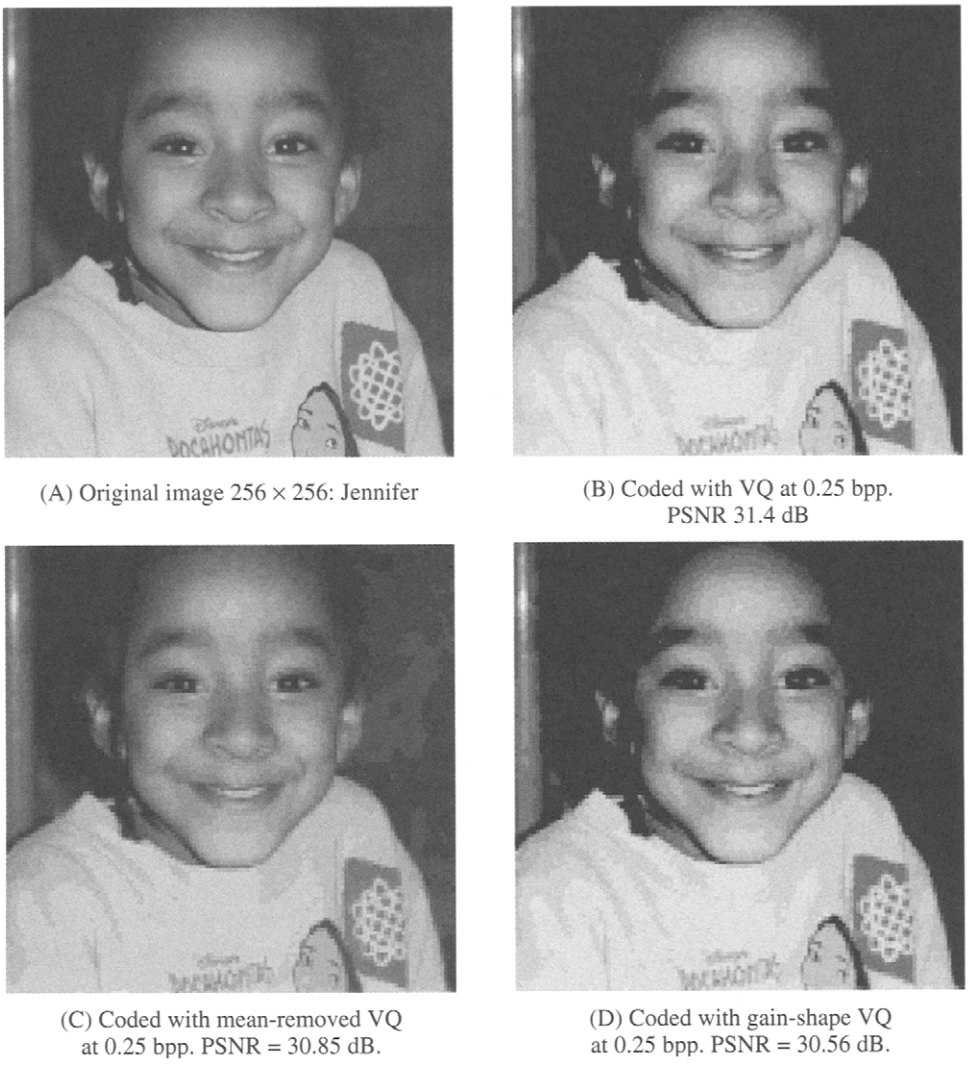


FIGURE 10 Block diagram of mean-removed vector quantization.



(A) Original image 256 × 256: Jennifer

(B) Coded with VQ at 0.25 bpp. PSNR 31.4 dB

(C) Coded with mean-removed VQ at 0.25 bpp. PSNR = 30.85 dB.

(D) Coded with gain-shape VQ at 0.25 bpp. PSNR = 30.56 dB.

FIGURE 11 Comparative illustration of images coded using conventional vector quantization (VQ), mean-removed VQ, and gain-shape VQ. All coded images were coded at 0.25 bits/pixel using 4 × 4 vector blocks. PSNR, peak signal-to-noise ratio.

The image in (A) is an original 8-bit/pixel 256 × 256 monochrome image. The image next to it is the same image coded with convention unstructured 4 × 4 VQ at a rate of 0.25 bits/pixel. The images on the bottom are results of the same coding using mean-removed and gain-shape VQ. From the example, one can observe distortion in all cases at this bit rate. The quality, however, for the unconstrained VQ case is better than that of the structured VQs in Figs. 11C and 11D, both subjectively and in terms of the signal-to-noise ratio (SNR). For quantitative assessment of the quality, we can consider the peak SNR (PSNR) defined as

$$\text{PSNR} = 10\log_{10}\left(\frac{\text{max value} \times [n_1, n_2]^2}{1/(N_1N_2)\sum_{n_1=1}^{N_1}\sum_{n_2=1}^{N_2}(x[n_1,n_2]-\hat{x}[n_1,n_2])^2}\right). \quad (5)$$

Although PSNR can be faulted easily as a good objective measure of quality, it can be useful if used with care. PSNRs are quoted in the examples shown, and confirm the quality advantage of unconstrained VQ over the structured methods shown below it. However, the structured VQs have significantly reduced complexity.

7 Multistage Vector Quantization

A technique that has proven to be valuable for storage and complexity reduction is multistage VQ. This technique is also referred to as *residual* VQ or RVQ. Multistage VQ divides the encoding task into a sequence of cascaded stages. The first stage performs a first-level approximation of the input vector. The approximation is refined by the second-level approximation that occurs in the second stage, and then refined again in

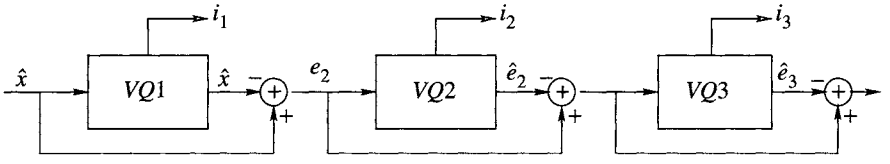


FIGURE 12 Block diagram of a residual vector quantization (VQ), also called multistage VQ.

the third stage, and so on. The series of approximations or successive refinements is achieved by taking stage vector input and subtracting the coded vector from it, producing a residual vector. Thus, multistage VQ is simply a cascade of stage VQs that operate on stage residual vectors. At each stage, additional bits are needed to specify the new stage vector. At the same time, the quality of the representation is improved. A block diagram of a residual VQ is shown in Fig. 12.

There are many ways to design a multistage VQ. Perhaps the most straightforward approach is to design the stages sequentially (i.e., independently, one at a time). For example, we can start with the original training set and use it to construct the first-stage codebook via the LBG algorithm. Then, we can compute first-stage residual vectors for each vector in the training set, resulting in a new training set for the second-stage codebook. In this manner, we can design as many stage codebooks as desired. This method works reasonably well for two-stage systems. However, when the number of stages becomes large, the loss in performance becomes more significant. Better performance can be achieved by designing the stage codebooks jointly. Joint design of multistage VQ was pioneered by Barnes [13]. A detailed treatment of this approach is available [13, 14]. In the remainder of this section, we present an abridged introduction to the joint design problem.

As a start, consider a multistage VQ with P stages and N code vectors in each stage. This multistage VQ can be expressed compactly as $\{(C^p, P^p);\ 1 \le p \le P\}$, where C^p and P^p denote the codebook and partitions respectively for the *pth* stage. The associated codevectors and Voronoi cells are given by

$$\{y_0^p, y_1^p, y_2^p, \ldots, y_{N-1}^p\}$$

and

$$\{S_0^p, S_1^p, S_2^p, \ldots, S_{N-1}^p\}.$$

The code vectors comprising the codebook C^p and the cells comprising the partition P^p are indexed with the subscript j^p. The quantized representation $\hat{\mathbf{x}}$ of the input source vector $\mathbf{x}$ is formed by the sum of the selected stage code vectors,

$$\hat{\mathbf{x}} = \sum_{\mathbf{p}=\mathbf{1}}^{\mathbf{P}} \mathbf{y}_{\mathbf{j}^\mathbf{p}}^{\mathbf{p}}. \tag{6}$$

Rather than picture the multistage VQ as a cascade of independent stages, it is useful for our purpose to picture the system as an equivalent VQ of the form shown in Fig. 2. Such an equivalent VQ, which we denote (C^e, P^e), is called a *direct sum* quantizer. The name is derived from the fact that the elements of the equivalent codebook C^e are obtained by summing the selected code vectors across all stages. When one considers all possible code vectors that can be reconstructed by traversing from the first stage to the last, we see that $C^e = C^1 + C^2 + \cdots + C^P$. A joint design algorithm can thus be built around this property by minimizing the distortion associated with the direct sum codebook C^e. To do this one can start by designing the first-stage codebook while holding the other codebooks fixed, and then design the second-stage codebook, holding the others fixed, and so on.

The direct sum structure of multistage VQ can be viewed as a tree where each path through the tree corresponds to a different selection of code vectors from each stage. This notion of a tree is a very intuitive way of understanding multistage VQ.

As an illustration, consider the multistage VQ tree with three stages and two code vectors/stage shown in Fig. 13A. The root node of the tree represents the original input $\mathbf{x}$. The leaf nodes at the bottom represent the direct sum codevectors, C^e. The intermediate nodes represent partial sums of the direct sum codevectors and the branches represent stage codevectors.

There is a hidden challenge associated with multistage VQ design of this form. Although we design the stage codebooks jointly, we are at this point searching the direct sum codebook sequentially, which leads to underperformance. This inefficiency is illustrated pictorially in Fig. 13. The branches of the tree for a jointly designed multistage VQ are typically entangled. Thus, if one is constrained to encode an input vector by sequentially proceeding down the stages of the multistage VQ, one may arrive at a direct sum codevector (leaf node) that is *not* the best matching vector in C^e. To reach the best matching code vector in an entangled tree may often require choosing a nonminimum distortion vector at a particular stage. That is to say, the path down the tree required to reach the minimum distortion direct sum vector may be one that requires choosing a nonminimum distortion stage vector along the way.

There are two general approaches to addressing the entanglement challenge. One is to use entanglement-permissive encoders. These are encoders that make use of multipath

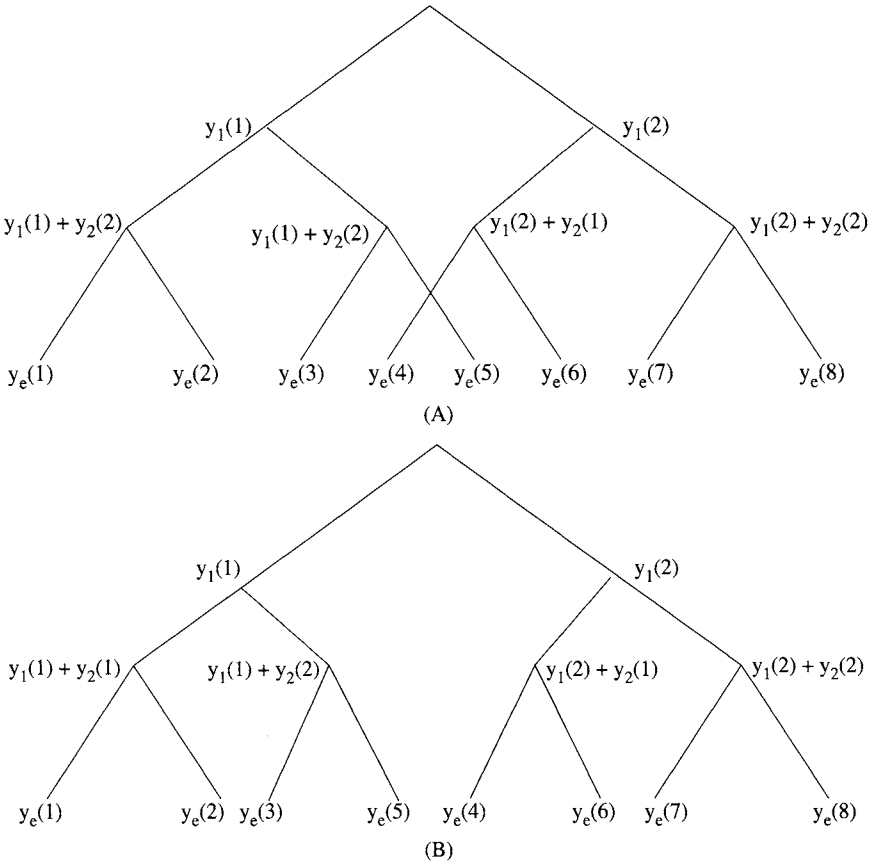


FIGURE 13 Example of multistage vector quantization (VQ) tree for three-stage VQ with two code vectors per stage.

searching (or M-search) algorithms to ensure that the optimal or near-optimal direct sum codevector is found [21]. M-search can be used in the encoding process and in the design process to achieve improved performance. Obviously, there is a computational cost incurred with M-search algorithms. By selecting the number of alternative search paths allowed, one can trade computational encoding complexity for performance.

The other general approach to managing entanglements is to impose additional structure on the RVQ to force the corresponding tree to be unentangled [23]. While the imposed structure does reduce performance compared to RVQ M-search, it can provide an attractive cost effective compromise.

The most dramatic advantage of multistage VQ comes from its savings in memory and complexity, which for large VQs can be orders of magnitude less than that of unconstrained VQ. In addition, multistage VQ has the property that it allows the bit rate to be controlled simply by specifying the number of VQ stage indices to be transmitted. Overall, it can be a very attractive method for VQ implementation.

8 Trellis-Coded Vector Quantization

Trellis-coded quantization (TCQ) is well known as an effective source-coding technique. The gains associated with TCQ over scalar quantization are largely attributed to capturing cell shape gain. In this section we discuss the vector extension of TCQ, which is called trellis-coded vector quantization or TCVQ.

TCVQ, like TCQ, operates on a *delayed decision* principle. In conventional scalar or vector quantization, each quantized symbol is fully determined by the current input symbol. This allows for immediate decoding at any point or time in the coded bit stream. For delayed decision coders, choices made in quantizing input samples affect future decisions taken by the coder. Consequently, we typically evaluate several good quantization choices for samples and choose the ones that lead to lower distortion up to that particular moment. The performance of such a delayed decision coder improves with increased delay.

TCVQs can be depicted conveniently by a trellis diagram, an example of which is shown in Fig. 14. The nodes of the trellis correspond to a state and the branches correspond to state transitions from one state to the next. As implied in the figure, the trellis is a cascade of sections, where each trellis section is associated with an input vector. During the encoding process each branch is attached to the distortion produced by replacing the input vector with the quantized codevector from the codebook labeling that branch. The TCVQ encoder finds the path through the trellis having

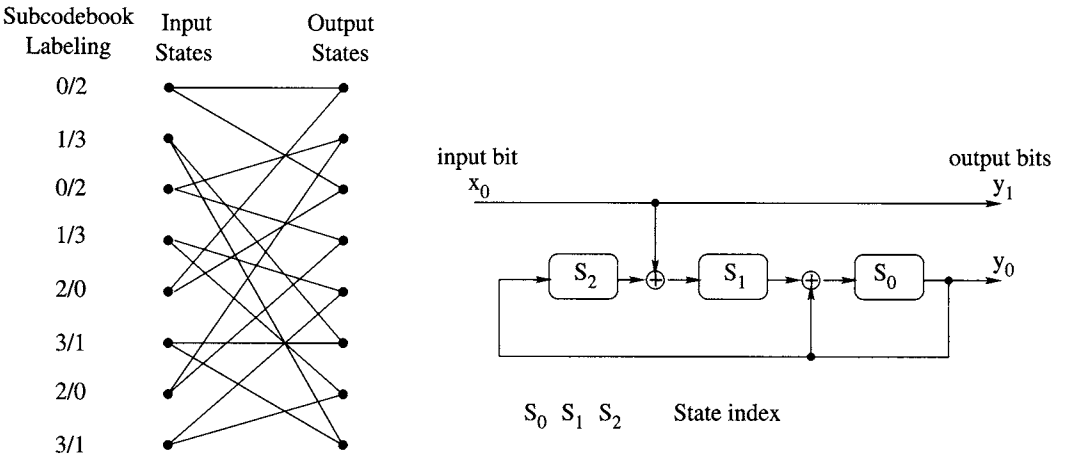


FIGURE 14 On the left, a section of Ungerboeck's eight-state amplitude modulation trellis. The branches are labeled here with subcodebooks. On the right, the corresponding convolution encoder.

minimum overall distortion (defined as the sum of the distortions of the branches composing that path). The key features of the TCVQ design are the selection of the trellis structure, the labeling of the trellis branches, the partitioning of the initial codebook, and the coding delay. We will elaborate on each of these features in the following paragraphs.

For the most part, TCVQ systems use modulation trellises in their construction [17]. The underlying finite-state machine is a $\frac{1}{2}$ binary convolutional encoder with each state having two input and two output branches. Trellis branches are labeled with vector codebooks. A path through the trellis corresponds to a concatenation of codevectors. A section of an eight-state amplitude modulation trellis and its corresponding convolutional encoder is depicted in Fig. 14.

The m bits available for quantizing each input vector are used as follows. One bit, called the transition bit, is input to the convolutional encoder, to determine the state transition which is represented by the trellis branch. There are four smaller codebooks used in labeling the trellis branches. The convolutional encoder outputs two bits that select the vector codebook labeling that branch. The remaining $m-1$ bits, called selection bits, specify the code vector within the given subcodebook. This means that the expanded codebook (the union of four subcodebooks that label the trellis branches) contains $4 \times 2^{m-1} = 2^{m+1}$ entries, twice the number of codevectors of the VQ codebook for the same bit rate.

In order to limit the coding delay, the code vectors must be periodically (let us say after P input vectors) sent to the decoder. This is accomplished in the following way. After the processing of vector $x[n]$, where n is the time index, suppose the least distortion path is $O[n+1]$. The indices of code vectors for input vectors from $x[n-D-P+1]$ to $x[n-D]$ from the path $O[n+1]$ are sent to the decoder. D is called the decision delay. The paths ending in a state different from the best path state at time $n-D$ are eliminated. In theory, performance improves the longer we make the delay D. However, in practice there is often a limit on acceptable delay and thus D is typically selected pragmatically.

The trellis branch labeling for the TCVQ typically employs the labeling rules attributed to Ungerboeck [25]. These rules are meant to maximize the minimum Euclidean distance between distinct trellis code vector sequences. The minimum Euclidean distances are maximized by discarding the trellis paths with similar codevector sequences.

Since the labeling rules makes use of the smaller subcodebooks, one needs to partition the expanded codebook. Ungerboeck's set partitioning used in trellis-coded modulation starts with an expanded codebook and then successively divides it into smaller subcodebooks while *maximally increasing* the intra-subcodebook distances. Because the optimized expanded vector codebook does not have a structure, its partitioning is not a trivial task. Fortunately there are several good solutions [26].

Given an initial common VQ codebook of size $M = 2^{kR+1}$, where R represents the bits per sample and k is the vector dimension, the Euclidean distances between all possible pairs of codevectors are calculated and listed in nondecreasing order along with the corresponding pairs. The partitioning algorithm repeats two basic steps. The first step is to look for the entry with the smallest distance code vector pair where one and only one of the code vectors in the pair is already assigned to either of the two subcodebooks. The next step is to add the unassigned code vector of the pair, selected in the first step, to the other subcodebook. The algorithm to partition a codebook A into two subcodebooks A_0 and A_1, can be described as follows:

1. Design an initial VQ codebook of size $M = 2^{kR+1}$ using the LBG algorithm. The Euclidean distances between all possible pairs of codevectors are calculated and listed in non-decreasing order. This provides a table of size

$M(M-1)/2$ entries, where the ith entry corresponds to code vectors c_i and $\acute{c}_i$ that are at distance $d_i = \|c_i - \acute{c}_i\|$.

2. Assign c_0 to A_0 (or A_1) and $\acute{c}_0$ to A_1 (or A_0), and remove the first entry from the table.
3. Search the table to find an index j such that $\forall i < j$, neither c_i nor $\acute{c}_i$ belongs to $A_0 \cup A_1$, but at least one of c_j and $\acute{c}_j$ belongs to $A_0 \cup A_1$.
4. If both c_j and $\acute{c}_j$ belongs to $A_0 \cup A_1$ (i.e., they are both already assigned), then remove this particular entry from the table and go back to Step 3.
5. If c_j belongs to A_0 (or A_1), then add $\acute{c}_j$ to A_1 (or A_0). If $\acute{c}_j$ belongs to A_0 (or A_1), then add c_j to A_1 (or A_0).
6. If the size of A_0 (or A_1) reaches $M/2$, then add the remaining unassigned codevectors (if any) to A_1 (or A_0) and stop. Otherwise, go to Step 2.

To partition the overall VQ codebook into four subcodebooks, the algorithm is first applied to partition the initial codebook of size 2^{kR+1} into two subcodebooks of size 2^{kR}, followed by applying it to those two subcodebooks to generate four subcodebooks of size 2^{kR-1}.

TCVQ has been studied quite extensively for uniform density sources and has proven to be an effective coding technique.

8.1 Predictive Vector Quantization

Many signals and sources have inherent redundancy, which can make them well suited to predictive modeling. The concept of *prediction* is a powerful principle that is well established in signal and image processing. It allows such redundant inputs to be represented efficiently. The most common form of prediction seen in the literature is linear prediction, where a sample is expressed as the weighted sum of previous samples. Mathematically, linear prediction takes the form

$$\tilde{x}[n] = \sum_{i=1}^{N} \alpha_i x[n-i],$$

where $\tilde{x}[n]$ is the predicted sample at time n and the α_is are the weights. But, of course, there are other types of predictors where previous samples can be used to estimate the current sample. Perhaps not surprisingly, this well-known principle of prediction can be extended to VQ in a straightforward way, resulting in what is known as *predictive VQ* or *PVQ*. A diagrammatic depiction of PVQ is shown in Fig. 15. The general predictive structure shown in this figure is common to many predictive systems, such as differential pulse

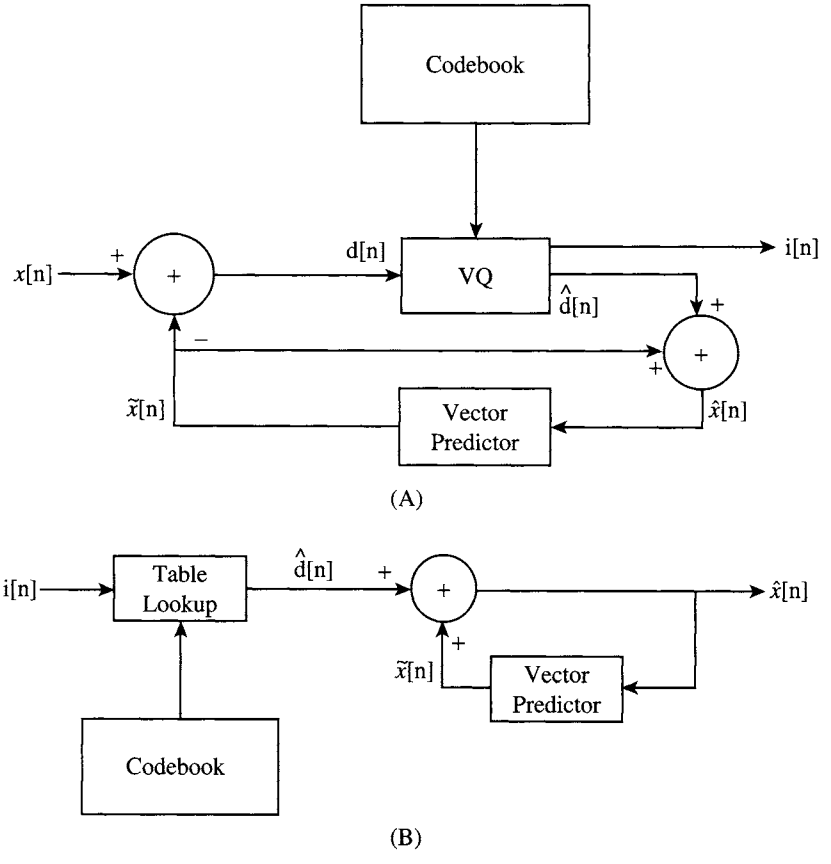


FIGURE 15 Predictive vector quantizer: (A) Encoder. (B) Decoder.

code modulation (DPCM), linear predictive coding (LPC), JPEG coders, MPEG coders, and a host of others. However, for PVQ the inputs, outputs, and quantizers involve vectors instead of scalars [19].

PVQ is based on the predictor $\tilde{\mathbf{x}}[n] = P(\hat{\mathbf{x}}[n])$, where P denotes the prediction operator (typically based on a finite number of previously available vectors), the tilde ($\tilde{}$) signifies the predicted signal vector, and the ($\hat{}$) signifies the quantized vector. As shown in Fig. 15, the input to the VQ is the vector $\mathbf{d}[n]$, which is the differential (or residual) vector

$$\mathbf{d}[n] = \mathbf{x}[n] - \tilde{\mathbf{x}}[n].$$

Assuming that the predictor performs a reasonable job of representing the input, coding $\mathbf{d}[n]$ is more efficient than coding $\mathbf{x}[n]$.

A key issue in PVQ is the design of the system. It turns out that for PVQ we can use the same VQ codebook design algorithms discussed earlier. Perhaps the simplest and most direct approach is to first design the predictors from a training set of vectors $\mathbf{x}[\mathbf{n}]$ and then design the VQ using a training set of residual vectors $\mathbf{d}[\mathbf{n}]$. This method allows one to use the LBG design principles to create a PVQ in short order. However, one recognizes that better results should be possible if the predictor and VQ designs are based on quantized training data rather than unquantized data. Such a design can be performed by creating an outer loop where the predictors and VQ are systematically updated. In other words, we can start with the predictors and a VQ designed as initially described. Then using these results compute a coded vector $\hat{\mathbf{x}}[\mathrm{n}]$ for use in designing an updated set of vector predictors. Those predictors in turn can be used to generate an updated residual training set $\mathbf{d}[\mathbf{n}]$. This process of successive updating can be repeated until acceptable convergence is obtained. Both the simple unquantized-based and more complex quantized-based approaches can lead to good performance.

PVQ can often provide an effective way to exploit redundancy in a signal. Such redundancy could be exploited using high dimensional vectors. However, the computational complexity of the encoding process grows exponentially with increasing vector size, which makes high dimension VQ unattractive. PVQ, on the other hand, can operate with relatively small vectors (e.g., 2×2 and 4×4) and hence can have a significant performance-complexity advantage over high dimension VQ.

8.2 Variable-Rate Vector Quantization

Up to this point our discussion has concentrated on the structure, implementation, and design of the quantizers. Each codevector was assumed to have associated with it an index (codeword) comprised of B bits. Thus, for a codebook of size N,

$$B = \log_2 N \quad \text{or} \quad N = 2^B. \tag{7}$$

Consequently, a conventional k-dimensional VQ with N codevector would have a bit rate R given by

$$R = \frac{1}{k}\log_2 N. \tag{8}$$

In addition to optimizing the codebooks to minimize distortion, one can also optimize the assignment of indices or codewords to exploit the statistics of the input. That is to say, gain in rate performance can be achieved by allowing the codewords to have different lengths, some long, some short. Variable rate coding schemes of this type are based on the notion that code vectors that are selected infrequently on average are assigned longer indices, while code vectors that are used frequently are assigned shorter length indices. Making the index assignments in this way is called *entropy coding*. For nonuniformly distributed sources, entropy coding results in a lower average bit rate and thus makes coding more efficient. Applying entropy coding to the codebook indices can be done in a straightforward way. One only needs estimates of the code vector probabilities $P(i)$. With these estimates, methods like Huffman coding will assign to the ith index a codeword whose length L_i is approximately $-\log_2 P(i)$ bits [24].

We can improve upon this approach by designing the VQ and the entropy coder together. This approach is called entropy-constrained VQ or ECVQ [28]. ECVQs can be designed by a modified LBG algorithm. Instead of finding the minimum distortion $d(\mathbf{x}, \mathbf{y_i})$ in the LBG iteration, one finds the minimum modified distortion

$$J_i = d(\mathbf{x}, \mathbf{y_i}) + \lambda L_i,$$

where $L_i = -\log_2 P(i)$. Using this modified distortion J_i, which is a Lagrangian cost function, effectively enacts a Lagrangian minimization that seeks the minimum weighted cost of quantization error and bits.

To achieve rate control flexibility, one can design a set of codebooks corresponding to a discrete set of λs, which gives a set of VQs with a multiplicity of bit rates. The concept of ECVQ is powerful and can lead to performance gains in data compression systems. It can also be applied in conjunction with other structured VQs such as mean-removed VQ, gain-shape VQ, and residual VQ—the last of these being particularly interesting. Entropy constrained residual VQ or EC-RVQ and variations of it have proven to be among the most effective VQ methods for direct application to image compression. Schemes of this type involve the use of

conditional probabilities in the entropy coding block where conditioning is performed on the previous stages and/or on adjacent stage vector blocks. Like ECVQ, the design is based on a Lagrangian cost function, but integrated into the RVQ design procedure. In the design algorithm reported by Kossentini and colleagues [14, 15], both the VQ stage codebooks and entropy coders are jointly optimized iteratively.

9 Closing Remarks

VQ can achieve significant coding gains over scalar quantization, but at a price. For conventional VQ, codebook memory and encoding complexity are exponentially related to the vector dimension and bit rate, which is evident from equations (7) and (8). A variety of structured VQs (like multistage VQ, PVQ, and mean-removed VQ) can be considered to trade a little bit of this performance advantage for a very large reduction in memory and complexity.

The concepts of optimality, partitioning, and distortion that we discussed in the context of VQ are insightful, and continue to inspire new contributions in the technical literature, particularly with respect to achieving useful trade-offs among memory, complexity, and performance. Equally important to a good understanding of this subject matter are the design methodologies and concepts of variable length encoding for efficient compression. Although we have attempted to touch on the basics, the reader should be aware that the VQ topic area embodies much more than can be covered in a concise tutorial chapter. Thus, in closing, it is appropriate to at least mention several other classes of VQ that have received attention in recent years. First, is the class of lattice VQs [22, 30, 33, 35]. Lattice VQs can be viewed as vector extensions of uniform quantizers, in the sense that the cells of a k-dimensional lattice VQ form a uniform tiling of the k-dimensional space. Searching such a codebook is highly efficient. The advantage achieved over scalar quantization is the ability of the lattice VQ to capture cell shape gain. The disadvantage of course is that cells are constrained to be uniform. Nonetheless, lattice VQ can be attractive in many practical systems.

Second, is the general class of predictive-based VQs, which may include finite-state VQ (FSVQ), predictive VQ, vector predictive VQ and several others [25]. Some of these predictive approaches involve using neighboring vectors to define a state unambiguously at the encoder and decoder and then employing a specially designed codebook for that state. VQs of this type can exploit statistical dependencies (both linear and nonlinear) among adjacent vectors, but have the disadvantage of being memory intensive.

Finally, it should be evident that VQ can be applied to virtually any lossy compression scheme. Prominent examples of this are transform VQ, where the output of a linear block transform such as the DCT is quantized with VQ; and subband VQ, where VQ is applied to the output of an analysis filter bank. Interestingly, the principles of VQ extend far beyond our data compression – oriented discussion. These principles are also applicable to problems in segmentation, classification, and recognition [27]. The inquisitive reader is encouraged to explore this rich area of information theory in the literature [16, 26, 29].

References

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

[2] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.* C-28, 84–95 (1980).

[3] W. Equitz, "A new vector quantization clustering algorithm," *IEEE Trans. Acoust. Speech Sig. Process.* 37, 1568–1575 (1989).

[4] R. King and N. Nasrabadi, "Image coding using vector quantization in the transform domain," *Pattern Recog. Letters* 1, 323–329 (1983).

[5] N. Nasrabadi and Y. Feng, "Vector quantization of images based upon the kohonen self-organizing feature map," in *IEEE Int. Conf. on neural Networks*, Vol. 1, (San Diego, CA), 101–105, 1988.

[6] J. McAuliffe, L. Atlas, , and C. Rivera, "A comparison of the lbg algorithm and kohonen neural network paradigm for image vector quantization," in *IEEE Int. Conf. Acoust. Speech., and Signal Process. '90*, (Alburquerque, NM), pp. 2293–2296, April 1990.

[7] J. Vaisey and A. Gersho, "Simulated annealing and codebook design," in *Int. Conf. on Acoustics, Speech and Signal Processing* (New York), pp. 1176–1179, April 1988.

[8] K. Zeger and A. Gersho, "A stochastic relaxation algorithm for improved vector quantizer design," *Electronics Letters* 25, 896–898 (1989).

[9] K. Zeger, J. Vaisey, and A. Gersho, "Globally optimal vector quantizer design by stochastic relaxation," *IEEE Trans. Signal Process.* 40, 2, 310–322, Feb. 1992.

[10] K. Krishna, K. Ramakrishna, and M. Thathachar, "Vector quantization using genetic k-means algorithm for image compression," in *Proc. of the (1997) international Conf. on Inform., Comm. and Signal Proce. ICICS, Part 3*, Vol. 3 (1997), pp. 1585–1587.

[11] M. Soleymani and S. Morgera, "An efficient nearest neighbor search method," *IEEE Trans. Commun.* COM-35, 677–679 (1987).

[12] D. Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham, "Fast search algorithms for vector quantization and pattern matching," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, (San Diego), pp. 911.1–911.4, March 1984.

[13] C. Barnes, *Residual Quantizers*. PhD thesis, Brigham Young Univ., Provo, UT, Dec 1989.

[14] F. Kossentini, M. Smith, and C. Barnes, "Necessary Conditions for the optimality of variable rate residual vector quantizers," *IEEE Trans. Inf. Theory* 1903–1915 (1995).

[15] F. Kossentini, W. Chung, and M. Smith, "Conditional entropy-constrained residual VQ with application to image coding," *Trans. Image Process. Special Issue on VQ* 311–321 (Feb. 1996).

[16] M. Barlaud, P. A. Chou, N. M. Nasrabadi, D. Neuhoff, M. Smith, and J. Woods, Guest Editors. *IEEE Transactions on Image Processing, Special Issue on Vector Quantization*, 5, 2, 202–206, February 1996.

[17] G. Ungerboeck, "Trellis-coded modulation with redundant signal sets, 2: State of the art," *IEEE Communication Magazine* 25, 12–21 (1987).

[18] M. Wang, and T. R. Fischer, "Trellis-coded quantization design for noisy channels," *IEEE Trans. Inf. Theory*, 40, 1790–1802 (1994).

[19] V. Cuperman, and A. Gersho, "Adaptive differential vector coding of speech," in *Conference Record GlobeCom*, ***82***, 1092–1096 (1982).

[20] H. Khalil, and K. Rose, "Predictive multistage vector quantization design using asymptotic closed-loop optimization," *IEEE Trans. Image Process.* 10, 1765–1770 (2001).

[21] F. Jelinek, and J. B. Anderson, "Instrumental tree encoding of information sources," *IEEE Trans. Inf. Theory*, 17, 118–119 (1971).

[22] J. Pan, and T. R. Fischer, "Two-stage vector quantization-lattice vector quantization," *IEEE Trans. Inf. Theory*, 41, 155–163 (1995).

[23] W. A. H. Mousa and M. A. U. Khan, "Design and analysis of entropy-constrained reflected residual vector quantization," in *Proceedings of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Vol. 3 (2002), pp. 2529–2532.

[24] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Entropy-constrained vector quantization," *IEEE Trans. Acoust Speech Sign. Process.* XX, 31–42 (1999).

[25] R. Aravind and A. Gersho, "Low-rate image coding with finite-state vector quantization," in *Proceedings of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, (Tokyo, 1986), pp. 137–140.

[26] K. Mukerjee, and A. Mukerjee, "Joint optical flow motion-compensation and video compression using hybrid vector quantization," in *Proceedings of (1999) Data Compression Conference* (Snowbird, Utah, March 1999), 541.

[27] J. Li, "A source coding approach to classification by vector quantization and the principle of minimum description length," in *Proceedings of the Data Compression Conferences, (DCC-2002)*, 382–391, April 2002.

[28] J. Pan and T. R. Fischer, "*Two-stage vector quantization-lattice vector quantization*," IEEE Transactions on Information Theory, *vol. 41, pp. 155–163, January 1995.*

[29] M. A. U. Khan, and M. J. T. Smith, and S. W. McLaughlin, "Trellis-coded residual vector quantization: its geometrical advantages and application to image coding," in *Proceedings of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Vol. 5, (2000), pp. 2617–2620.

[30] V. K. Goyal, and J. A. Kelner, and J. Kovacevic, "Multiple-description vector qunatization with a coarse lattice," *IEEE Trans. Inf. Theory*, 48, 781–788 (2002).

[31] A. Orlitsky, "Scalar versus vector quantization: worst case analysis," *IEEE Trans. Inf. Theory*, 48, 1393–1409 (2002).

[32] A. O. Etemoglu, and V. Cuperman, "Structured vector qunatization using linear transforms," *IEEE Trans. Sign. Proces.* 15, 1625–1631 (2003).

[33] D. Mukherjee, S. K. Mitra, "Successive refinement lattice vector quantization," *IEEE Trans. Image Proces.* 11, 1337–1348 (2002).

[34] C. A. Kam-fai, W. Kam-Tim, and K. Chi-Wah, "Vector quantization fast search algorithm using hyperplane based k-dimensional multi-node search tree," *IEEE International Symposium on Circuits and Systems*, Vol. 1 (2002), pp. 793–796, .

[35] J. Pan, "Extension of two-stage vector quantization-lattice vector quantization," *IEEE Trans. Commun.* 45, 1538–1547 (1997).

[36] A. Ebrahim-Moghadam and S. Shirani, "Image foveation on vector quantization," in *Proceedings of Data Compression Conference*, (March 2003), p. 426.

[37] J. Perez-Cordoba, A. Peinado, V. Sanchez, and A. Rubio, "A Study of joint source channel coding of LSP parameters for wideband speech coding," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2 (2003), 189–192.

[38] G. Shen, B. Zeng, and M. L. Lion, "Adaptive vector quantization with codebook updating based on locality and history," *IEEE Trans. Image Proces.* 12, 283–295 (2003).

[39] P. Y. Chen, "An efficient prediction algorithm for image vector quantization," *IEEE Trans. Man Cybernet.* 34, 740–746 (2004).

[40] K. Demirciler and A. Ortega, "Image coding based on multiple projections and multistage vector quantization," in *Proceedings of the International Conference on Image Processing*, Vol. 2, (2003) pp. 287–290.

[41] V. Krishnan, D. V. Anderson, and K. K. Truong, "Optimal multistage vector quantization of LPC parameters over noisy channels," *IEEE Trans. Speech and Audio Process.* 12, 1–8, (2004).