Available online at www.sciencedirect.com
ScienceDirect
Digital Signal Processing 17 (2007) 138–171
Digital Signal Processing
www.elsevier.com/locate/dsp

Efficient product code vector quantisation using the switched split vector quantiser

Stephen So [a,*], Kuldip K. Paliwal [b]

[a] *School of Engineering, Griffith University, PMB 50 Gold Coast Mail Centre, Gold Coast, QLD 9726, Australia*
[b] *School of Microelectronic Engineering, Griffith University, Brisbane, QLD 4111, Australia*

Available online 15 November 2005

Abstract

In this article, we first review the vector quantiser and discuss its well-known advantages over the scalar quantiser, namely the space-filling advantage, the shape advantage, and the memory advantage. It is important to understand why vector quantisers always perform better than any other quantisation scheme for a given dimension, as this will provide the basis for our investigation on improving product code vector quantisers which, despite having much lower computational and memory requirements, result in suboptimal quantisation performance. The main focus is on improving the efficiency of the split vector quantiser (SVQ), in terms of computational complexity and rate-distortion performance. Though SVQ has lower computational and memory requirements than those of the unconstrained vector quantiser, the vector splitting process adds numerous constraints to the codebook, which results in suboptimal quantisation performance. Specifically, the reduced dimensionality affects all three vector quantiser advantages. Therefore, we investigate a new type of hybrid vector quantiser, called the switched split vector quantiser (SSVQ), that addresses the memory and shape suboptimality of SVQ, leading to better quantisation performance. In addition, the SSVQ has lower computational complexity than the SVQ, at the expense of higher memory requirements for storing the codebooks. We evaluate the performance of SSVQ in LPC parameter quantisation, used in narrowband CELP speech coders, and compare it against other quantisation schemes.

Keywords: Vector quantisation; Product code vector quantisers; LPC-based speech coding; Split vector quantisers; Vector quantiser advantages; Line spectral frequencies

1. Introduction

Shannon [1] showed that quantising a vector of data points is more efficient than quantising individual scalar values, in the rate-distortion sense. According to asymptotic Shannon theory, when the dimension of the vectors is arbitrarily large, the operational rate-distortion function of the vector quantiser can approach the Shannon limit [2]. Further studies (such as [3]) have shown that even for a small, finite number of dimensions, the vector quantiser is indeed optimal. Therefore, the vector quantiser can achieve the lowest distortion of any quantisation scheme for a given bitrate, or conversely, require the least number of bits to quantise at a fixed distortion. With the availability of

* Corresponding author. Fax: +61 7 5552 8065.
E-mail addresses: s.so@griffith.edu.au (S. So), k.paliwal@griffith.edu.au (K.K. Paliwal).

1051-2004/$ – see front matter
doi:10.1016/j.dsp.2005.08.005

non-variational design algorithms such as the LBG algorithm [4], vector quantisation has become a powerful tool and its application has been frequently reported in the speech and image coding literature [5–12]. Despite its theoretical rate-distortion optimality, the vector quantiser has its own inherent limitations, specifically the computational and memory requirements for codebook searching and storage, respectively. There are many applications which require high bitrates and/or large dimensionality and this leads to an exponential increase in the complexity of the vector quantiser. These complexity issues can be alleviated by using product code vector quantisers [12], which divide the quantisation process into independent stages. However, the structural constraints imposed by product codes result in a loss of quantisation performance (in terms of bitrate and distortion), hence there is a trade-off with the computational complexity and memory requirements.

The aim of this article is to first provide a general review of basic vector quantisation theory from the coding literature, which includes their design (Section 2.1), and more importantly their advantages over scalar quantisation (Section 2.2). The practical limitations, with regards to computational complexity and memory requirements as a function of bitrate and dimensionality, are then covered in Section 2.3. Following this, product code vector quantisation schemes are described in Section 2.4. The second part of the paper covers a new hybrid vector quantiser called the switched split vector quantiser (SSVQ), introduced in [13], where in Section 3, we describe in detail, its operation and how it compensates for the suboptimality of the split vector quantiser. We evaluate the performance of the SSVQ in LPC parameter quantisation for narrowband speech coding in Section 4 and compare the spectral distortion performance, computational complexity, and memory requirements of SSVQ with other quantisation schemes that have been reported in the literature. The results show that the SSVQ is a computationally efficient quantisation scheme that achieves lower spectral distortion at the expense of an increase in memory requirements.

2. Review of vector quantisation

The reader may be interested in the following articles and textbooks, which cover the general aspects of vector quantisation and provide further references to its applications and performance [2–4,7,10,14]. Vector quantisation (VQ) can be viewed as a generalisation of scalar quantisation where, instead of mapping scalar values to a finite set of reproduction scalars, it maps vectors to a finite set of reproduction code-vectors. The basic definition of a vector quantiser Q of dimension n and size K is a mapping of a vector from n-dimensional Euclidean space, $\mathcal{R}^n$, to a finite set, $\mathcal{C}$, containing K reproduction *code-vectors* [14]:

$$Q : \mathcal{R}^n \to \mathcal{C}, \tag{1}$$

where $\mathcal{C} = \{\boldsymbol{y}_i : i \in \mathcal{I}\}$ and $\boldsymbol{y}_i \in \mathcal{R}^n$ [14]. Associated with each reproduction code-vector is a partition of $\mathcal{R}^n$, called a *region* or *cell*, $\mathcal{S} = \{S_i : i \in \mathcal{I}\}$ [2]. The most popular form of vector quantiser is the Voronoi or nearest neighbour vector quantiser [14], where for each input source vector, $\boldsymbol{x}$, a search is done through the entire codebook to find the nearest code-vector, $\boldsymbol{y}_i$, which has the minimum distance [15]:

$$\boldsymbol{y}_i = Q[\boldsymbol{x}] \quad \text{if } d(\boldsymbol{x}, \boldsymbol{y}_i) < d(\boldsymbol{x}, \boldsymbol{y}_j) \quad \text{for all } i \neq j, \tag{2}$$

where $d(\boldsymbol{x}, \boldsymbol{y})$ is a distance measure between the vectors, $\boldsymbol{x}$ and $\boldsymbol{y}$. The regions in a nearest neighbour vector quantiser are also called Dirichlet or Voronoi regions [3] and these are shown in Fig. 1, where the mean squared error (MSE) is used as the distance measure. Depending on the coding application, other more meaningful distance measures may be used such as the Mahalanobis distance [16], Itakura–Saito distance [17], or other perceptually-weighted distance measures [10].

If the dimension of the vectors is n and a codebook of K code-vectors is used, each vector will be represented as a binary code of length $\lceil \log_2 K \rceil$ bits. Hence the bitrate of the vector quantiser is given by $\frac{1}{n}\lceil \log_2 K \rceil$ bits/sample [15].

2.1. Vector quantiser design using the Linde–Buzo–Gray algorithm

The Lloyd conditions for optimality form the basis of the methods of Lloyd [18] and Max [19] for designing minimum distortion non-uniform scalar quantisers. These conditions are stated as follows [18]:

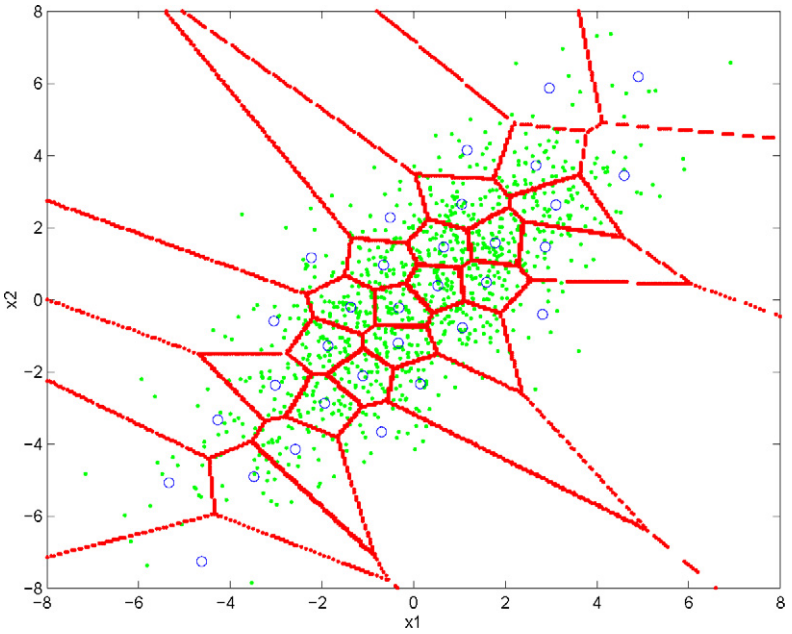

Fig. 1. Voronoi regions of a nearest neighbour (MSE) vector quantiser, showing the input vectors (dots), code-vectors (circles), and hyperplanes defining each Voronoi region (lines).

1. the best reproduction value within a partition is the centroid;
2. the best partition is formed by values which are closest to the centroid (nearest neighbour condition).[1]

Using these conditions, it is possible to design optimum non-uniform scalar quantisers for any arbitrary density in an iterative fashion by continually finding new centroids (condition 1), adjusting the cell boundaries (condition 2), recalculating centroids (condition 1), ad infinitum. The distortion is guaranteed to decrease or remain the same after each iteration, giving a local optimal solution. Lloyd [18] and Max [19] derived tables of quantiser levels for the Gaussian density while Paez and Glisson [20] provided quantiser tables for other densities such as Gamma and Laplacian.

The Linde–Buzo–Gray (LBG) algorithm, also known as the generalised Lloyd algorithm (GLA), is an extension of the iterative Lloyd method I [18], for use in vector quantiser design. Because the LBG algorithm is not a variational technique,[2] it can be used for cases where: the probability distribution of the data is not known a priori; we are only given a large set of training vectors; and the source is assumed to be ergodic [4]. The LBG algorithm involves refining an initial set of reproduction code-vectors using the Lloyd conditions, based on the given training vectors. The iterative procedure is stopped after the change in distortion becomes negligible.

Linde et al. [4] also introduced a 'splitting' technique for initialising the LBG algorithm, rather than assume an arbitrary set of reproduction code-vectors. In the LBG splitting technique, the centroid, $\boldsymbol{y}_1^{(1)}$, of the entire training set, is split into two code-vectors, via a perturbation procedure, $\boldsymbol{y}_1^{(1)} + \boldsymbol{\epsilon}$ and $\boldsymbol{y}_1^{(1)} - \boldsymbol{\epsilon}$. Then the training set is classified to these two code-vectors, based on the nearest neighbour criterion, and the centroids of these two clusters are refined using the Lloyd conditions to give the code-vectors for a 1 bit vector quantiser, $\boldsymbol{y}_1^{(2)}$ and $\boldsymbol{y}_2^{(2)}$. These code-vectors are split and the process continues until we reach the desired number of code-vectors, $\{\boldsymbol{y}_i^{(k)}\}_{i=1}^{K}$, at the kth step. Figure 2 shows the code-vectors and Voronoi regions after each successive step of the LBG algorithm. It should be noted that the splitting technique for vector quantiser initialisation is not guaranteed to produce an optimal solution and that other training algorithms are available which may give a better solution.

[1] For the scalar case, the boundaries of each partition are mid-way between the reproduction values.

[2] That is, it does not require the evaluation of derivatives, as opposed to Lloyd method II [4].

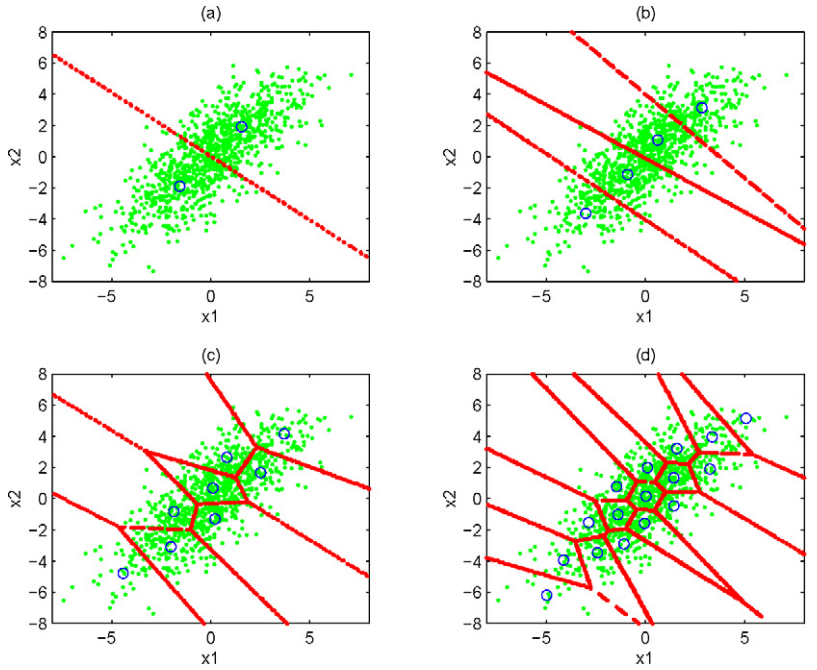

Fig. 2. Successive code-vectors and Voronoi regions during the design of a 4 bit vector quantiser using the LBG algorithm: (a) after first split and refinement; (b) after second split and refinement; (c) after third split and refinement; (d) after final split and refinement.

2.2. *Vector properties and vector quantiser advantages*

Shannon [1] showed that quantising a vector of data points is more efficient than quantising individual scalar values, in the rate-distortion sense. Therefore, for a given bitrate and dimension, the vector quantiser will, theoretically, incur the least distortion of any quantiser. The question to ask is why, and by how much, does a vector quantiser gain over the scalar quantiser? In order to understand the advantages of vector quantisation, it is necessary to first describe certain properties of vectors.

2.2.1. *Properties of vectors*

Makhoul [10] noted four properties of vectors that should be exploited by quantisers in order for them to "result in optimal performance." These properties relate to Shannon's asymptotic quantisation theory of vectors with arbitrarily large dimensions.

Linear and non-linear dependency. Linear dependency refers to the correlation between vector components. It is well known in the coding literature that correlation between components constitutes redundancy. A quantiser that can exploit linear dependencies or correlation between vector components will result in better quantiser performance. With the exception of vectors originating from a Gaussian source, non-linear dependency between vector components remains, even after decorrelation [10]. Block quantisers and transform coders, which utilise the KLT to decorrelate the vectors, cannot exploit non-linear dependencies, hence they are suboptimal. An optimum quantiser should be able to exploit both linear and non-linear dependencies between vector components.

Dimensionality. In higher dimensions, cell shapes[3] become an important factor in determining quantiser performance. Working in higher dimensions allows an optimal quantiser to have the flexibility of using different cell shapes. For example, scalar quantising each component of a two dimensional vector with a uniformly distributed PDF results in square cells only, with the reproduction code-vector appearing in the centroid [10]. For quantisers that operate at and are 'aware' of two (or, higher) dimensions, then hexagonally-shaped quantiser cells are possible. It can be shown

[3] In higher dimensional space, regions bounded by numerous hyperplanes are termed as *polytopes* [10]. However, for the sake of simplicity, we will refer to them as cells.

that using hexagonal cells results in approximately 0.17 dB less MSE than when using the square cells for the same number of bits. If the distortions of both shapes are made equal, then it can be shown that the hexagonal cells are about 1.94% larger than square cells, hence they can cover the same area and incur the same quantiser distortion with lesser cells, which corresponds to a saving of 0.028 bits [10]. In summary, going to higher dimensions allows more freedom in choosing cell shapes that either minimise distortion for the same number of bits, or use less bits to achieve the same amount of distortion.

Probability density function shape. The distortion of a quantiser is also highly dependent on how well it matches the probability density function of the data. Therefore, an optimal quantiser places more quantiser reproduction values in regions of high probability and less in regions of low probability. *Cell size*, rather than shape, is important in this regard, as smaller cell sizes allow a closer packing of reproduction vectors (or, cell centroids).

2.2.2. Vector quantiser advantages

Using high resolution quantisation theory,[4] Lookabaugh and Gray [3] described three 'advantages' of the vector quantiser over the scalar quantiser and related them to Makhoul's vector properties [10]. They quantitatively defined the vector quantiser advantage, $\Delta(n, r)$, where n is the dimensionality and r is the power of the distortion measure,[5] as the "ratio of the distortion of repeated scalar quantisation to that due to vector quantisation." Assuming a stationary source, the vector quantiser advantage can be decomposed into three parts:

$$\Delta(n, r) = F(n, r)S(n, r)M(n, r), \tag{3}$$

where each of the terms on the right denote the space-filling advantage, the shape advantage, and the memory advantage, respectively [3].

Space-filling advantage. According to [3], the space-filling advantage is the ratio of the coefficient of quantisation[6] of scalar quantisation, $C(1, r)$, to that of n-dimensional vector quantisation, $C(n, r)$. That is:

$$F(n, r) = \frac{C(1, r)}{C(n, r)}. \tag{4}$$

Since Gersho [22] conjectured the coefficient of quantisation to be dependent on the inverse of the polytope or cell volume, then a quantiser whose cells can fill more space will have a smaller coefficient of quantisation. For a two-dimensional ($n = 2$) vector quantiser using MSE ($r = 2$) as the distortion measure, the optimal cell shape is the hexagon [3,14]. As we have observed with regards to the dimensionality property of vectors, hexagonal cells occupy more area than square cells (scalar quantisation). Hence $C(2, 2) < C(1, 2)$ which means $F(2, 2) > 1$. Therefore, due to the space-filling advantage, which utilises Makhoul's dimensionality property [10], vector quantisers will always do better than scalar quantisers, regardless of the source PDF, in the high resolution sense.

Shape advantage. According to [3], the shape advantage, $S(n, r)$, is dependent on the shape of the marginal PDF. Values of $S(n, 2)$, calculated for standard densities (Gaussian, Laplacian, Gamma), show that the vector quantiser is expected to perform at least 1.14 dB (for a Gaussian density and $n = 2$) better than the scalar quantiser, due to the shape advantage alone [3]. In this case, the vector quantiser is able to exploit the PDF shape property described by Makhoul [10].

Memory advantage. According to [3], the memory advantage, $M(n, r)$ is the ratio of the $n/(n + r)$th "norms of the vector's probability density and the product of its marginals." Therefore, the memory advantage is dependent on how much statistical dependency exists between vector components, and is therefore related to Makhoul's linear and non-linear dependence property of vectors [10]. For independent and identically distributed (iid) vectors, the source PDF is equivalent to the product of the marginal PDFs, thus the memory advantage for iid vector sources is equal to 1 [3]. In other words, the vector quantiser has no memory advantage when quantising vectors whose components are statistically independent.

---

[4] Attributed to the work of Bennett [21], high resolution quantisation theory assumes arbitrarily large bitrates for a fixed dimension, which is different to Shannon's asymptotic quantisation theory of arbitrarily large dimensions.

[5] $r = 2$ is mean squared error [3].

[6] The coefficient of quantisation is defined as $C(n, r) = \frac{1}{n} \inf_{H \in H_n} I(H)$, where $I(H)$ is the normalised inertia of the polytope, H, and H_n is the class of admissible polytopes in $\mathcal{R}^n$ [22].

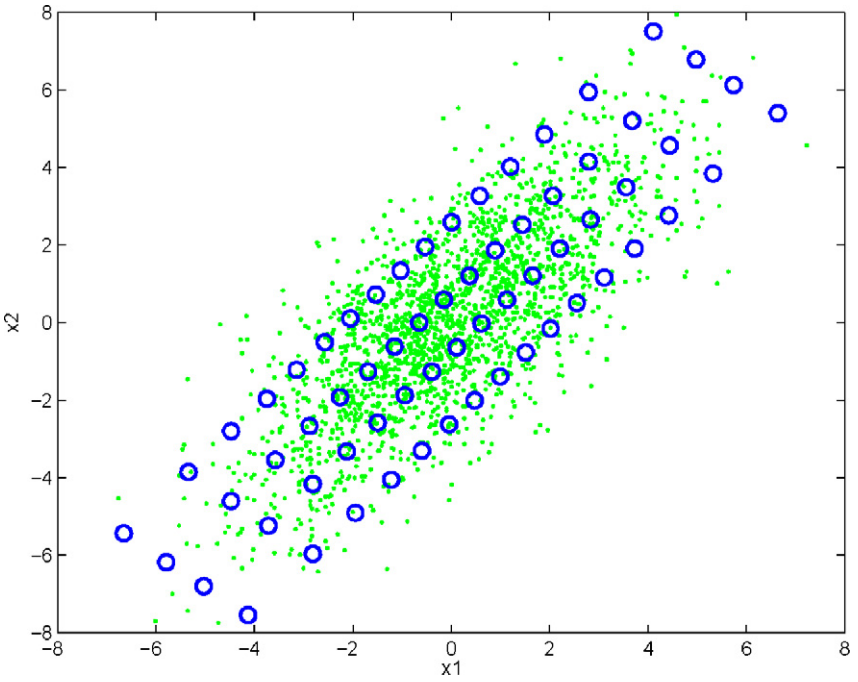

Fig. 3. Block quantisation using 3 bits/sample with KLT applied (SNR = 16.33 dB). Reproduction points (circles) and original data (dots) is also shown.

2.2.3. Vector quantisation of statistically-independent sources

An interesting result that can be drawn from these vector quantiser advantages, is that when quantising vectors that are statistically independent, though there will be no memory advantage, the vector quantiser will manage to outperform the scalar quantiser because of the space-filling and shape advantages [3]. We can experimentally show this by comparing the SNRs of a block quantiser with the vector quantiser at the same bitrate. The block quantiser[7] was investigated by Huang and Schultheiss [25] and combines the use of non-uniform scalar quantisers with the Karhunen–Loève transform (KLT). The KLT decorrelates vectors and then each transform coefficient is independently scalar quantised using Lloyd–Max scalar quantisers. Two-dimensional vectors that are jointly Gaussian with the following covariance matrix:

$$\mathbf{\Sigma}_{\boldsymbol{x}} = \begin{bmatrix} 4.5 & 4.0 \\ 4.0 & 6.0 \end{bmatrix} \tag{5}$$

were generated and used to train a 3 bits/sample (or 6 bits/vector) block quantiser and vector quantiser.

Figures 3 and 4 show the code-points of the block quantiser and vector quantiser applied on another set of random Gaussian vectors with the covariance matrix given by (5), respectively. Because the source is Gaussian, the KLT will produce *independent components*, or in other words, remove all dependencies (linear and non-linear). This fact allows us to compare the block quantiser with the vector quantiser having no memory advantage and relying solely on the space-filling and shape advantages. Based on the high resolution results of [3], the expected vector quantiser advantage, $\Delta(2,2) = F(2,2)S(2,2)$, is 1.31 dB. The SNR of the 3 bits/sample block quantiser is 16.33 dB while the SNR of the vector quantiser is 17.05 dB, giving an advantage of 0.72 dB. Therefore, even on independent vectors, the vector quantiser will always perform better than the scalar quantiser. The discrepancy between high resolution quantisation theory and experiment may be due to the relatively low bitrate as well as mismatch between the training and test vector set. Comparing the locations of reproduction code-vectors in Figs. 3 and 4, it can be said that the code-vectors of the scalar quantiser are constrained to lie in a rectangular grid, unlike those of the vector quantiser, which are not constrained but are free to appear anywhere, forming arbitrary cell shapes.

[7] The block quantiser is one type of transform coder that is fixed-rate and does not use entropy coding [23], unlike the variable-rate transform coder used in the JPEG image coding standard [24].

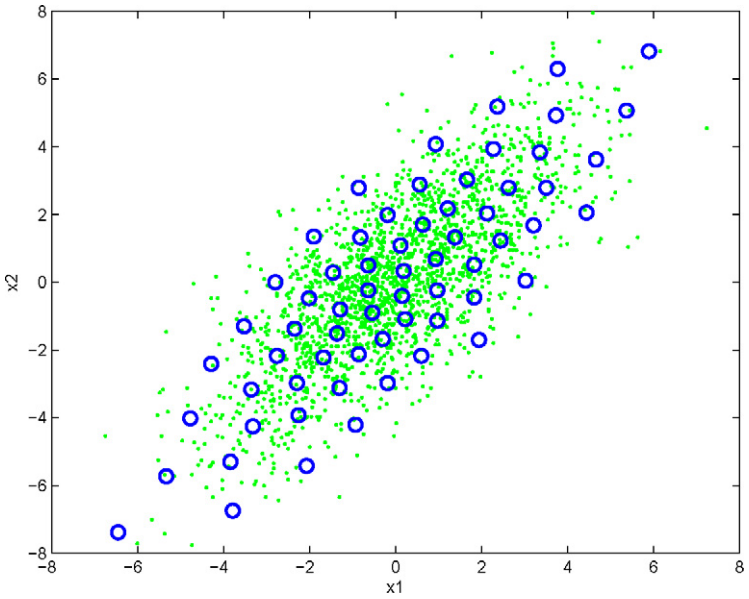

Fig. 4. Vector quantisation using 3 bits/sample (17.05 dB). Reproduction points (circles) and original data (dots) is also shown.

2.3. *Practical limitations of unconstrained vector quantisation*

Though the unconstrained vector quantiser is the optimal quantiser for achieving the lowest distortion at a given bitrate and dimension, its exponentially-growing computational complexity and memory requirements often render it impractical for applications where a high bitrate is required. A b bit, n-dimensional vector quantiser will possess a codebook of 2^b code-vectors. In terms of the memory requirements and computational complexity, this vector quantiser needs to store $n2^b$ floating point values and compute $3n2^b - 1$ flops/vector,[8] respectively, when using the mean squared error as the distortion measure.

For image coding applications, blocks of $8 \times 8 = 64$ are typically quantised by transform coders such as JPEG [24], and this is considered the optimum block size [23,26]. Quantising each block using a full search, unconstrained vector quantiser at a bitrate of 0.25 bits/pixel, requires 227 kflops/pixel of computations and 4.2 million floats of memory for storing the codebook. In narrowband speech coding, linear predictive coding (LPC) parameter vectors of dimension 10 need to be quantised at 20 bits/vector using a full search, unconstrained vector quantiser, to achieve transparent quality. This corresponds to a computational complexity of about 31.5 Mflops/vector and a memory requirement of 10.5 million floating point values.

Another problem related to vector quantiser design is that at high bitrates and dimensionality, where the codebook is large, the LBG algorithm will require a larger amount of training vectors in order to design a codebook that is representative of the source. Otherwise, there will be too much 'over-fitting' of the training set [27].

In general, vector quantisers possess a number of attributes: bitrate, distortion, computational complexity, and memory requirement. For a given bitrate, exhaustive search vector quantisers generally achieve the lowest distortion but, as noted before, they also require a large amount of searching and memory at high bitrates. In order to alleviate this computational and memory burden, we can apply structural constraints to the vector quantiser. With tree-structured vector quantisers (TSVQ), the search complexity is very low, though they require more memory than full search VQ and, due to suboptimal searching, quantisation performance is degraded [14]. Classified [28] or switch vector quantisers [29], which are equivalent to a two-stage TSVQ [14], also reduce the search complexity at the expense of higher memory requirements and generally suboptimal coding performance. Product code vector quantisers, such as split

[8] To calculate the MSE between two vectors of dimension n requires n subtractions, n multiplications, and $n - 1$ additions, giving a total of $3n - 1$ flops. Searching through the codebook of 2^b code-vectors needs $(3n - 1)2^b$ flops and in addition to the minimum distortion search, which requires $2^b - 1$ flops, gives a total complexity of $3n2^b - 1$ flops. Note that in our calculations, each addition, multiplication, and comparison is considered a floating point operation (flop).

and multistage [9,11], reduce the computational complexity and memory requirements by designing and operating independent vector quantisers, either of smaller dimension or consisting of more stages, respectively. However, the structural constraints degrade their quantisation performance. These *constrained vector quantisers* are described in the following subsections.

2.4. Product code vector quantisers

In order to make vector quantisers practical for large dimension and high bitrates, a structure can be imposed on the codebook to decrease the search complexity and/or storage requirements. One way of achieving this is to use decompose the codebook into a Cartesian product of smaller codebooks [2,14]. The idea of using product codes was first introduced by Sabin and Gray [12] with their shape-gain vector quantiser, in which vectors are quantised using both a shape codebook and gain codebook. Indices for each codebook are then transmitted to the decoder, which reconstructs the vector using its stored shape and gain codebook.

The definition of a product code vector quantiser is one with a codebook, $\mathcal{C} = \{\boldsymbol{y}_i\}_{i=1}^{K}$, that consists of m codebooks, $\{\mathcal{C}_i\}_{i=1}^{m}$, where $\mathcal{C}_i = \{\boldsymbol{u}_{i,j}\}_{j=1}^{K_i}$, such that [14]:

$$\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_m, \tag{6}$$

where $\times$ symbolises the Cartesian product. The code-vectors, $\boldsymbol{u}$, in each of the smaller codebooks, combine in a permutative fashion to form the product code-vectors, $\boldsymbol{y}$. Therefore, the *effective* size of the product codebook is [14]:

$$K = \prod_{i=1}^{m} K_i. \tag{7}$$

However, the actual size of the product codebook, which consists of the sum of the individual codebook sizes, is generally smaller than that of the effective product codebook [14]:

$$K \geqslant \sum_{i=1}^{m} K_i. \tag{8}$$

The advantages are that with smaller codebooks, code-vector searches are reduced and in most cases, the memory requirements are reduced as well. These come at the cost of suboptimal coding performance as the product code vector quantiser codebook has structural constraints [14]. Also, the lowest complexity product code-vector quantisers typically use sequential or independent searching and design and this leads to suboptimal product code-vectors being chosen or generated [14]. Also the issue of bit allocation among the various codebooks arises [2], which can often complicate design and lead to further suboptimality.

2.4.1. Split vector quantisers

An m part, n-dimensional split vector quantiser (SVQ)[9] [11] operating at b bits/vector, divides the vector space, $\mathcal{R}^n$, into m lower dimensional subspaces, $\{\mathcal{R}_i^{n_i}\}_{i=1}^{m}$, where $n = \sum_{i=1}^{m} n_i$. Independent codebooks, $\{\mathcal{C}_i\}_{i=1}^{m}$, operating at $\{b_i\}_{i=1}^{m}$ bits/vector, where $b = \sum_{i=1}^{m} b_i$, are then designed for each subspace.

Figure 5 shows the block diagram of a two part split vector quantiser. In order to quantise a vector of dimension n, the vector is split into subvectors of smaller dimension. Each of these subvectors are then encoded using their respective codebooks. The memory and computational requirements of the SVQ codebook are smaller than that of an unstructured VQ codebook. In terms of the number of floating point values for representing the SVQ codebooks as opposed to that of unstructured VQ:

$$\sum_{i=1}^{m} n_i 2^{b_i} \leqslant n 2^b \tag{9}$$

while the effective number of code-vectors of the resulting product codebook is the same as that of unstructured VQ at the same bitrate:

[9] Split vector quantisation is also known to as partitioned vector quantisation [14].

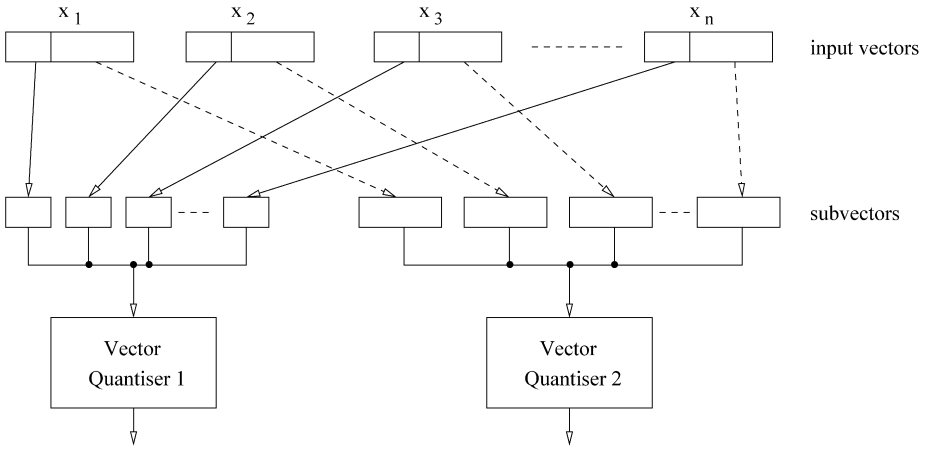

Fig. 5. Block diagram of a two part split vector quantiser (after [30]).

$$\prod_{i=1}^{m} 2^{b_i} = 2^b. \tag{10}$$

Therefore, the computational complexity and memory requirements of SVQ can be reduced considerably, by splitting vectors into more parts. In fact, the transform coder can be considered a special case of SVQ (when $m = n$) operating on transform coefficients [2].

Because there are independent vector quantisers, the bits need to be allocated to each of them. Generally, assigning each subvector quantiser an equal number of bits, whenever possible, results in a good compromise between quantisation performance and complexity [11]. The split vector quantiser was first introduced by Paliwal and Atal [11,31] for quantisation of line spectral frequencies (LSF) in narrowband CELP speech coders and is used in the adaptive multirate narrowband (AMR-NB) codec [32]. SVQ is also used for quantising Mel frequency-warped cepstral coefficients (MFCCs) in the ETSI distributed speech recognition (DSR) standard [33].

2.4.2. Suboptimality of split vector quantisers

The reductions in complexity and storage come at the cost of suboptimal coding performance. The coding performance of split vector quantisation is suboptimal because vector quantisers are designed independently within each lower dimensional subspace. This condition causes linear and non-linear dependencies that exist across these subspaces to not be exploited [14]. In addition to this, the vector splitting constrains the shape of the quantiser cells as well as the ability of the code-vectors to match the marginal PDF shape. In essence, all three vector quantiser advantages described by Lookabaugh and Gray [3] are affected by the vector splitting. In order to illustrate the shortcomings of SVQ and how it may be improved, we consider the simple case of designing a two-part SVQ for two dimensional vectors and examine each type of suboptimality.

Loss of the memory advantage. It is readily apparent that when the number of parts, m, is equal to the vector dimension, n, SVQ becomes equivalent to scalar quantising each vector component independently [11]. Similar to that from [14], Fig. 6a shows the probability density function of vector data with two correlated components, x and y. For a 4 bit SVQ (2 bits per partition), each vector, $[x, y]$, is partitioned and a codebook is designed for each partition based on the marginal PDFs shown in Figs. 6b and 6c. The resulting SVQ has an effective codebook of 16 code-vectors, as shown in Fig. 6d, while coding requires only $2^2 + 2^2 = 8$ searches. It can be observed that the resulting SVQ is suboptimal, as there are 8 product code-vectors which do not fall in areas of finite probability. In other words, the vector partitioning in SVQ does not allow for the exploitation of dependencies between subvectors, x and y. One would expect an improvement in the quantisation performance if subvectors are first 'decorrelated' before independent quantisers are designed. This is analogous to *block quantisation* or *transform coding* [25], where a Karhunen–Loève transform (KLT) is used to decorrelate individual vector components before they are independently coded using non-uniform scalar quantisers.

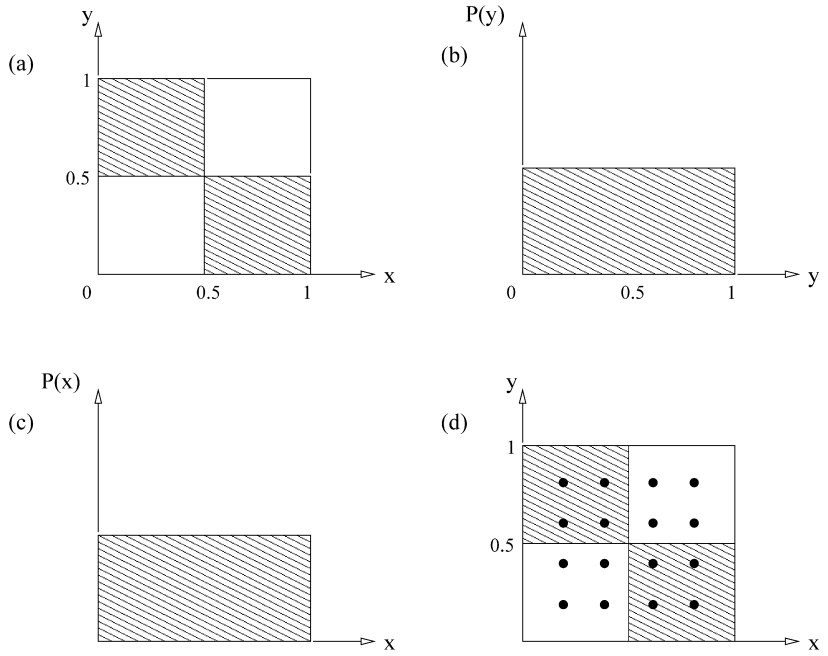

Fig. 6. Illustrating the memory suboptimality of a 4 bit two-part split vector quantiser in two dimensions (after [14]). (a) Two-dimensional PDF (shaded areas indicate uniform probability, white areas indicate zero probability); (b) marginal PDF of component y; (c) marginal PDF of component x; (d) product codebook of 4 bit split vector quantiser.

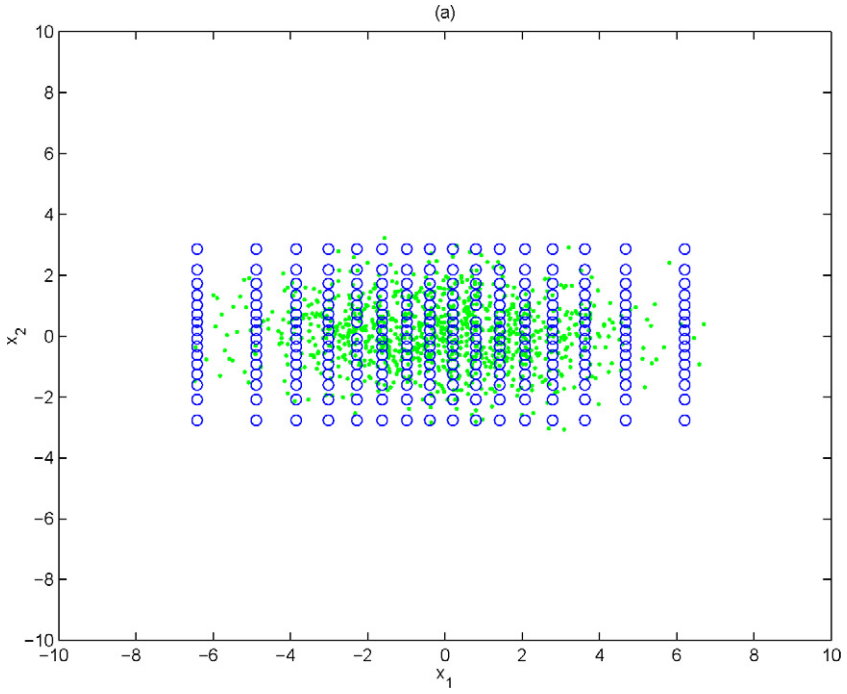

Fig. 7. Illustrating the space-filling and shape suboptimality of an 8 bit two-part split vector quantiser in two dimensions.

Loss of the space filling advantage. Figure 7 shows the product code-vectors of the two-part SVQ operating on two-dimensional Gaussian random vectors that are statistically independent. The covariance matrix of this random source is:

$$\mathbf{\Sigma} = \begin{bmatrix} 6 & 0 \\ 0 & 1 \end{bmatrix}. \tag{11}$$

For each vector quantiser, the best polytope in one dimension is the *interval* [3]. This results in the product code-vectors lying on a rectangular grid, hence the Voronoi regions are rectangular. However, in two dimensions, the optimum cell shape in two dimensions is the *hexagon* [22]. Therefore, we can see that the lower dimensionality has constrained the quantiser cell shape in a way that reduces the space-filling advantage of vector quantisation. For our example of two-dimensional vectors, the reduced dimensionality results in a theoretical loss of 0.17 dB, according to the results of [3].

Loss of the shape advantage. Each of the subvector quantisers have no problem with matching the marginal PDF shape of each dimension as they are equivalent to the Lloyd–Max non-uniform scalar quantiser, which are optimal (at least, locally) for *one dimension*. However, Fig. 7 also shows that the product code-vectors of the two-part SVQ do not match the elliptical shape of the marginal PDF in *two dimensions*. Therefore, the reduced dimensionality has constrained the shape advantage, which in the case of our example, results in a theoretical loss of 1.14 dB for a Gaussian source, according to the results of [3].

For an information theoretic interpretation of the losses in split vector quantisation, the reader should refer to [34], where conditional quantisation of splits is presented to reduce the losses. Smith et al. [35] have also reported on a normalisation technique for improving the split vector quantisation of speech line spectral frequencies.

2.4.3. Multistage vector quantisers

Figure 8 shows the block diagram of another type of product code vector quantiser,[10] first introduced by Juang and Gray [8], called the multistage vector quantiser (MSVQ). It is also referred to as a multistep, residual or cascaded vector quantiser [2]. Each vector, $\boldsymbol{x}$, is quantised by a coarse vector quantiser to produce an approximate vector, $\hat{\boldsymbol{x}}_1 = Q[\boldsymbol{x}]$. The quantisation error or residual, $\boldsymbol{e}_1 = \boldsymbol{x} - \hat{\boldsymbol{x}}_1$, is calculated and this is quantised by another vector quantiser, giving a quantised version of the residual vector, $\hat{\boldsymbol{e}}_1 = Q[\boldsymbol{e}_1]$. This process of determining the residual vector, quantising it, etc. can be continued, depending on how many stages are used. The decoder takes the indices, I_i, from each vector quantiser stage and adds them to get the reconstructed vector, $\hat{\boldsymbol{x}}$:

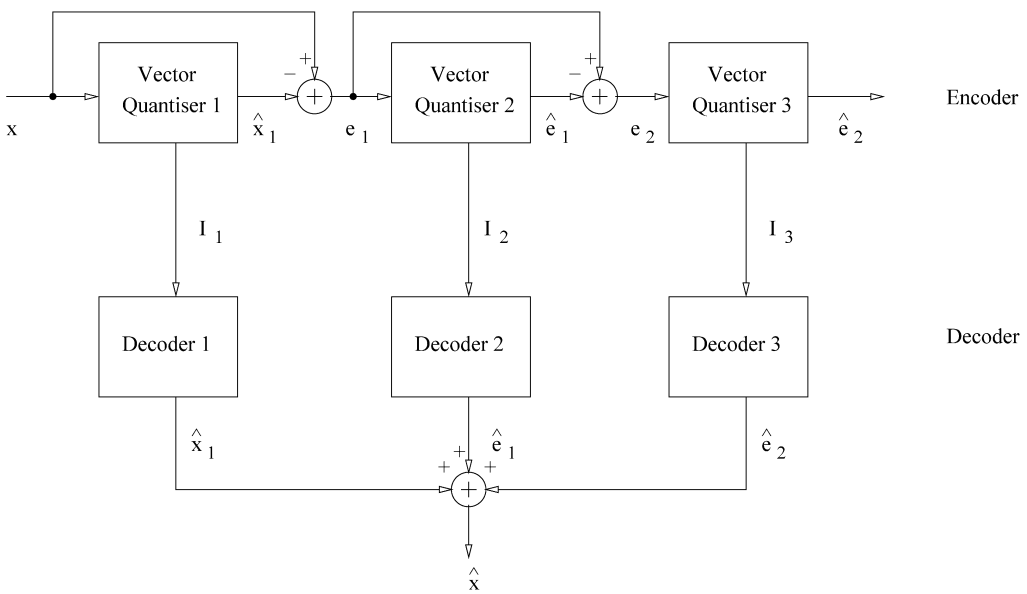

Fig. 8. Block diagram of a three-stage multistage vector quantiser (I_i denotes the ith quantiser index).

[10] The codebook of the multistage vector quantiser is formed by the direct sum of codebooks from each stage. However, it can also be viewed as a product code-vector quantiser, in the sense that the final codebook is formed from a Cartesian product of the individual codebooks [2].

$$\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}_1 + \sum_{i=1}^{m-1} \hat{\boldsymbol{e}}_i, \tag{12}$$

where m is the number of stages.

Because each stage is an independent vector quantiser, bits need to be allocated to each of them. With each vector quantiser operating at a lower bitrate, the memory and computational requirements are reduced. Comparing the total size (in number of floating point values) of the codebooks of an m stage, n-dimensional MSVQ operating at b bits/vector, with that of an equivalent unconstrained vector quantiser of the same bitrate and dimensionality:

$$\sum_{i=1}^{m} n2^{b_i} \leqslant n2^b, \tag{13}$$

where b_i is the number of bits given to the ith stage vector quantiser and $b = \sum_{i=1}^{m} b_i$. We can see that search and memory requirements of the MSVQ, while lower than those of the unconstrained vector quantiser, are not as low as those of the split vector quantiser, where the dimensionality of the codebooks is reduced, in addition to the bitrate.

The MSVQ is generally suboptimal because of the constrained structure of the codebooks as well as the sequential nature of the design and code-vector searches [2,9,36]. The sequential design of MSVQs, where each stage is designed independently using the LBG algorithm to minimise the distortion of the error from the previous stage, is suboptimal, in general [2]. The sequential design algorithm at each stage assumes that "subsequent stages are populated with zero vectors only" [9]. Also, the sequential searching, where a codevector is independently selected, such that it minimises the distortion at each stage, does not generally give optimal product code-vectors[11] [2,36].

The M–L searched multistage vector quantiser, also known as the tree-searched multistage vector quantiser, was introduced by LeBlanc et al. [9] which used a more optimal searching algorithm than the traditional sequential search. At the first stage of the MSVQ, M code-vectors are selected such that they give the least distortion. M residual vectors are calculated and a search of the second stage codebook is performed for each of the M residual vectors. This causes M paths to be created in the MSVQ. At the final stage, the path which gives the lowest overall distortion is chosen [9]. The M–L search codebook search procedure is shown in Fig. 9 where $M = 4$. It was shown that the performance of the M–L searched MSVQ approached that of the full-search vector quantiser for small values of M [9].

LeBlanc et al. [9] proposed several codebook design algorithms which attempt to minimise overall distortion of the reproduction algorithms. Sequential optimisation trains the codebook for each stage sequentially, fixing previous stage codebooks, in order to minimise overall distortion [9]. Iterative sequential optimisation initialises with the traditional sequential design. Then for each stage, the codebook is retrained while keeping all other stages fixed, in order to minimise overall distortion [9]. Other design algorithms include joint MSVQ codebook design [9,36,37], which generally have faster convergence and result in better rate-distortion performance.

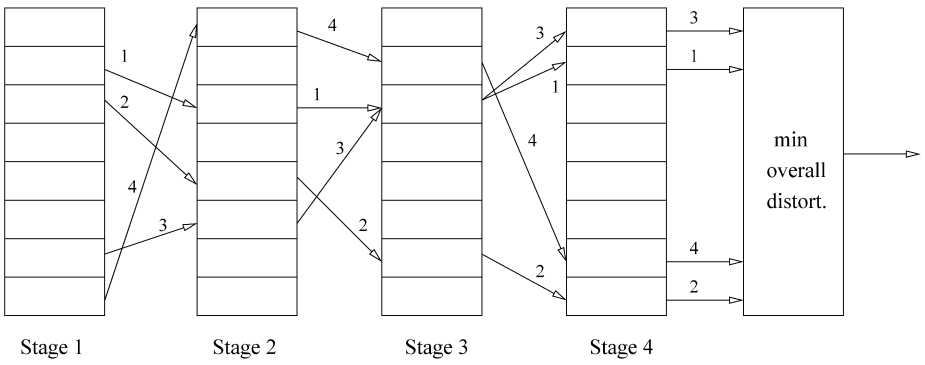

Fig. 9. Block diagram showing codebook searches in an M–L searched four-stage multistage vector quantiser (where $M = 4$). Each of the 4 paths is labelled.

[11] In contrast, sequential code-vector searches in the split vector quantiser are optimal (given the codebook), in the mean squared sense. SVQ loses its performance in the splitting of vectors, which reduces dimensionality and adds constraints to the subvector codebooks [9].

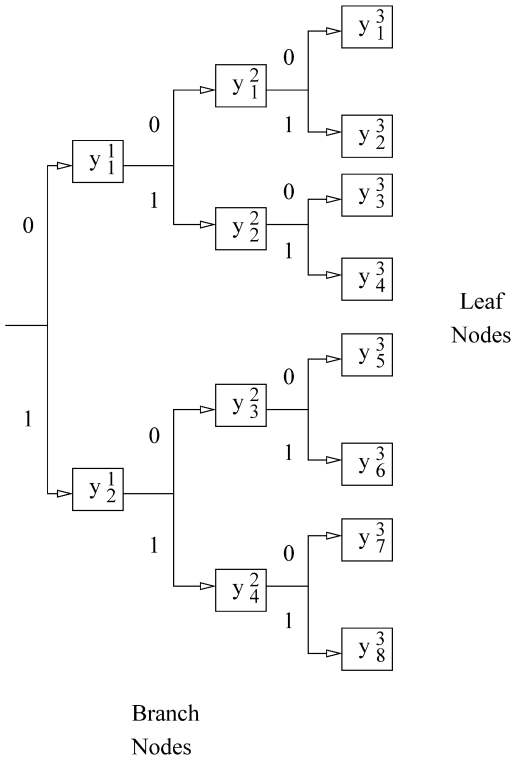

Fig. 10. A 3 bit tree structured vector quantiser.

2.5. Tree structured and classified vector quantisers

The tree structured vector quantiser (TSVQ) [5], as shown in Fig. 10, enforces a tree structure on the vector quantiser codebook [14]. By doing so, the number of code-vector searches is considerably reduced. The greedy method of designing a TSVQ is to recursively run the LBG algorithm on training vectors that are classified to each branch [2].

To quantise a vector, $\boldsymbol{x}$, it is firstly compared with the code-vector in each branch node. The branch node code-vector which is closest to x determines the path through the tree until we reach the code-vector at the leaf node, which completes the search. For a b bit TSVQ, only 2^b distortion calculations are required, as opposed to 2^b searches in an unconstrained, exhaustive search vector quantiser. Therefore, the computational complexity of TSVQ is very low [2]. However, the memory requirements of the TSVQ codebook are higher than the unconstrained vector quantiser. For an n-dimensional, b bit TSVQ, the total memory requirement (in floats) is:

$$\text{memory}_{\text{TSVQ}} = n \sum_{i=1}^{b} 2^i. \tag{14}$$

Hence for our 3 bit TSVQ example, we need to store 14 code-vectors of dimension n.

The performance of TSVQ is generally suboptimal due to the sequential searching algorithm and the structural constraint on the codebook. The path that is chosen through minimising the distortion at each stage, does not necessarily terminate at the optimum code-vector [14]. Also, the greedy method of designing the TSVQ does not necessarily produce an optimal TSVQ codebook either [2].

Related to the TSVQ are the classified vector quantiser (CVQ), introduced by Ramamurthi and Gersho [28] and the switched vector quantiser, used by Wang and Gersho [29]. Instead of having the binary branch structure, CVQ and switched VQ use m branches, each leading to a codebook representing a certain class. A classifier is used on vectors to be quantised, in order to determine the best codebook.

3. Switched split vector quantisation

3.1. Hybrid vector quantisation schemes

In the literature, hybrid vector quantisers have been investigated, where two or more of the VQ schemes are combined. Examples include the two stage vector quantisation–lattice vector quantiser (VQ–LVQ) by Pan and Fischer [38] and tree structured two stage vector quantisation–pyramidal lattice vector quantiser (TSVQ–PLVQ) by Pan [39]. A hybrid of split and classified vector quantisation was investigated by Zhou and Shoham [40]. The computational cost is reduced by replacing the full search vector quantisers with classified vector quantisers while the quantisation performance remained about the same as that of the full search-based SVQ [40]. The split–multistage vector quantiser (S–MSVQ) is a hybrid scheme that is used in the quantisation of LPC parameters in the ITU-T G.729 narrowband speech coder [41] and G.722.2 Adaptive multi-rate wideband (AMR–WB) speech coder [42]. S–MSVQ is essentially a multistage vector quantiser with a split vector quantisers in the second stage (in G.729) or both stages (in AMR–WB) and combines the low complexity characteristics of both quantisers.

In this section, we investigate the use of a hybrid of switch vector quantisation and split vector quantisation, called the switched split vector quantiser (SSVQ). It is different than the hybrid scheme considered by Zhou and Shoham [40], where they split vectors to be quantised by CVQs. In our scheme, vectors are classified using an exhaustive search switch vector quantiser and are then quantised by individual split vector quantisers. The advantage of classifying before splitting is that global dependencies between vector components are exploited in the first stage. Also, the suboptimality of splitting is then limited to local regions rather than the entire vector space. Hence, it will be shown that the SSVQ provides a better trade-off than traditional split vector quantisers in terms of bitrate and distortion performance, and offers a lower computational complexity, though at the cost of an increase in memory requirements. The SSVQ shares some similarities with the S–MSVQ of G.729, with a VQ and SVQ in the first and second stages, respectively. However, it differs from S–MSVQ, since in the latter scheme, an SVQ in the second stage quantises residual vectors (from the first stage) using the same codebook. In the case of SSVQ, vectors in the second stage are quantised by one from a number of SVQs, whose codebooks are adapted to each switched vector class.

3.2. Switched split vector quantisers

As we have seen in Section 2.4.2, using the two dimensional vector space analogy, we have shown how the split vector quantiser effectively becomes equivalent to the scalar quantising of the vector components when the number of parts is equal to the vector dimension. In block quantisation, applying a decorrelating linear transformation on the vectors can improve the coding efficiency of the independent scalar quantisers. Likewise in the SVQ case, we need to find a way of exploiting dependencies (linear and non-linear) between the subvectors before quantising them independently using vector quantisers. This leads to the switched split vector quantiser (SSVQ).

In SSVQ, an initial unconstrained vector quantiser (the switch vector quantiser) is used to classify[12] the vector space into Voronoi regions or clusters, which allows the exploitation of linear and non-linear dependencies across all dimensions. Then for each cluster, a local split vector quantiser is designed. The novelty of SSVQ is to populate the vector space with a number of different split vector quantisers such that they are positioned to exploit global dependencies. Each split vector quantiser is adapted to the local statistics of the Voronoi region and the suboptimalities of SVQ are localised.

Figure 11 shows a schematic of SSVQ codebook training. The LBG algorithm [4] is first applied on all vectors to produce m centroids (or means), $\{\boldsymbol{\mu}_i\}_{i=1}^{m}$. In the Euclidean distortion sense, these centroids are the best representation of all the vectors in that region. Hence, we can use them to form the switch vector quantiser codebook which will be used for switch-direction selection. All the training vectors are classified based on the nearest-neighbour criterion:

$$j = \underset{i}{\operatorname{argmin}}\, d(\boldsymbol{x} - \boldsymbol{\mu}_i), \tag{15}$$

[12] We note that SSVQ is similar to the two-stage classified vector quantiser with a second stage that is dependent on the first stage class. As opposed to other classified vector quantisation schemes, we use unsupervised classification in order to exploit vector component dependencies.

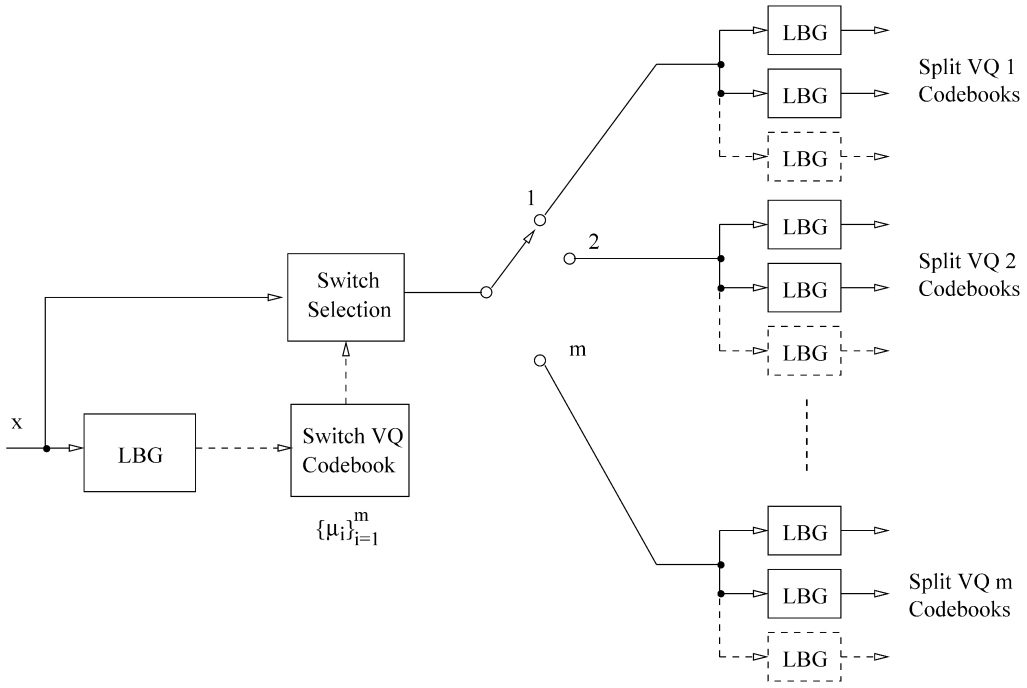

Fig. 11. Switched split vector quantiser (training).

where $\boldsymbol{x}$ is the vector under consideration, $d(\boldsymbol{x} - \boldsymbol{\mu}_i)$ is a suitable distortion measure between two vectors, and j is the cluster (or switching direction) to which the vector is classified. With the training vectors classified to the m clusters, local SVQ codebooks are designed for each cluster (or switching direction) using the corresponding training vectors.

Figure 12 shows a block diagram of SSVQ coding. Each vector to be quantised is first switched to one of the m possible directions based on the nearest-neighbour criterion defined by (15), using the switch VQ codebook, $\{\boldsymbol{\mu}_i\}_{i=1}^{m}$, and then quantised using the corresponding SVQ.

The SSVQ decoder needs to know which SVQ was chosen to quantise each vector. This can be done by transmitting the switching direction number as explicit side information. Alternatively, this information may be implicitly coded by partitioning the quantiser index range, similar to what is done in GMM-based block quantisers [43]. Assuming an SSVQ operating at b bits per vector and m-direction switch, we require $\log_2 m$ bits to uniquely identify all possible switching directions. Therefore, each split vector quantiser would be given a budget of $(b - \log_2 m)$ bits or $2^b/m$ indices. This suggests that the overall quantiser index range of 2^b can be divided into m partitions, each containing the valid indices which a particular SVQ can assign. The decoder can determine which SVQ was used for encoding a vector by finding which partition the quantiser index belongs to. Figure 13 shows an example of dividing the quantiser index range for a 3 bit SSVQ with a two-directional switch. Since one bit is needed to uniquely identify each switching direction, 2 bits are allocated to each of the SVQs. The binary indices, 000, 001, 010, 011, are valid for the SVQ of switching direction 1 while 100, 101, 110, 111, are valid indices for the SVQ of switching direction 2.

In a minimum-distortion sense, it would be logical to quantise each vector using all the split vector quantisers and then pick the one which results in the least distortion. This is an approach taken in multiple transform domain split vector quantisation [44] and similarly, in Gaussian mixture model-based block quantisation [43]. The disadvantage with this 'soft' decision scheme is the high computational complexity which results from each vector being quantised multiple times. SSVQ reduces the complexity by employing a switch, which makes a 'hard' decision based on nearest neighbour classification in order to determine the SVQ that would most likely quantise with the least distortion. As such, there will be a distortion penalty incurred by using the 'hard' decision, though this is offset by a considerable reduction in computational complexity.

One modification that can be made, in order to lower the suboptimality introduced by the 'hard' decision made by the switch vector quantiser, is to adopt a scheme similar to the M–L search MSVQ. Specifically, instead of choosing one switching direction, we choose M switching directions, quantising using their respective SVQ codebooks, and

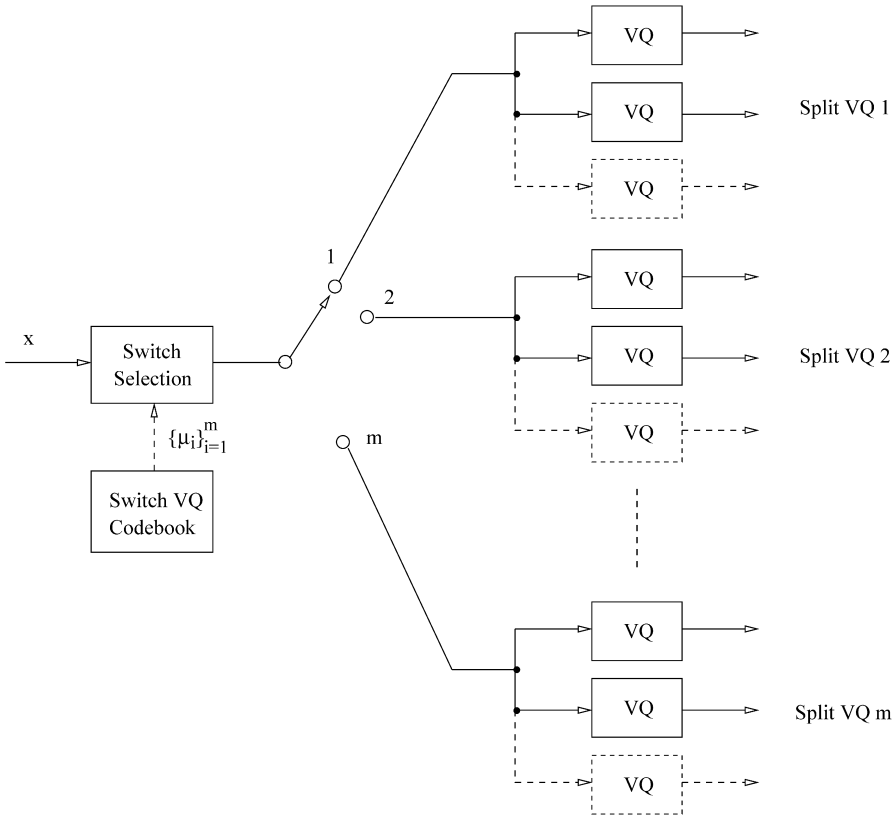

Fig. 12. Switched split vector quantiser (coding).

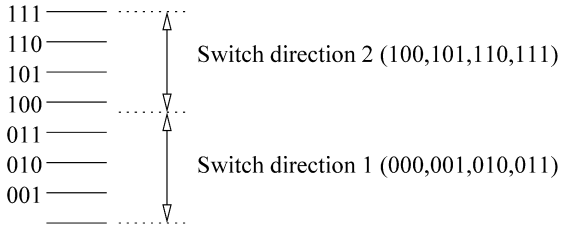

Fig. 13. Quantiser index partitioning for 3 bit SSVQ with two-way switch.

pick the one which incurs the least quantiser distortion. This is a compromise between the 'hard' decision ($M = 1$) and a full-search, 'soft' decision ($M = m$), and quantiser performance is expected to improve, though at the expense of an increase in computational complexity.

3.3. Advantages of the switched split vector quantiser

3.3.1. The memory advantage of the SSVQ

Returning to our two dimensional PDF illustration, as shown in Fig. 14, the SSVQ (with two switching directions) quantises the entire vector space with two initial code-vectors and classifies them into two clusters. Then two-part split vector quantisers are designed for each cluster. As we can see in Fig. 14b, the initial unconstrained vector quantiser (which we referred above as the switch vector quantiser) has exploited the global dependencies between the two vector components and two-part split vector quantisers are positioned to reflect this dependency. In contrast to the product code-vectors of Fig. 6d, the SSVQ product code-vectors are better placed. Therefore, the switch vector quantiser, which is unconstrained, allows the SSVQ to exploit at least some of the global dependencies that would, otherwise,

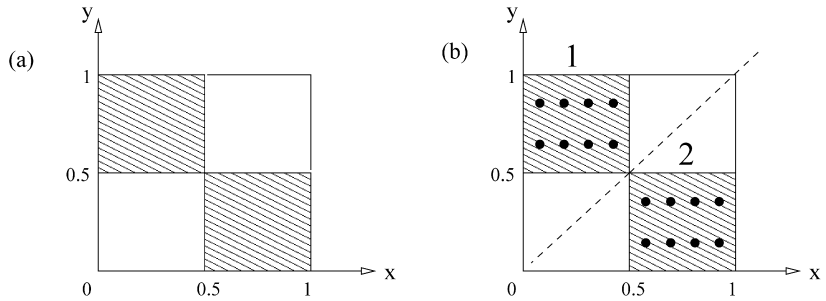

Fig. 14. Illustrating the memory advantage of the switched split vector quantiser. (a) Two-dimensional PDF (shaded areas indicate uniform probability, white areas indicate zero probability); (b) product codebook of 4 bit switched split vector quantiser (dashed line shows boundary between switch cells while numbers indicate switching direction).

have not been exploited by SVQ (as shown in Fig. 6). Therefore, SSVQ should recover some of the memory advantage lost due to vector splitting.

For a quantitative comparison of SSVQ with SVQ, correlated Gaussian random vectors of two dimensions with a covariance matrix of:

$$\mathbf{\Sigma} = \begin{bmatrix} 4.5 & 4.0 \\ 4.0 & 6.0 \end{bmatrix} \tag{16}$$

were generated and quantised using a two-part SVQ and two-part SSVQ at 8 bits/vector. The SSVQ uses 16 switching directions and each part was allocated 2 bits. Figs. 15a and 15b show the resulting product code-vectors of the SVQ and SSVQ, respectively. It can be observed in Fig. 15a that the SVQ has not exploited the correlation in the vectors, characterised by the tilt. This has resulted in a large amount of code-vectors that fall in areas where there are no vectors. In comparison, Fig. 15b shows the switch code-vectors (marked as crosses) have captured the major dependency between the components, x_1 and x_2. Each of the local SVQs have been placed according to these switch code-vectors where we can see a better utilisation of the code-vectors. The 8 bits/vector SSVQ has made a 2.58 dB gain in SNR over the SVQ.

3.3.2. The shape advantage of the SSVQ

In order to observe the shape advantage of the SSVQ over SVQ, we have generated some memoryless Gaussian random vectors with a covariance of (11). Because the vectors have no memory, any gains in SNR are mostly due to the shape advantage. In this example, where the same vector splitting is used for both SSVQ and SVQ, there will be no space-filling advantage of SSVQ over SVQ since both split vectors into subvectors of the same dimensionality, which constrains the quantiser cell shapes to be the same (in this case, rectangular).

Figures 16a and 16b show the product code-vectors of an 8 bits/vector split vector quantiser and 8 bits/vector switched split vector quantiser, respectively. In both cases, there are a total of $2^8 = 256$ code-vectors. For the SSVQ, 16 switching directions were used and 2 bits were assigned to each subvector in the local SVQs. When observing Fig. 16a, we can see that the lattice of SVQ code-vectors does not match the marginal shape of the two dimensional PDF, which is elliptical. In comparison, the code-vectors resulting from SSVQ, as shown in Fig. 16b, match the elliptical shape of the marginal PDF more closely than the rectangular shape of the SVQ. In terms of SNR, we see that the shape advantage gain of SSVQ over SVQ is approximately 0.5 dB.

3.4. Computational complexity and memory requirements

The main advantages of SSVQ are the better trade-off between bitrate and distortion and low computational complexity. In this section, we examine the latter characteristic and compare it to that of the split vector quantiser. Computational complexity will be measured in flops/vector, where each addition, multiplication, and comparison is counted as one floating point operation (flop). We also assume the use of a weighted mean squared error distance measure (that will be discussed in Section 4.2.2), similar to the one proposed in [11]. This increases the complexity of the vector quantiser from $3n2^b - 1$ to $4n2^b - 1$ flops, where n and b are the vector dimension and number of bits, respectively.

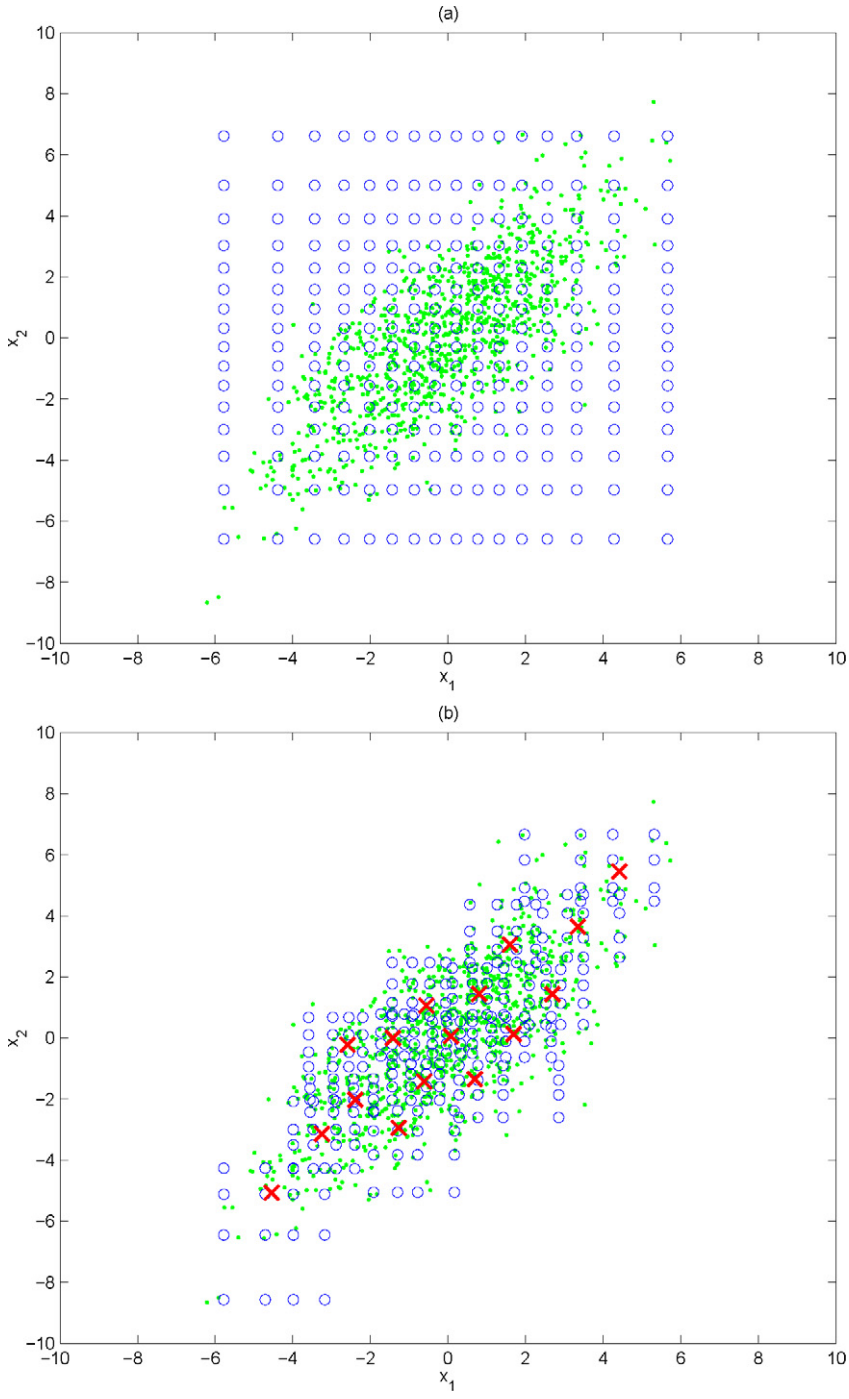

Fig. 15. Comparing product code-vectors of (a) 8 bits/vector split vector quantiser (SNR = 20.32 dB); with (b) 8 bits/vector switched split vector quantiser (SNR = 22.9 dB). The correlated Gaussian random vectors are represented as dots, the code-vectors as circles, and the switch code-vectors of SSVQ as crosses.

There are a number of design parameters of the SSVQ and these are stated below:

- dimension of the vectors, n;
- the number of bits allocated to the switch vector quantiser, b_m, and number of switching directions, $m = 2^{b_m}$;

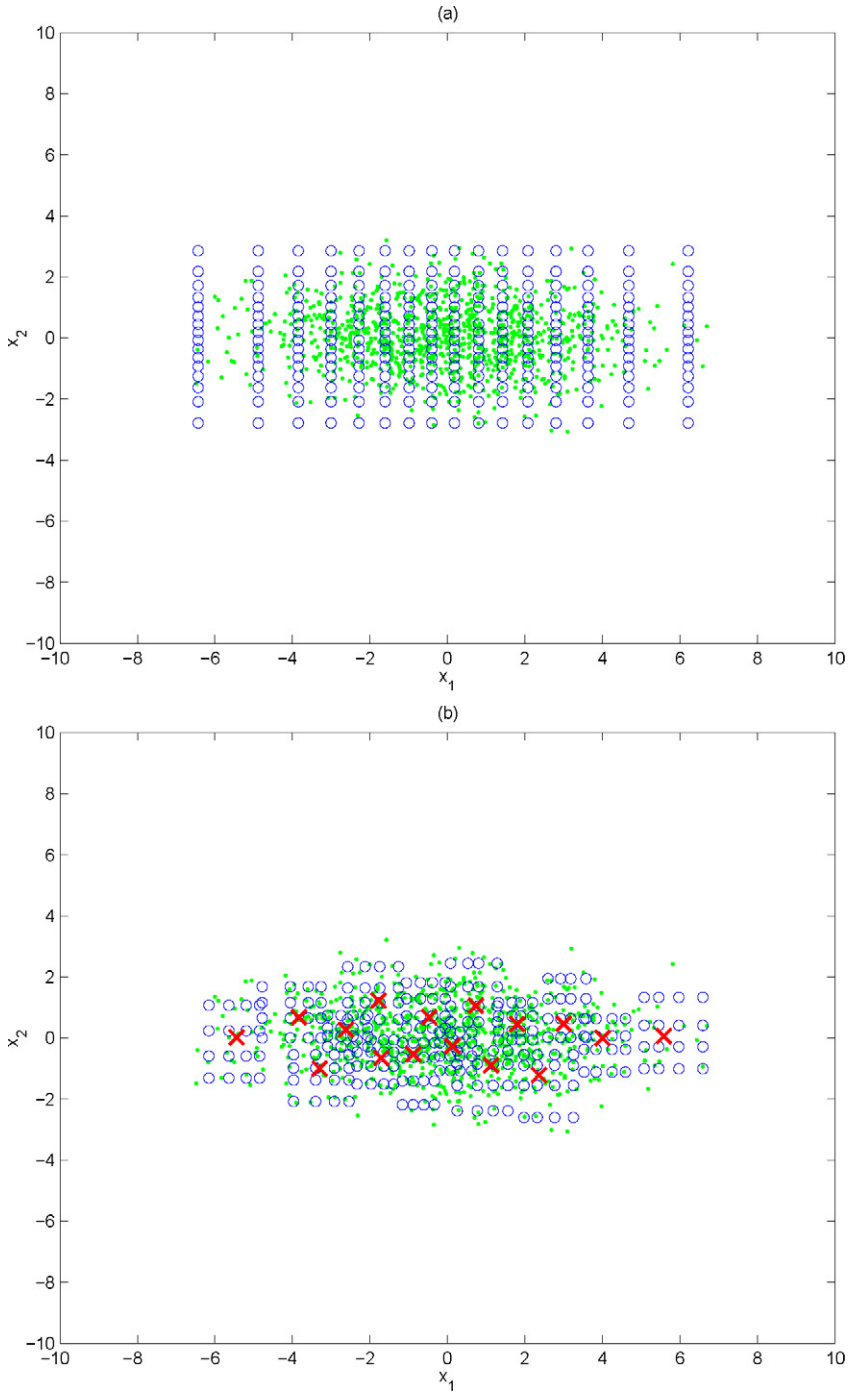

Fig. 16. Comparing product code-vectors of (a) 8 bits/vector split vector quantiser (SNR = 21.29 dB); with (b) 8 bits/vector switched split vector quantiser (SNR = 21.78 dB). The memoryless Gaussian random vectors with covariance of (11) are represented as dots, the code-vectors as circles, and the switch code-vectors of SSVQ as crosses.

- the total number of bits, b_{tot};
- number of parts in vector splitting, s;
- dimension of the subvectors, $\{n_i\}_{i=1}^{s}$, where $n = \sum_{i=1}^{s} n_i$;
- bits allocated to each of the subvectors, $\{b_i\}_{i=1}^{s}$, where $b_{\text{tot}} = \sum_{i=1}^{s} b_i$.

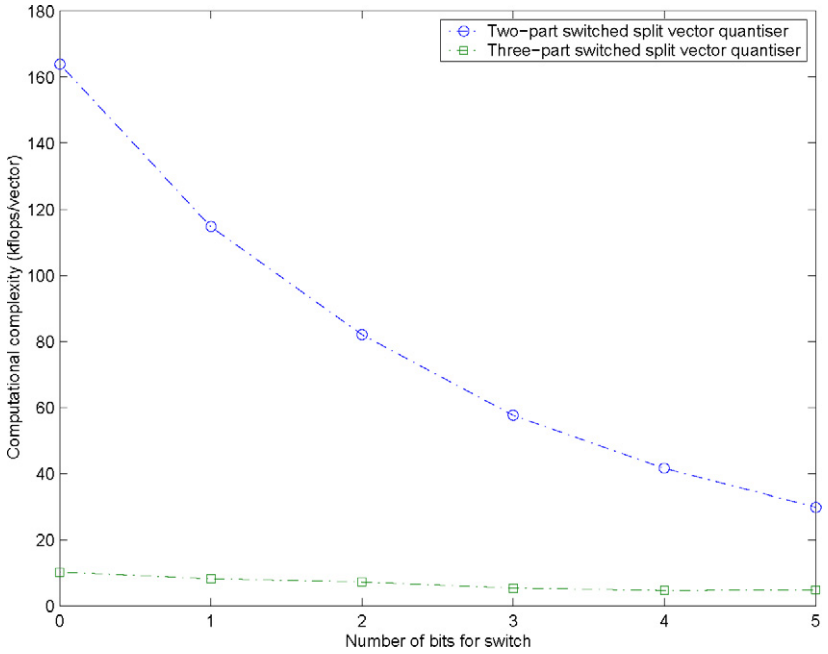

Fig. 17. Computational complexity (in kflops/frame) of two-part and three-part 24 bits/vector SSVQ (dimension 10) as a function of number of bits for switch vector quantiser.

The computational complexity of the switch vector quantiser is given by

$$\text{complexity}_{\text{switch}} = 4n2^{b_m} - 1 \tag{17}$$

while that of a single split vector quantiser, is given by

$$\text{complexity}_{\text{SVQ}} = \sum_{i=1}^{s} \left(4n_i 2^{b_i} - 1\right). \tag{18}$$

Therefore, the total complexity of the switched split vector quantiser is given by

$$\text{complexity}_{\text{SSVQ}} = 4n2^{b_m} - 1 + \sum_{i=1}^{s} \left(4n_i 2^{b_i} - 1\right). \tag{19}$$

In order to get an idea of the reduction in computational complexity that SSVQ requires as opposed to that of the split vector quantiser, Fig. 17 shows the computational complexity of a two-part SSVQ as a function of the number of bits, b_m, for the switch vector quantiser. The bitrate is 24 bits/vector and the vectors are of dimension 10. Vectors are split into $(4, 6)$ or $(3, 3, 4)$ for the two-part and three-part SSVQ, respectively. Bits are assigned uniformly to the subvectors whenever possible. When $b_m = 0$, the SSVQ reverts to a single split vector quantiser. We can see that the single SVQ has the highest computational complexity and as we use more switching directions, the computational complexity of SSVQ drops. This is because the number of bits available for each SVQ decreases as b_m increases. Also shown in Fig. 17 is the computational complexity of a three-part SSVQ where we can see that the further split results in a quantiser of much lower complexity.

The memory requirements, as a number of floating point values, of the SSVQ is given by

$$\text{memory}_{\text{SSVQ}} = n2^{b_m} + 2^{b_m} \sum_{i=1}^{s} n_i 2^{b_i}. \tag{20}$$

Figure 18 shows the memory requirements, as a number of floats, of the codebooks for the same two-part and three-part SSVQ considered above, as a function of the number of bits for the switch vector quantiser. We can see that for SSVQ, the gains made in quantiser performance and reductions in computational complexity come at the expense

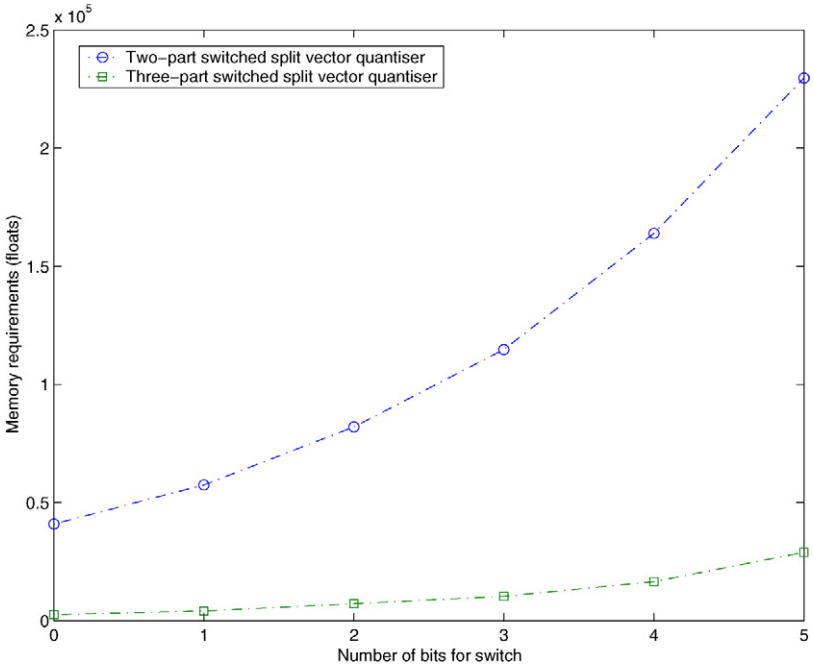

Fig. 18. Memory requirements (in number of floats) of two-part and three-part 24 bits/vector SSVQ (dimension 10) as a function of number of bits for switch vector quantiser.

of an exponential increase in the memory requirements, with the SVQ ($b_m = 0$) having the lowest memory. This is a disadvantage of the SSVQ scheme and may be minimised by using more vector splitting. Looking at Fig. 18, the memory requirements of the three-part SSVQ are not as high as the two-part SSVQ. Therefore, at this bitrate and vector dimension, the three-part SSVQ is a more practical scheme, with a moderate memory requirement.

4. LPC parameter quantisation for narrowband speech coding

4.1. LSF representation of LPC coefficients

In the LPC analysis of speech, a short segment of speech is assumed to be the output of an all-pole filter, $H(z) = \frac{1}{A(z)}$, driven by either white Gaussian noise (for unvoiced speech) or a periodic sequence of impulses (for voiced speech), where $A(z)$ is the inverse filter given by [11]

$$A(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}. \tag{21}$$

Here, n is the order of LPC analysis and $\{a_i\}_{i=1}^{n}$ are the LPC coefficients. Because $H(z)$ is used to reconstruct speech in linear predictive speech coders, its stability is of utmost importance and cannot be ensured when LPC coefficients are coded directly. Many representations of LPC coefficients have been proposed in the literature that are more robust, in terms of filter stability. These include the reflection coefficients (RC) or partial autocorrelation coefficients (PARCOR) [45], arc-sine reflection coefficients (ASRC) [46], and log area ratios (LAR) [47]. The line spectral frequency (LSF) representation, proposed by Itakura in [48], has been shown in the literature to be superior to other representations for speech coding [11,49,50].

The line spectral frequencies are defined as the roots of the following polynomials:

$$P(z) = A(z) + z^{-(n+1)} A(z^{-1}) \tag{22}$$

and

$$Q(z) = A(z) - z^{-(n+1)} A(z^{-1}). \tag{23}$$

These two polynomials, $P(z)$ and $Q(z)$, are parametric models of the acoustic tube in two extremal states, where the $(n+1)$th stage (representing the glottis) is either completely closed or completely opened, respectively [49,50]. Consequently, LSFs have the following properties [49]:

1. all zeros of $P(z)$ and $Q(z)$ lie 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 quantisation of the LSFs if the first two properties are satisfied.

Because of property 1, where each zero effectively represents a particular frequency (since it has no bandwidth), clusters of two to three LSFs define the location and bandwidth of formants in the power spectrum [11,50]. Quantisation errors in the LSFs result in localised distortion in the power spectrum [11].

4.2. Distortion measures for LPC parameters

4.2.1. Spectral distortion and conditions for transparent coding

A popular method for objectively evaluating the quantisation performance of LPC parameters is the use of the average spectral distortion of all frames. The spectral distortion of each frame is defined as the root mean squared error between the power spectral density estimate of the original and reconstructed LPC parameter vector [11]:

$$d_{sd}(i) = \sqrt{\frac{1}{F_2 - F_1}\int_{F_1}^{F_2}\left[10\log_{10} P_i(f) - 10\log_{10}\hat{P}_i(f)\right]^2 df}, \tag{24}$$

where $P_i(f)$ and $\hat{P}_i(f)$ are the LPC power spectra of the reconstructed and original ith frame, respectively. For narrowband speech that is sampled at 8 kHz, F_1 and F_2 are equal to 0 and 4 kHz for full-band spectral distortion or 0 and 3 kHz for partial-band spectral distortion [51], respectively. The partial-band spectral distortion is normally used for evaluating quantisation schemes that use a weighted distance measure.

Transparent coding means that the coded speech is indistinguishable from the original speech through listening tests. The conditions for transparent coding of speech from LPC parameter quantisation, which we have adopted in this work, are [11]:

1. the average spectral distortion (SD) is approximately 1 dB;
2. there is no outlier frame having more than 4 dB of spectral distortion;
3. less than 2% of outlier frames are within the range of 2–4 dB.

4.2.2. Weighted LSF distance measure

The weighted distance measure was introduced in [11] to replace the mean squared error (MSE) used in vector quantiser design. The weighting consists of:

1. fixed or static weights, $\{c_i\}$, which place emphasis on the lower LSFs in order to account for the difference in sensitivity of the human ear to low and high frequencies;
2. varying or dynamic weights, $\{w_i\}$, which emphasise the LSFs where the power spectrum is higher (i.e., formant regions).

The weighted distance measure, $d_w(\boldsymbol{f}, \hat{\boldsymbol{f}})$, between the original vector, $\boldsymbol{f}$, and the approximated vector, $\hat{\boldsymbol{f}}$, is defined as [11]

$$d_w(\boldsymbol{f}, \hat{\boldsymbol{f}}) = \sum_{i=1}^{10}\left[c_i w_i\left(f_i - \hat{f}_i\right)\right]^2, \tag{25}$$

where f_i and $\hat{f}_i$ are the ith LSF in the original and approximated vector respectively. The dynamic weights, $\{w_i\}$ are given by [11]

$$w_i = \left[P(f_i)\right]^r, \tag{26}$$

where $P(f)$ is the LPC power spectrum and r is a constant (typical value used is 0.15). The static weights, $\{c_i\}$, are given by [11]

$$c_i = \begin{cases} 1.0 & \text{for } 1 \leqslant i \leqslant 8, \\ 0.8 & \text{for } i = 9, \\ 0.4 & \text{for } i = 10. \end{cases} \tag{27}$$

4.3. Review of narrowband LPC parameter quantisation

Linear predictive coding (LPC) of speech requires the accurate quantisation of LPC parameters representing the spectral envelope. It has been shown in various studies that vector quantisers perform better than scalar quantisers, in terms of the number of bits needed for the transparent coding[13] of LPC parameters. For example, in the US Federal Standard 1016, 4.8 kbps code-excited linear prediction (CELP) coder [52], a total of 34 bits are required for independent non-uniform scalar quantisers to code the LPC parameters of each frame [11,52]. Extrapolating from the operating curve of full search vector quantisation suggests that we need about 19 bits/frame to achieve transparent coding of these parameters [53], while high rate analysis predicts a lower bound of 23 bits/frame[14] [27]. However, it is not possible to design codebooks at these rates and in addition, the computational cost of the resulting full search vector quantiser is very high.

Less complex but suboptimal vector quantisers such as multistage and split vector quantisers (MSVQ and SVQ) have been investigated in the speech coding literature [9,11], where it was generally observed that 22 to 24 bits/frame were required to achieve transparent coding in speech, with varying degrees of complexity. As with most product code vector quantisers [14], there is a trade-off between the quantiser performance, in terms of distortion and number of bits, as well as computational complexity in SVQ. Paliwal and Atal [11] reported a two-part SVQ requiring 24 bits/frame to achieve a spectral distortion of 1.03 dB with a computational complexity of 164 kflops/frame. By splitting vectors into three parts, the computational complexity was reduced to 13 kflops/frame at the cost of an extra bit (25 bits/frame) for a spectral distortion of 1.05 dB [11]. It is clear that while the computational complexity of SVQ can be reduced by splitting vectors into more parts, it comes at the cost of higher quantiser distortion and number of bits. The complexity trade-off was improved by Zhou et al. [40], where the full-search vector quantisers within SVQ were replaced with classified vector quantisers (CVQ). Their proposed system required 25% of the search complexity of full-search-based SVQ with no further reduction in distortion [40]. However, this system remains constrained in the number of bits required to achieve a certain spectral distortion as vectors are split off to independent CVQs, which are suboptimal when compared with full-search VQs. In other words, this system can only perform as well as full-search-based SVQ, but not any better.

4.4. Experimental setup

The TIMIT database was used to train and test the various quantisation schemes. It consists of speech down-sampled to 8 kHz with a 3.4 kHz anti-aliasing filter applied. A 20 ms Hamming window is used and a tenth order linear predictive analysis is performed on each frame using the autocorrelation method [53]. There is no overlapping between successive speech frames. High frequency compensation and bandwidth expansion of 15 Hz[15] was used to correct the effects of the anti-aliasing filter [51] as well as formant underestimation, respectively [54]. The training data consists of 333789 vectors while the evaluation set, which is exclusive of the training, contains 85353 vectors. Vectors

[13] Transparent coding means that the coded speech is indistinguishable from the original through listening. An objective measure of quality is the spectral distortion (SD), which is defined as the root-mean-square difference between the log-spectra of the coded and original speech. Coded speech is generally accepted as being transparent when the average SD is about 1 dB [11].

[14] This is the lower bound for full-band spectral distortion (0–4 kHz) while for partial-band (0–3 kHz), the bound is 22 bits/frame [27].

[15] This is the same high frequency compensation and bandwidth expansion contained in the source code of the US Federal Standard 1016, 4.8 kbps CELP coder described in [52].

are split into subvectors of dimension (4, 6) and (3, 3, 4) for the two-part and three part SVQ/SSVQ, respectively [11]. The weighted mean squared error from Section 4.2.2 is used in the design and encoding of the vector quantisers and partial-band spectral distortion [51] is used as a measure of the fidelity for these schemes.

We also present comparative results for other quantisation schemes, such as the multistage vector quantiser (sequentially-searched and M–L searched), PDF-optimised non-uniform scalar quantisers, and the GMM-based block quantiser. Most practical LSF quantisation schemes employ a moving-average (MA) predictor to exploit interframe correlation and these generally result in lower spectral distortion. However, these schemes often suffer when there are bit errors and also incur a larger percentage of outlier frames (those with large spectral distortion) due to the failure of the predictor to handle fast changes in the LSFs across speech frames. In this work, we have evaluated the SSVQ and other quantisation schemes with no interframe prediction.

In our implementations, we have added constraints to each sub-vector quantiser in the SVQ and SSVQ to ensure that the ordering property of the LSF frames (i.e., their stability) is not violated. For the two-part SVQ/SSVQ, the second sub-vector is vector quantised first. Then the first part is vector quantised with the constraint that the last coefficient in the chosen code-vector is less than the first coefficient of the second code-vector. Similarly, for the three-part SVQ/SSVQ, the second sub-vector is vector quantised first. Then the first and third sub-vectors are vector quantised with similar constraints to ensure the ascending order is preserved. In the results section, we also examine the performance of the SSVQ with no ordering constraints.

4.5. Performance of the switched split vector quantiser

4.5.1. Performance as a function of the number of switch directions

Table 1 shows the spectral distortion and computational performance of the two-part SSVQ at 24 bits/frame with varying number of switching directions and different bit allocations. For each number of switching directions, m, the least spectral distortion is highlighted in bold. We can see that non-uniform bit allocation generally results in lower spectral distortion, particularly for cases where the number of bits is divisible by two. Therefore, the bit allocation scheme we have adopted, where bits are uniformly allocated where-ever possible, is not optimal in the distortion sense. However, we also observe that non-uniform bit allocation leads to an increase in the number of computations required, thus there is a distortion/computational complexity trade-off.

Table 2 shows the spectral distortion and computational performance of the three-part SSVQ at 24 bits/frame with varying number of switching directions and different bit allocations. For each number of switching directions, m, the least spectral distortion is highlighted in bold. It can be seen that lower spectral distortion can be achieved when the first and second parts are given more bits than the third.

Another important observation to be made from Tables 1 and 2 is the consistent downward trend of the spectral distortion, as the number of switching directions is increased, which is plotted in Fig. 19. This is due to the increased ability of the unconstrained switch vector quantiser to exploit dependencies and PDF shape as its codebook becomes larger.

Table 1
Average spectral distortion (SD) and computational performance of the two-part switched split vector quantiser at 24 bits/frame as a function of the number of switch directions using different bit allocations

m	Total bits/frame $(b_m + b_1 + b_2)$	Avg. SD (in dB)	Outliers (in %)		kflops/frame
			2–4 dB	>4 dB	
1	24 (0 + 12 + 12)	0.943	0.54	0.00	163.9
2	24 (1 + 12 + 11)	0.943	0.62	0.00	114.8
	24 (1 + 11 + 12)	**0.920**	0.45	0.00	131.1
4	24 (2 + 11 + 11)	0.924	0.56	0.00	82.1
	24 (2 + 10 + 12)	**0.917**	0.50	0.00	114.8
8	24 (3 + 11 + 10)	0.926	0.68	0.00	57.7
	24 (3 + 10 + 11)	**0.903**	0.43	0.00	65.9
16	24 (4 + 10 + 10)	0.912	0.57	0.00	41.6
	24 (4 + 9 + 11)	**0.903**	0.46	0.00	58.0

Table 2
Average spectral distortion (SD) and computational performance of the three-part switched split vector quantiser at 24 bits/frame as a function of the number of switch directions using different bit allocations

m	Total bits/frame $(b_m + b_1 + b_2 + b_3)$	Avg. SD (in dB)	Outliers (in %)		kflops/frame
			2–4 dB	>4 dB	
1	24 (0 + 8 + 8 + 8)	1.061	1.68	0.01	10.30
2	24 (1 + 8 + 8 + 7)	**0.982**	0.97	0.01	8.27
	24 (1 + 8 + 7 + 8)	1.057	1.97	0.01	8.78
	24 (1 + 7 + 8 + 8)	1.019	1.15	0.01	8.78
4	24 (2 + 8 + 7 + 7)	1.006	1.39	0.00	6.81
	24 (2 + 7 + 8 + 7)	1.084	1.95	0.00	6.81
	24 (2 + 7 + 7 + 8)	1.031	1.55	0.00	7.32
	24 (2 + 8 + 8 + 6)	**0.968**	0.97	0.00	7.32
8	24 (3 + 8 + 7 + 6)	0.954	0.95	0.00	5.95
	24 (3 + 8 + 8 + 5)	**0.950**	1.01	0.00	6.97
16	24 (4 + 7 + 7 + 6)	**0.925**	0.75	0.00	4.73
	24 (4 + 8 + 8 + 4)	0.979	1.75	0.00	7.04
	24 (4 + 8 + 7 + 5)	0.948	1.02	0.00	5.76
32	24 (5 + 7 + 7 + 5)	**0.921**	0.86	0.00	4.86
	24 (5 + 7 + 6 + 6)	0.936	0.85	0.00	4.60
	24 (5 + 8 + 6 + 5)	0.959	1.20	0.00	5.63

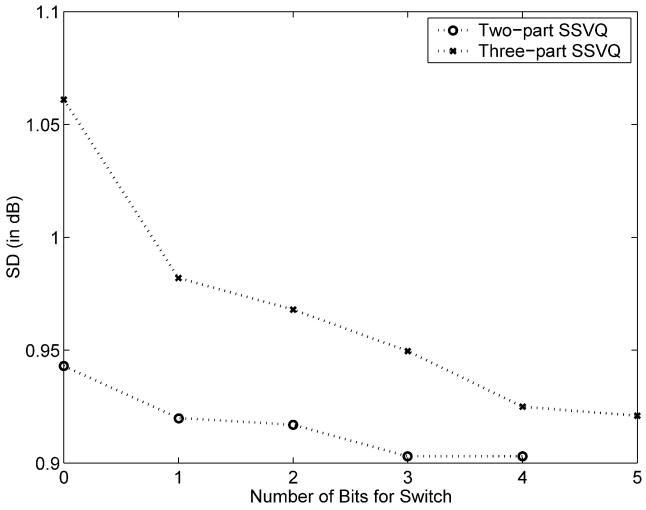

Fig. 19. Average spectral distortion (SD) of two-part and three-part SSVQ at 24 bits/frame as a function of the bits required for switching.

As well as this, there is a reduction in the number of computations required, which is shown in Fig. 20. By using three-part split vector quantisers in the SSVQ, we can reduce the number of computations by as much as 90% in exchange for only a modest increase in spectral distortion. Therefore, for the same bitrate, the distortion and computational complexity can be reduced by using more switching directions. We should note that there is a saturation of the distortion for large values of m because there are an insufficient number of training vectors.

4.5.2. Performance as a function of bitrate

Table 3 shows the spectral distortion and computational performance of the two-part switched split vector quantiser as a function of bitrate. It can be observed that transparent coding was achieved using 22 bits while requiring 17.7 kflops for each LSF frame. Table 4 shows the performance of the three-part SSVQ where transparent coding was achieved using 23 bits/frame.

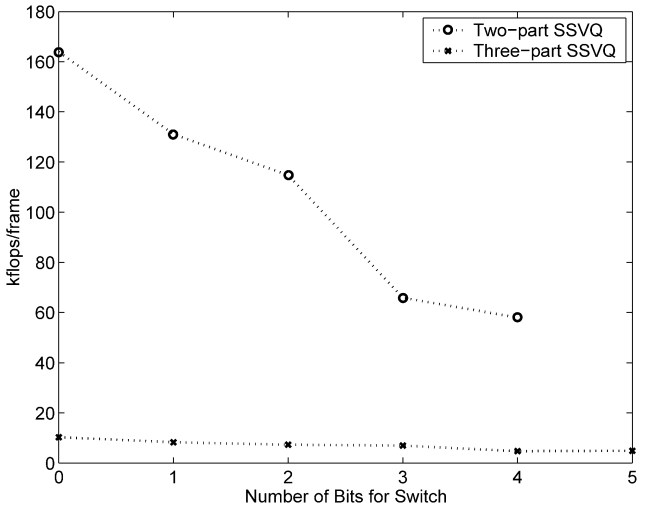

Fig. 20. Computational complexity of two-part and three-part SSVQ at 24 bits/frame as a function of the bits required for switching.

Table 3
Average spectral distortion (SD), computational complexity, and memory requirements (ROM) of the two-part switched split vector quantiser as a function of bitrate and number of switch directions

m	Total bits/frame $(b_m + b_1 + b_2)$	Avg. SD (in dB)	Outliers (in %)		kflops/frame	ROM (floats)
			2–4 dB	>4 dB		
8	24 (3 + 10 + 11)	0.902	0.43	0.00	65.9	131,152
	23 (3 + 10 + 10)	0.973	0.84	0.00	41.3	82,000
	22 (3 + 9 + 10)	1.031	1.20	0.00	33.1	65,616
16	24 (4 + 10 + 10)	0.912	0.57	0.00	41.6	164,000
	23 (4 + 9 + 10)	0.965	0.78	0.00	33.4	131,232
	22 (4 + 9 + 9)	1.038	1.29	0.00	21.1	82,080
32	24 (5 + 9 + 10)	0.903	0.52	0.00	34.0	262,464
	23 (5 + 9 + 9)	0.972	0.94	0.00	21.8	164,160
	22 (5 + 8 + 9)	1.029	1.26	0.00	17.7	131,392

Table 4
Average spectral distortion (SD), computational complexity, and memory requirements (ROM) of the three-part switched split vector quantiser as a function of bitrate and number of switch directions

m	Total bits/frame $(b_m + b_1 + b_2 + b_3)$	Avg. SD (in dB)	Outliers (in %)		kflops/frame	ROM (floats)
			2–4 dB	>4 dB		
8	24 (3+7+7+7)	0.955	0.90	0.00	5.44	10,320
	23 (3+7+7+6)	1.006	1.21	0.00	4.41	8272
	22 (3+7+6+6)	1.112	2.46	0.01	3.64	6736
16	24 (4+7+7+6)	0.925	0.75	0.00	4.73	16,544
	23 (4+7+6+6)	1.002	1.38	0.00	3.96	13,472
	22 (4+6+6+6)	1.080	1.97	0.01	3.20	10,400
32	24 (5+7+7+5)	0.921	0.86	0.00	4.86	28,992
	23 (5+6+6+6)	0.991	1.13	0.00	3.84	20,800
	22 (5+6+6+5)	1.049	1.70	0.00	3.32	16,704

Also shown in Tables 3 and 4 are the memory requirements (ROM) of SSVQ. It can be observed that while SSVQ achieves gains in quantiser performance with reduced computational complexity, they are at the expense of larger memory requirements. Three-part SSVQ with 8 switching directions at 23 bits/frame is therefore, the best coder

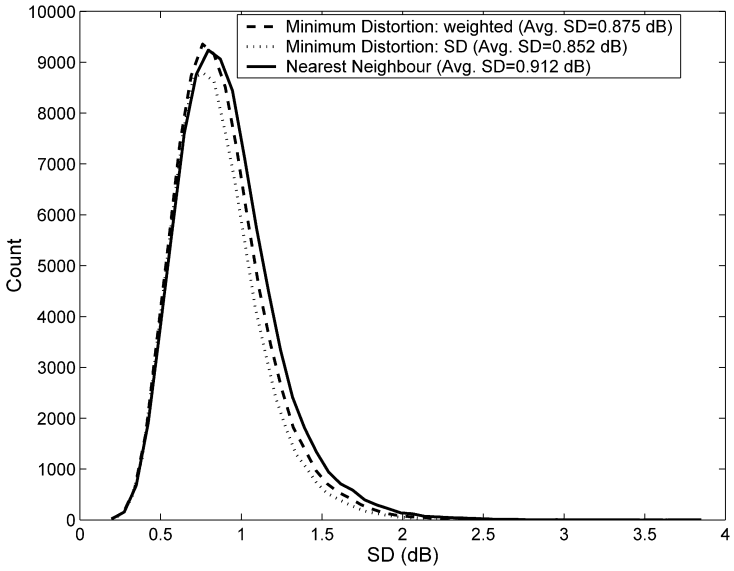

Fig. 21. Average spectral distortion (SD) histograms for the 24 bits/frame two-part switched split vector quantiser ($m = 16$) using minimum distortion (using weighted distance measure and spectral distortion) and nearest neighbour selection.

configuration with a good balance of low spectral distortion (1.006 dB), computational complexity (4.41 kflops/frame) and memory requirements (8272 floats).

4.5.3. Minimum distortion versus nearest neighbour selection

It was mentioned earlier that optimal performance, in terms of spectral distortion, can be achieved by quantising a vector using all split vector quantisers and then choosing the one which incurs the least distortion. However, such a scheme would involve a large amount of computations as each vector to be coded needs to be quantised multiple times in order to find the best representation. Therefore, we adopted a classification approach where each Voronoi region, from which split vector quantisers are designed, is represented by its centroid, and a switching decision, based on the nearest neighbour criteria, is made before it is quantised by the corresponding split vector quantiser. We have argued that the penalty in spectral distortion would be more than offset by the reduction in computational complexity. Figure 21 shows the histograms of the spectral distortion from SSVQ operating at 24 bits/frame using minimum distortion (weighted distance measure and spectral distortion) and nearest neighbour selection. As expected, the spectral distortion version of minimum distortion selection gives the lowest distortion (0.852 dB). However the resulting quantiser is very computationally intensive since m spectral distortion calculations need to performed for each vector to be coded. The weighted MSE version of minimum distortion selection gave a slightly higher average spectral distortion (0.875 dB) while the spectral distortion of the nearest neighbour selection was the worst out of the three (0.912 dB). However, when we compare the number of computations required in minimum distortion selection (approx. 655 kflops/frame) compared with the nearest neighbour selection (42 kflops/frame), we see the computational complexity reduced by 15 times at the expense of an extra 0.04–0.06 dB. It is clear that the nearest neighbour version of SSVQ is the best in terms of the distortion versus computational complexity trade-off.

4.5.4. Unstable LSF frames in the SSVQ with no ordering constraint

The correct ordering of the LSF coefficients for each frame is important as this ensures that the synthesis filter is stable. Because the sub-vectors are quantised independently in SVQ/SSVQ, there is a chance that the ascending order property may be violated in the reconstructed frame, especially when operating at lower bitrates. In our implementation, we have added an ordering constraint to prevent this. However, it is useful to examine the amount of unstable frames as a result of SSVQ with no ordering constraint.

Tables 5 and 6 list the number of unstable frames for the two and three-part SSVQ, respectively. We observe that SSVQ produces only a small percentage of unstable frames (at most 0.02% of the 85,353 frames), with three-part

Table 5
Number of unstable frames from the two-part switched split vector quantiser with no ordering constraint

m	Total bits/frame ($b_m + b_1 + b_2 + b_3$)	Number of unstable frames
8	24 (3 + 10 + 11)	4
	23 (3 + 10 + 10)	10
	22 (3 + 9 + 10)	13
16	24 (4 + 10 + 10)	6
	23 (4 + 9 + 10)	7
	22 (4 + 9 + 9)	11
32	24 (5 + 9 + 10)	2
	23 (5 + 9 + 9)	7
	22 (5 + 8 + 9)	7

Table 6
Number of unstable frames from the three-part switched split vector quantiser with no ordering constraint

m	Total bits/frame ($b_m + b_1 + b_2 + b_3$)	Number of unstable frames
8	24 (3+7+7+7)	14
	23 (3+7+7+6)	17
	22 (3+7+6+6)	20
16	24 (4+7+7+6)	6
	23 (4+7+6+6)	9
	22 (4+6+6+6)	10
32	24 (5+7+7+5)	2
	23 (5+6+6+6)	3
	22 (5+6+6+5)	6

SSVQ generally producing more unstable frames when $m = 8$. Also, we can see that more unstable frames result as the bitrate is lowered. Finally, the number of unstable frames produced decreases as we use more switching directions, m.

4.6. Comparison with other intraframe quantisation schemes

4.6.1. Comparison with split vector quantisers

Tables 7 and 8 show the performance of the two-part and three-part split vector quantiser, described in [11], on the TIMIT database, respectively. By comparing Tables 3 with 7 and Tables 4 with 8, we can see that SSVQ outperforms SVQ for all bitrates. The 23 bits/frame three-part SSVQ is comparable to a three-part split vector quantiser operating at 25 bits/frame while requiring only about 33% of the number of computations of the latter. This shows the effectiveness of SSVQ in exploiting correlation between subspaces for lower distortion performance while requiring less computations, at the same bitrate.

In general, SSVQ requires more codebook memory than SVQ. However, the 23 bits/frame three-part SSVQ has slightly better spectral distortion than the 23 bits/frame two-part SVQ but with a third of the memory requirements (8272 cf. 28,672 floats) and a fraction of the complexity (4.4 cf. 114 kflops/frame).

4.6.2. Comparison with multistage vector quantisers

Table 9 lists the results of the sequentially-searched multistage vector quantiser (MSVQ) on the TIMIT database where transparent coding has been achieved at 22 and 24 bits/frame for two and three stages, respectively. Comparing the two-stage MSVQ with the two-part SSVQ in Table 4, we observe comparable spectral distortion performance. However, the two-stage MSVQ is considerably more complex (163.8 cf. 33.1 kflops/frame) than the two-part SSVQ at 22 bits/frame, though the latter has higher memory requirements. When we compare the three-stage MSVQ with the three-part SSVQ, we observe that the latter achieves slightly lower spectral distortion for all bitrates as well as lower percentages of outlier frames that have an SD of between 2 and 4 dB. The three-part SSVQ also has lower computational complexity than the three-stage MSVQ.

Table 7
Average spectral distortion (SD), computational complexity, and memory requirements (ROM) of the two-part split vector quantiser as a function of bitrate

Bits/frame $(b_1 + b_2)$	Avg. SD (in dB)	Outliers (in %)		kflops/frame	ROM (floats)
		2–4 dB	>4 dB		
24 (12 + 12)	0.943	0.54	0.00	163.8	40,960
23 (12 + 11)	1.023	1.09	0.00	114.7	28,672
22 (11 + 11)	1.080	1.44	0.00	81.9	20,480

Table 8
Average spectral distortion (SD), computational complexity, and memory requirements (ROM) of the three-part split vector quantiser as a function of bitrate

Bits/frame $(b_1 + b_2 + b_3)$	Avg. SD (in dB)	Outliers (in %)		kflops/frame	ROM (floats)
		2–4 dB	>4 dB		
26 (9 + 9 + 8)	0.892	0.55	0.00	16.4	4096
25 (9 + 8 + 8)	1.001	1.38	0.00	13.3	3328
24 (8 + 8 + 8)	1.061	1.68	0.01	10.2	2560

Table 9
Average spectral distortion (SD), computational complexity, and memory requirements (ROM) of the sequentially-searched multistage vector quantiser as a function of bitrate and number of stages

Number of stages	Bits/frame	Avg. SD (in dB)	Outliers (in %)		kflops/frame	ROM (floats)
			2–4 dB	>4 dB		
2	24	0.908	0.68	0.00	327.7	81,920
	23	0.967	1.06	0.00	245.8	61,440
	22	1.038	1.58	0.00	163.8	40,960
3	24	0.999	1.65	0.00	30.7	7680
	23	1.064	2.42	0.00	25.6	6390
	22	1.135	3.41	0.00	20.5	5100

Table 10
Average spectral distortion (SD), computational complexity, and memory requirements (ROM) of the M–L searched multistage vector quantiser at 24 bits/frame

Bits/frame (type)	M	Avg. SD (in dB)	Outliers (in %)		kflops/frame	ROM (floats)
			2–4 dB	>4 dB		
24 (256–3)	1	0.999	1.65	0.00	30.7	7680
	2	0.951	1.14	0.00	51.2	7680
	3	0.933	1.03	0.00	71.7	7680
	4	0.925	0.98	0.00	92.2	7680
24 (64–4)	1	1.087	2.99	0.01	10.2	2560
	2	1.029	2.09	0.00	17.2	2560
	3	1.011	1.87	0.00	25.6	2560
	4	1.003	1.79	0.00	33.3	2560

The poor performance of the three-stage MSVQ may be attributed to the suboptimality of the sequential codebook search. Table 10 lists the spectral distortion performance of the M–L searched MSVQ for different numbers of preserved paths, M, at 24 bits/frame. Three-stage and four-stage M–L searched MSVQ[16] are considered, with the latter configuration (64–4) considered to be the best tradeoff between complexity and spectral distortion performance [9]. As M is increased, we see that the spectral distortion decreases, accompanied by increases in complexity.

[16] As was done in [9], the configuration of the MSVQ is denoted by (codebook size, number of stages).

Table 11
Average spectral distortion (SD) performance of non-uniform scalar quantisers for different bitrates

Bits/frame	Avg. SD (in dB)	Outliers (in %)	
		2–4 dB	>4 dB
34	0.925	0.89	0.01
33	1.001	1.30	0.01
32	1.002	1.32	0.01
31	1.153	3.68	0.01

Table 12
Average spectral distortion (SD) performance of scalar quantisers operating at 34 bits/frame in the FS-1016 4.8 kbps CELP coder

Bits/frame	Avg. SD (in dB)	Outliers (in %)	
		2–4 dB	>4 dB
34	1.44	10.24	0.02

We now compare Tables 4 and 10. The average spectral distortion of the 8 switch SSVQ at 24 bits/frame is about the same as the (256–3) M–L searched MSVQ with $M = 2$, with the SSVQ achieving a lower percentage of outlier frames (0.9% cf. 1.65%) and possessing a lower computational complexity (5.44 cf. 51.2 kflops/frame). Similarly, the 16 switch SSVQ at 24 bits/frame incurs the same average spectral distortion as the (256–3) M–L searched MSVQ with $M = 4$, but with less outlier frames (0.75% cf. 0.98%) and considerably lower complexity (4.73 cf. 92.2 kflops/frame). However, SSVQ has higher memory requirements than the MSVQ in all cases. Finally, we note that the M–L search MSVQ with the best tradeoff (64–4), still has more than twice the computational complexity of the three-part SSVQ at 24 bits/frame.

4.6.3. Comparison with PDF-optimised non-uniform scalar quantisers

Table 11 shows the spectral distortion performance of independent, non-uniform scalar quantisers. The generalised Lloyd algorithm (GLA) [14] is used to design optimum levels for each LSF. We use a greedy bit allocation algorithm, based on the one proposed in [55], where zero bits are initially assigned to each quantiser. One bit is given to the quantiser which results in the maximum reduction in quantiser distortion and the process is continued until all bits are allocated. A spectral distortion of 1 dB is achieved at 32 bits/frame. It can be seen that the 23 bits/frame three-part SSVQ is comparable in performance to scalar quantisers operating at 32–34 bits/frame. Table 12 shows the performance of the non-uniform scalar quantisers that are used in the US Federal Standard 1016, 4.8 kbps CELP coder [52]. It is clear that SSVQ performs considerably better than the scalar quantisers in the FS-1016 4.8 kbps CELP coder.

4.6.4. Comparison with GMM-based block quantisers

Subramaniam and Rao [43] proposed a quantisation scheme where the probability density function (PDF) of the source is modelled as a Gaussian mixture model and independent block quantisers are designed for each Gaussian mixture component. Table 13 presents the spectral distortion performance of this quantisation scheme[17] on the TIMIT database. The rate independent computational complexity[18] and memory requirements are also included in the table. The GMM consists of 16 clusters, initialised by the LBG algorithm [14], followed by 20 iterations of the expectation-maximisation (EM) algorithm. Comparing Table 13 and 4, the 23 bits/frame three-part SSVQ gives lower spectral distortion than the GMM-based block quantiser operating at 24 bits/frame.

[17] In contrast to Subramaniam and Rao's implementation of the scalar quantisers, where they used a combination of compander lookup tables and a uniform scalar quantiser, we have used a lookup table of reproduction levels (up to 8 bits) that were derived using the iterative Lloyd method I for a Gaussian distribution. This lookup table consists of 252 floats and has been included in the ROM calculations.

[18] Note that we have assumed the use of the full spectral distortion calculation, rather than the approximation given in [56].

Table 13
Average spectral distortion (SD) performance with rate independent memory requirements (ROM) and computational complexity of the 16 cluster, memoryless, fixed rate GMM-based block quantiser for different bitrates

Bits/frame	Avg. SD (in dB)	Outliers (in %)		kflops/frame	ROM (floats)
		2–4 dB	>4 dB		
25	0.987	0.98	0.00	253.9	2588
24	1.049	1.46	0.00	253.9	2588
23	1.111	2.27	0.00	253.9	2588

This quantisation scheme, as we have reported in [57], can be particularly intensive in computation when using the full spectral distortion calculation, which requires at least 15.27 kflops/frame.[19] As an informal comparison of computational performance, we measured the times it took to quantise LSF vectors at 24 bits/frame using the 16 cluster GMM-based block quantiser and three-part SSVQ (8 switching directions). The platform used to perform the test was a 2.4 GHz, Intel Pentium 4 processor running the Linux operating system. The 24 bits/frame GMM-based block quantiser took, on average, 150 s to quantise 85353 LSF vectors (SD of 1.049 dB) while the 24 bits/frame three-part SSVQ took 7 s (SD of 0.955 dB). It should be noted, though, that a fairer comparison of computational complexity between the two schemes should include the use of the Gardner–Rao [56] approximation of spectral distortion. However, our primary aim was to evaluate the GMM-based block quantiser with the best rate-distortion efficiency, hence our preference for the full spectral distortion calculation over the Gardner–Rao approximation.

5. Conclusions and future work

This article has provided a general review of vector quantisation, its advantages over the scalar quantiser, and its limitations, with regards to its exponential growth of complexity as a function of the number of bits and dimensionality. Product code vector quantisers, such as the split and multistage vector quantiser, alleviate the complexity issue by dividing the quantisation process, into codebooks of lower dimensionality, or sequential and independent stages, respectively. These structural constraints though cause suboptimal quantisation performance. We have also identified and analysed the main source of suboptimality in the split vector quantiser (SVQ), namely the vector splitting which degrades the memory advantage, the shape advantage, and the space-filling advantage. In order to address at least two of these suboptimalities, we have introduced a new type of product code vector quantiser called the switched split vector quantiser (SSVQ), which consists of a hybrid of a full-dimension, unconstrained switch vector quantiser and numerous split vector quantisers. The first stage (i.e., switch vector quantiser) allows the SSVQ to exploit global statistical dependencies as well as match the marginal PDF shape of the data, which would otherwise have not been exploited by normal SVQ. Also, the tree structured characteristic of the switch vector quantiser provides a dramatic reduction in search complexity. We have shown via computer simulations of 2-D vector quantisation and LPC parameter quantisation experiments, how SSVQ is superior to SVQ and other quantisation schemes in terms of quantisation performance and computational complexity. The only disadvantage of SSVQ is the increase in memory requirements.

Future work will include an algorithm for adaptively determining appropriate subvector dimensions for each local SVQ. As mentioned, the vector splitting is the cause of suboptimality in SVQ and in the current SSVQ scheme, the same vector splitting is used for local SVQs. It is hoped that variable vector splitting in the local SVQs will allow more flexibility in terms of exploiting local dependencies and PDF shape. Also, a partial soft-decision scheme, where more than one switch is chosen, is expected to give better performance for only a moderate increase in computational complexity.

[19] The spectral distortion involves calculating the root-mean-squared-error between the log periodograms of both the original and quantised LPC coefficients. Therefore, two 256-point split-radix FFTs are used, each expending 6664 flops, according to Table 6.2 of [58]. The reciprocal of the squared magnitude is calculated for each FFT, requiring an additional 516 flops, which followed by the decibel calculation, brings the total computations to 14.876 kflops. The RMS calculation adds an extra 389 flops. The number of flops required for the square root and logarithm calculation are not known and are assumed to be 1 flop, which brings the total computational complexity of the spectral distortion calculation to approximately 15.27 kflops.

Acknowledgments

The authors wish to thank the three anonymous reviewers for their invaluable comments and feedback, which have helped to improve the quality and clarity of this paper.

References

[1] C.E. Shannon, Coding theorems for a discrete source with a fidelity criterion, IRE National Convention Record, 1959, part 4, pp. 142–163.
[2] R.M. Gray, D.L. Neuhoff, Quantization, IEEE Trans. Inform. Theory 44 (6) (1998) 2325–2383.
[3] T.D. Lookabaugh, R.M. Gray, High-resolution quantization theory and the vector quantizer advantage, IEEE Trans. Inform. Theory 35 (5) (1989) 1020–1033.
[4] Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantizer design, IEEE Trans. Commun. COM-28 (1) (1980) 84–95.
[5] A. Buzo, A.H. Gray Jr., R.M. Gray, J.D. Markel, Speech coding based upon vector quantization, IEEE Trans. Acoust. Speech Signal Process. 28 (1980) 562–574.
[6] A. Gersho, B. Ramamurthi, Image coding using vector quantization, in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, 1982, pp. 428–431.
[7] R.M. Gray, Vector quantization, IEEE ASSP Mag. 1 (1984) 4–29.
[8] B.H. Juang, A.H. Gray Jr., Multiple stage vector quantisation for speech coding, in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, vol. 1, 1982, pp. 597–600.
[9] W.P. LeBlanc, B. Bhattacharya, S.A. Mahmoud, V. Cuperman, Efficient search and design procedures for robust multi-stage VQ for LPC parameters for 4 kb/s speech coding, IEEE Trans. Speech Audio Process. 1 (4) (1993) 373–385.
[10] J. Makhoul, S. Roucos, H. Gish, Vector quantization in speech coding, Proc. IEEE 73 (1985) 1551–1588.
[11] K.K. Paliwal, B.S. Atal, Efficient vector quantization of LPC parameters at 24 bits/frame, IEEE Trans. Speech Audio Process. 1 (1) (1993) 3–14.
[12] M.J. Sabin, R.M. Gray, Product code vector quantizers for waveform and voice coding, IEEE Trans. Acoust. Speech Signal Process. ASSP-32 (3) (1984) 474–488.
[13] S. So, K.K. Paliwal, Efficient vector quantisation of line spectral frequencies using the switched split vector quantiser, in: Proc. Int. Conf. Spoken Language Processing, October 2004.
[14] A. Gersho, R.M. Gray, Vector Quantization and Signal Compression, Kluwer Academic, Dordrecht/Norwell, MA, 1992.
[15] K. Sayood, Introduction to Data Compression, Morgan Kaufmann Publishers, San Francisco, 1996.
[16] P.C. Mahalanobis, On the generalized distance in statistics, in: Proc. Indian Nat. Inst. Sci. Calcutta, vol. 2, 1936, pp. 49–55.
[17] F. Itakura, Minimum prediction residual principle applied to speech recognition, IEEE Trans. Acoust. Speech Signal Process. ASSP-23 (1) (1975) 67–72.
[18] S.P. Lloyd, Least square quantization in PCM, IEEE Trans. Inform. Theory IT-28 (2) (1982) 129–137.
[19] J. Max, Quantising for minimum distortion, IRE Trans. Inform. Theory IT-6 (1960) 7–12.
[20] M.D. Paez, T.H. Glisson, Minimum mean-squared-error quantization in speech PCM and DPCM systems, IEEE Trans. Commun. COM-20 (1972) 225–230.
[21] W.R. Bennett, Spectra of quantized signals, Bell Syst. Tech. J. 27 (1948) 446–472.
[22] A. Gersho, Asymptotic optimal block quantization, IEEE Trans. Inform. Theory IT-25 (1979) 373–380.
[23] P.A. Wintz, Transform picture coding, Proc. IEEE 60 (7) (1972) 809–820.
[24] G.K. Wallace, The JPEG still picture compression standard, Commun. ACM 34 (4) (1991) 30–44.
[25] J.J.Y. Huang, P.M. Schultheiss, Block quantization of correlated Gaussian random variables, IEEE Trans. Commun. Syst. CS-11 (1963) 289–296.
[26] A.N. Netravali, J.O. Limb, Picture coding: A review, Proc. IEEE 68 (3) (1980) 366–406.
[27] P. Hedelin, J. Skoglund, Vector quantization based on Gaussian mixture models, IEEE Trans. Speech Audio Process. 8 (4) (2000) 385–401.
[28] B. Ramamurthi, A. Gersho, Classified vector quantization of images, IEEE Trans. Comm. COM-34 (1986) 1105–1115.
[29] S. Wang, A. Gersho, Phonetically-based vector excitation coding of speech at 3.6 kbit/s, in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, Glasgow, May 1989, pp. I-349–I-352.
[30] N.P. Koestoer, Robust linear prediction analysis for speech coding, PhD dissertation, Griffith University, 2002.
[31] K.K. Paliwal, B.S. Atal, Efficient vector quantisation of LPC parameters at 24 bits/frame, in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, 1991, pp. 661–664.
[32] 3rd generation partnership project; Technical specification group services and system aspects; Mandatory speech codec speech processing functions; Adaptive multi-rate (AMR) speech codec; Transcoding functions (Release 5), Technical Specification TS 26.090, 3rd Generation Partnership Project (3GPP), June 2002.
[33] Speech processing, transmission and quality aspects (STQ); Distributed speech recognition; Front-end feature extraction algorithm; Compression algorithms, Tech. Rep. Standard ES 201 108, European Telecommunications Standards Institute (ETSI), April 2000.
[34] F. Nordén, T. Eriksson, On split quantization of LSF parameters, in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, Montreal, 2004, pp. I-157–I-160.
[35] A.M. Smith, J.P. Ashley, M.A. Jasiuk, W. Peng, Normalization and polygon error detection for split VQ of line spectral frequencies, in: IEEE Speech Coding Workshop, 2000, pp. 125–127.

[36] V. Krishnan, D.V. Anderson, K.K. Truong, Optimal multistage vector quantization of LPC parameters over noisy channels, IEEE Trans. Speech Audio Process. 12 (1) (2004) 1–8.
[37] W.-Y. Chan, S. Gupta, A. Gersho, Enhanced multistage vector quantization by joint codebook design, IEEE Trans. Commun. 40 (11) (1992) 1693–1697.
[38] J. Pan, T.R. Fischer, Vector quantization-lattice vector quantization of speech LPC coefficients, in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, vol. 1, 1994, pp. 513–516.
[39] J. Pan, Two-stage vector quantization-pyramidal lattice vector quantization and application to speech LSP coding, in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, vol. 2, 1996, pp. 737–740.
[40] J. Zhou, Y. Shoham, A. Akansu, Simple fast vector quantization of the line spectral frequencies, in: Proc. ICSLP '96, vol. 2, 1996, pp. 945–948.
[41] Draft recommendation G. 729: Coding of speech at 8 kbits/s using conjugate-structure algebraic code-excited linear prediction (CS-ACELP), ITU-T, 1995.
[42] 3rd generation partnership project; Technical specification group services and system aspects; speech codec speech processing functions; AMR wideband speech codec; Transcoding functions (Release 5), Technical Specification TS 26.190, 3rd Generation Partnership Project (3GPP), December 2001.
[43] A.D. Subramaniam, B.D. Rao, PDF optimized parametric vector quantization of speech line spectral frequencies, IEEE Trans. Speech Audio Process. 11 (2) (2003) 130–142.
[44] W.B. Mikhael, V. Krishnan, Energy-based split vector quantizer employing signal representation in multiple transform domains, Digital Signal Process. 11 (4) (2001) 359–370.
[45] F. Itakura, S. Saito, Speech analysis-synthesis based on the partial autocorrelation coefficient, Proc. JSA (1969) 199–200.
[46] A. Gray, J. Markel, Quantization and bit allocation in speech processing, IEEE Trans. Acoust. Speech Signal Process. ASSP-24 (1976) 459–473.
[47] R. Viswanathan, J. Makhoul, Quantization properties of transmission parameters in linear predictive systems, IEEE Trans. Acoust. Speech Signal Process. ASSP-23 (1975) 309–321.
[48] F. Itakura, Line spectrum representation of linear predictive coefficients of speech signals, J. Acoust. Soc. Amer. 57 (1975) S35.
[49] F.K. Soong, B.H. Juang, Line Spectrum Pair (LSP) and Speech Data Compression, in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, San Diego, CA, March 1984, pp. 37–40.
[50] N. Sugamura, F. Itakura, Speech analysis and synthesis methods developed at ECL in NTT—From LPC to LSP, Speech Commun. 5 (1986) 199–215.
[51] B.S. Atal, M.R. Schroeder, Predictive coding of speech signals and subjective error criteria, IEEE Trans. Acoust. Speech Signal Process. Astrophys. Space. Sci. P-27 (3) (1979) 247–254.
[52] J.P. Campbell Jr., V.C. Welch, T.E. Tremain, An expandable error-protected 4800 bps CELP coder (US Federal Standard 4800 bps voice coder), in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, Glasgow, 1989, pp. 735–738.
[53] K.K. Paliwal, W.B. Kleijn, Quantization of LPC parameters, in: W.B. Kleijn, K.K. Paliwal (Eds.), Speech Coding and Synthesis, Elsevier, Amsterdam, 1995, pp. 443–466.
[54] P. Kroon, W.B. Kleijn, Linear-prediction based analysis-by-synthesis coding, in: W.B. Kleijn, K.K. Paliwal (Eds.), Speech Coding and Synthesis, Elsevier, Amsterdam, 1995, pp. 79–119.
[55] F.K. Soong, B.H. Juang, Optimal quantization of LSP parameters, in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, New York, April 1988, pp. 394–397.
[56] W.R. Gardner, B.D. Rao, Theoretical analysis of the high-rate vector quantization of LPC parameters, IEEE Trans. Speech Audio Process. 3 (1995) 367–381.
[57] K.K. Paliwal, S. So, Multiple frame block quantisation of line spectral frequencies using Gaussian mixture models, in: Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, Montreal, 2004, pp. I-149–I-152.
[58] J.G. Proakis, D.G. Manolakis, Digital Signal Processing: Principles, Algorithms, and Applications, third ed., Prentice Hall, NJ, 1996.

Stephen So was born in Hong Kong in 1977. He received the BEng (Hons) in 1999 and the PhD degree in 2005, both from the School of Microelectronic Engineering, Griffith University, Australia. He is currently working as an Associate Lecturer in electronic engineering in the School of Microelectronic Engineering, Griffith University, Australia. His research interests include block and vector quantisation schemes, image coding, speech coding, audio coding, distributed speech recognition, and face recognition.

Kuldip K. Paliwal was born in Aligarh, India, in 1952. He received the BS degree from Agra University, Agra, India, in 1969, the MS degree from Aligarh Muslim University, Aligarh, India, in 1971, and the PhD degree from Bombay University, Bombay, India, in 1978.

He has been carrying out research in the area of speech processing since 1972. He has worked at a number of organizations including Tata Institute of Fundamental Research, Bombay, India; Norwegian Institute of Technology, Trondheim, Norway; University of Keele, UK; AT&T Bell Laboratories, Murray Hill, NJ, USA; AT&T Shannon Laboratories, Florham Park, NJ, USA; and Advanced Telecommunication Research Laboratories, Kyoto, Japan. Since July 1993, he has been a Professor at Griffith University, Brisbane, Australia, in the School of Microelectronic Engineering. His current research interests include speech recognition, speech coding, speaker recognition, speech enhancement, face recognition, image coding, pattern recognition, and artificial neural networks. He has published more than 250 papers in these research areas.

Dr. Paliwal is a Fellow of Acoustical Society of India. He has served the IEEE Signal Processing Society's Neural Networks Technical Committee as a Founding Member from 1991 to 1995 and the Speech Processing Technical Committee from 1999 to 2003. He was an Associate Editor of the IEEE Transactions on Speech and Audio Processing during the periods 1994–1997 and 2003–2004. He also served as an Associate Editor of the IEEE Signal Processing Letters from 1997 to 2000. He was the General Co-Chair of the Tenth IEEE Workshop on Neural Networks for Signal Processing (NNSP 2000). He has co-edited two books: "Speech Coding and Synthesis" (published by Elsevier), and "Speech and Speaker Recognition: Advanced Topics" (published by Kluwer). He has received IEEE Signal Processing Society's best (senior) paper award in 1995 for his paper on LPC quantization. He is currently serving the Speech Communication Journal (published by Elsevier) as its Editor-in-Chief.