Constrained-Storage Vector Quantization with a Universal Codebook

Sangeeta Ramakrishnan, Kenneth Rose, *Member, IEEE,* and Allen Gersho, *Fellow, IEEE*

***Abstract*— Many image compression techniques require the quantization of multiple vector sources with significantly different distributions. With vector quantization (VQ), these sources are optimally quantized using separate codebooks, which may collectively require an enormous memory space. Since storage is limited in most applications, a convenient way to gracefully trade between performance and storage is needed. Earlier work addressed this problem by clustering the multiple sources into a small number of source groups, where each group shares a codebook. We propose a new solution based on a size-limited universal codebook that can be viewed as the union of overlapping source codebooks. This framework allows each source codebook to consist of any desired subset of the universal codevectors and provides greater design flexibility which improves the storage-constrained performance. A key feature of this approach is that no two sources need be encoded at the same rate. An additional advantage of the proposed method is its close relation to universal, adaptive, finite-state and classified quantization. Necessary conditions for optimality of the universal codebook and the extracted source codebooks are derived. An iterative design algorithm is introduced to obtain a solution satisfying these conditions. Possible applications of the proposed technique are enumerated, and its effectiveness is illustrated for coding of images using finite-state vector quantization, multistage vector quantization, and tree-structured vector quantization.**

***Index Terms*—Adaptive quantization, constrained storage, universal codebook, universal source coding, vector quantization.**

I. Introduction

A. Motivation and Applications

VECTOR quantization (VQ) is an appealing coding technique because the rate-distortion bound can be approached by increasing vector dimension [1]. In applications where high reproduction quality is required, relatively high bit rates are needed. This can lead to very high complexity for such applications, because the computational and storage complexity of unstructured vector quantization grows exponentially with the rate and dimension. In applications where signals from multiple sources are to be encoded, each would normally require separate quantization codebooks, thus further compounding the problem of storage. As a result, codebook storage can become the dominant complexity obstacle.

Constraining the amount of stored data tables in VQ-based coding systems while designing the tables to obtain optimal performance has several applications. In *adaptive VQ* (AVQ) [2], [3], updated codebooks need to be communicated to the receiver periodically. Here, the bit rate overhead incurred in transmitting codebooks can be greatly reduced by having a fixed “universal” codebook in the receiver from which new codebooks can be specified. If, further, fixed-rate coding is used to specify the new codebook, then the size of the universal codebook determines the cost in bit rate incurred in updating the codebook. Since the storage space of the universal codebook is a key limitation, a constrained storage approach offers an effective solution. In applications where different types of sources need to be encoded, the proposed scheme would be an excellent candidate. For example in image coding, where a wide variety of images like portraits, textures, medical images, text documents, etc., need to be encoded, the storage requirements can be reduced by use of a constrained-storage VQ scheme, rather than having individual codebooks for each type of image to be encoded.

In *tree-structured VQ* (TSVQ) [4], the key limitation is storage space, since VQ encoding complexity is greatly reduced by the tree structure. A family of structured VQ schemes that include TSVQ as a special case is based on *generalized product code VQ* (GPC-VQ), where a source is sequentially coded in stages and the codebook for each stage depends on the outcome of the previous stage [5]. The use of constrained-storage methods is essential for GPC-VQ. As a final example, we observe that in *classified VQ* (CVQ) [1], [6] and *finite-state VQ* (FSVQ) [7], [20], the critical performance limitation is the storage of a large number of class/state specific codebooks.

B. Relevant Prior Work

An earlier approach to mitigating the problem of storage complexity in VQ, called *constrained-storage VQ* (CSVQ) was introduced by Chan and Gersho and was based on the concept of *codebook sharing*. Given a set of N sources, a set of $M < N$ codebooks is designed, where each codebook is assigned to be shared by a group of sources, so as to optimize an overall performance measure. The basic theory is given in [8], and the approach is specialized for TSVQ codebook design in [10]. In this paper, we use the acronym *CSVQ* to refer more generally to VQ coding schemes that employ storage constraints rather than narrowly to the work of [8].

Manuscript received January 14, 1996; revised June 4, 1997. This work was supported in part by the National Science Foundation under Grant NCR-9314335, the University of California MICRO Program, DSP Group, Inc., Moseley Associates, National Semiconductor Corp., Qualcomm, Inc., Rockwell International Corporation, Cisco Systems, and Texas Instruments, Inc. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Michael W. Marcellin.

S. Ramakrishnan was with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA. She is now with Chromatic Research, Inc., Sunnyvale, CA 94089.

K. Rose and A. Gersho are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA (e-mail: gersho@ece.ucsb.edu).

Publisher Item Identifier S 1057-7149(98)04003-2.

Another approach suggested for AVQ by Gersho and Gray [1, p. 620] and studied for universal VQ by Zeger *et al.* [11], [12] is to identify a subset of codevectors from a fixed *universal codebook* that is chosen to represent the current statistics of the source. This problem is directly related to the one we address here, and further, by effectively designing a suitable universal codebook of limited size, we can provide a constrained-storage solution to AVQ.

Another study of relevance to CSVQ was reported by Lyons *et al.* [14]. In this work, the storage space of a large TSVQ codebook is reduced by quantizing the codevector and test vector components with a smaller secondary quantizer of low dimensionality (in most cases, a scalar quantizer is preferred). There are important ties between this approach and the one we take in this paper. We discuss these further in Section IV-B in the context of TSVQ design.

The problem of FSVQ design subject to storage constraints was addressed by Kim [13] and by Nasrabadi *et al.* [15]. Here, the high memory cost introduced by the use of a large number of states is overcome by the use of a "super-codebook." The state codebooks are extracted as subsets of this large super-codebook in a heuristic adaptive manner, without direct minimization of the distortion.

C. CSVQ with a Universal Codebook

In this work, we examine and solve the optimization problem for a general form of CSVQ which in most respects encompasses the prior work of Chan and Gersho [8] as a restricted special case, while offering a more versatile and powerful solution for several applications. We directly solve the problem of optimal VQ design under storage constraints without imposing any additional constraints on how memory should be shared by the sources. In particular, *memory can be shared by sources at the level of individual codevectors* rather than entire codebooks as proposed in [8]. Some aspects of our approach were presented in [9]. Since the method of Chan and Gersho can be viewed as a special case of our approach where groups of sources are constrained to share the exact same codebook, our CSVQ will always improve on (or at the theoretical worst-case be the same as) such a solution by removing these restrictions. Our framework provides more efficient storage and takes full advantage of the flexibility allowed by the storage constraint.

It is also important to note that this framework is directly applicable to the important case where different codebook sizes are used by different sources. The ability to assign different codebook sizes to individual sources makes the method attractive for applications where bit allocation is needed, for example, when the sources correspond to different components or features extracted from a given signal. In such cases, one can optimize the bit allocation, and still impose a practical storage constraint on the overall number of codevectors. This also has obvious application in structurally constrained VQ, where all the codebooks used for the various stages can be extracted from a universal codebook that satisfies the given storage limitations.

We note further that our approach is applicable to the context of adaptive and universal quantization (e.g., [11]), where adaptation is realized by extracting the current codebook from the universal codebook and transmitting the selection parameters to the decoder. In such applications, the size of the universal codebook has a direct impact on the cost in bit rate associated with specifying a codebook to the decoder, in addition to the obvious storage considerations. Our approach also finds application in FSVQ and CVQ where, typically, individual codebooks are used for each state or class. By adopting the proposed CSVQ approach, it is possible to increase the number of states or classes without an increase in memory requirements.

TSVQ is another structured vector quantizer which suffers from the problem of substantial memory requirements. An appropriate modification of our problem formulation leads to a method for TSVQ design that substantially reduces the memory requirements while maintaining the low computational complexity of standard TSVQ. Thus, the basic approach proposed here is applicable to a large variety of vector quantization schemes.

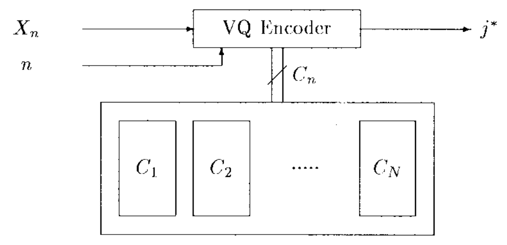
The organization of the paper is as follows. Section II gives the system description, some definitions, details about the encoding and decoding rules, and the formulation of the design problem. Section III presents the design procedure. Section IV provides the experimental results that validate the method and Section V gives our conclusions and future directions.

II. System Description and Problem Formulation

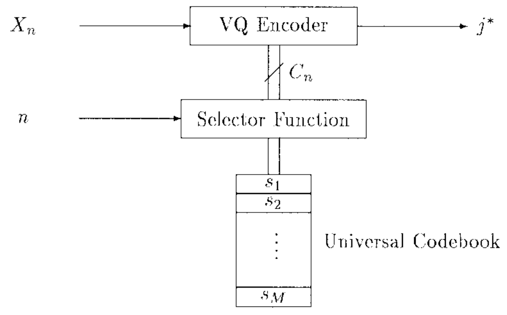
A variety of signal compression applications involving VQ can be modeled as using the encoder depicted in Fig. 1(a). A set of N stationary vector sources, where each source generates vectors of dimension k, are to be quantized under the same distortion measure with possibly different resolutions r_n (b/vector). At a given time, the encoder is presented with the observed value $\mathbf{x}_n$ of a particular source vector and the *label* or index n that identifies to which of the N sources it belongs. The encoder outputs a codeword of r_n bits, also referred to as the index for the source vector. The decoder receives the codeword and is assumed to know the label n; it then uses codebook C_n to produce $\hat{\mathbf{x}}$, an approximation of the source vector $\mathbf{x}_n$.

Generally, a separate codebook for each of the N sources would be required for best performance, where each VQ codebook is tailored for the specific source's statistics. Such a system is depicted in Fig. 1(a). To reduce storage requirements, however, codevector sharing is introduced, wherein there exists a universal codebook containing M codevectors and each source uses a particular subset of these vectors as its codebook. A careful choice of codebook, for each source, from the large number of possible codebooks is hence required. The selector function is designed to perform this choice. The CSVQ system is depicted in Fig. 1(b).

In many applications, the various sources do not need to be quantized at the same rate. CSVQ is very well suited for such applications, as the only constraint on the source codebook is that it be extracted from the universal codebook. In particular, we allow for different codebook sizes and hence different rates.

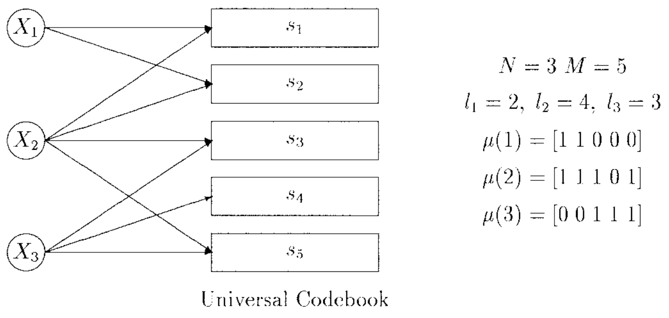


(a)



(b)

Fig. 1. Multiple source encoding using (a) separate source codebooks and (b) the CSVQ approach.



X_1
X_2
X_3
s_1
s_2
s_3
s_4
s_5
Universal Codebook
$N = 3 \; M = 5$
$l_1 = 2, \; l_2 = 4, \; l_3 = 3$
$\mu(1) = [1\;1\;0\;0\;0]$
$\mu(2) = [1\;1\;1\;0\;1]$
$\mu(3) = [0\;0\;1\;1\;1]$

Fig. 2. Example of a CSVQ scheme.

A. Definitions and Notation

We are given N sources, each generating a k dimensional random vector $\mathbf{X}_n \in \mathcal{R}^k$, for $n = 1, 2, \cdots, N$. A vector quantizer Q_n for the nth source is a mapping $Q_n : \mathcal{R}^k \rightarrow C_n$, where $C_n = \{\hat{\mathbf{x}}_{ni}, i = 1, 2, \cdots, l_n\}$ is an ordered finite set of l_n codevectors $\hat{\mathbf{x}}_{ni} \in \mathcal{R}^k$. The set C_n is called the codebook for source $\mathbf{X}_n$.

A storage constraint is imposed wherein the total number of stored codevectors is limited to M where typically $M \ll \sum_n l_n$. We therefore consider a *universal codebook* $S = \{\mathbf{s}_i, i = 1, 2, \ldots, M\}$, from which each of the N source codebooks is extracted. To each source codebook we assign a particular subset of universal codevectors. The codevector assignment is specified by the mapping $\mu : \{1, 2, \cdots, N\} \rightarrow \{0, 1\}^M$ which we call the *selector function*. More specifically, the assignment for source $\mathbf{X}_n$ is given by the M-dimensional binary vector $\mu(n)$ whose ith component is defined as

$$\mu_i(n) = \begin{cases} 1, & \text{if } \mathbf{s}_i \in C_n \\ 0, & \text{if } \mathbf{s}_i \notin C_n. \end{cases}$$

Hence, we require $\mu(n)$ to contain exactly l_n "1"s. Correspondingly, the nth source codebook is given by

$$C_n = \{\mathbf{s}_i \in S : \mu_i(n) = 1\}. \tag{1}$$

Thus, the selector function μ is determined by the set of N binary vectors: $\{\mu(n), n = 1, \cdots, N\}$, which will be referred to as *selector vectors.*

A constrained storage quantizer is then completely specified by the following:

1) the universal codebook S;
2) the mapping (selector function) μ.

The notation is illustrated in Fig. 2 which shows a simple example with three sources, $N = 3$, with codebook sizes $l_1 = 2$, $l_2 = 4$, and $l_3 = 3$, and with memory constraint $M = 5$. The arrows linking the sources X_1, X_2 and X_3 to the universal codevectors indicate which universal codevectors are included in the respective codebooks. For example, $C_1 = \{s_1, s_2\}$ is the codebook for the source X_1. The corresponding binary selector vectors are also indicated in the figure.

B. Encoding and Decoding in CSVQ

Suppose a universal codebook with M codevectors, $S = \{\mathbf{s}_i, i = 1, \cdots, M\}$, and the mapping vectors $\mu(\cdot)$, are available at the encoder and decoder. Assume that when a vector $\mathbf{x}_n$ arrives at the encoder, its label n, indicating it is an outcome of source $\mathbf{X}_n$, is given to both encoder and decoder.

Using (1) the encoder extracts $C_n = \{\hat{\mathbf{x}}_{n1}, \cdots, \hat{\mathbf{x}}_{nl_n}\}$ and performs a nearest neighbor search through this codebook by computing the distortion measure $d(\mathbf{x}_n, \hat{\mathbf{x}}_{ni})$ for each vector in C_n. The optimal index j^* is selected where

$$j^* = \arg \min_{1 \leq i \leq l_n} d(\mathbf{x}_n, \hat{\mathbf{x}}_{ni}).$$

The input vector $\mathbf{x}_n$ is thus quantized to the codevector $\hat{\mathbf{x}}_{nj^*}$ and the optimal index j^* is transmitted. Assuming fixed-rate coding, $\log_2 l_n$ bits are needed to specify j^*.

The decoder receives the optimal index j^* and has available the source label n. From its copy of the universal codebook, S, and the mapping $\mu(n)$, the decoder extracts the selected codevector $\hat{\mathbf{x}}_{nj^*}$ and outputs it as the reproduced approximation to the original vector $\mathbf{x}_n$.

Both encoder and decoder need to know to which source the vector being encoded/decoded belongs. In AVQ, the type of source changes infrequently and a number of successive input vectors are treated as emanating from one particular source. This reduces the overhead in transmitting the source label. In certain other applications, the decoder is capable of determining the source index that has been assigned to the encoded vector, due to some prior available information and the source index n need not be explicitly transmitted. For such cases, each successive input vector may be associated with a different source label.

C. Design Objective

Each vector quantizer, Q_n performs the usual nearest neighbor operation, mapping an input vector $\mathbf{x}_n$ to the nearest codevector $\hat{\mathbf{x}}_n = Q_n(\mathbf{x}_n)$ in its codebook C_n. A codebook

C_n is defined to be optimal with respect to the probability distribution of the random vector $\mathbf{X}_n$ if the codebook C_n minimizes the expected distortion $E\{d[\mathbf{X}_n, Q_n(\mathbf{X}_n)]\}$ over all codebooks of size l_n *which can be extracted from* S.

The performance measure of the multiple source coding system is the overall distortion, which is defined here as

$$D = \sum_{n=1}^{N} w_n E\{d[\mathbf{X}_n, Q_n(\mathbf{X}_n)]\} \tag{2}$$

where w_n are nonnegative weights, which measure the relative importance of distortion incurred in quantizing the various sources. These weights are application dependent and are assumed to be given prior to the design process. Often, the weights would simply be the relative frequencies of occurrence of the sources, in which case D has the natural interpretation as the expected distortion (in quantizing a source vector selected at random from the set of possible sources according to their given prior probability distribution).

Problem Statement: Given N sources $\mathbf{X}_n$, rates r_n, and storage constraint M, minimize D over all choices for the universal codebook S and selector function $\mu(n)$.

III. CSVQ Design Method

For a given set of N sources and a storage constraint of M vectors, we wish to determine the optimal set $S = \{\mathbf{s}_i, i = 1, 2, \cdots, M\}$ of universal codevectors, and the optimal selector function $\mu(\cdot)$. We propose an iterative algorithm, where each step consists of fixing a subset of the parameters while optimizing over the others. In particular, it is closely related to known clustering algorithms, such as the generalized Lloyd algorithm (GLA) [17].

Starting with an initial universal codebook $S^{(0)}$ and an initial selector function $\mu^{(0)}$, we iterate the following two steps.

1) Fix the selector function μ and optimize S, the set of universal codevectors.
2) Fix the universal codebook S and optimize μ, the set of binary selector vectors.

Each of these steps is clearly monotonically nonincreasing in distortion. Thus, by iterating we obtain a sequence of CSVQ coding schemes, $(S^{(m)}, \mu^{(m)})$, where m counts the iterations. This sequence of quantizers is nonincreasing in distortion, and the distortion will converge to some limiting value. This provides an (at least locally) optimal CSVQ system.

It should be noted that the above algorithm description is given mainly for its simplicity. It is a skeletal description emphasizing the main concepts. We thus view it as a conceptual starting point whose optimality is easily demonstrated, as well as the relation to the GLA. However, an efficient, practical algorithm will require some modifications. In particular, we shall see that there is a natural and straightforward way to implement step A above. On the other hand, Step 2 is less trivial and would, in fact, require an entire iterative procedure for its implementation alone. In the sequel we first consider in detail and derive the conceptual algorithm, and then we provide an efficient algorithm by embedding step A within the iterations needed to implement step 2.

A. Universal Codebook Optimization

For a fixed set of selector vectors, the universal codevectors can be modified to match the effective sources they represent. The overall distortion for the constrained-storage quantizer is given by

$$D = \sum_j D_j$$

where

$$D_j = \sum_{n:\mu_j(n)=1} w_n E[d(\mathbf{X}_n, \mathbf{s}_j) \mid \mathbf{X}_n \in R_{jn}] P[\mathbf{X}_n \in R_{jn}] \tag{3}$$

where R_{jn} is the nearest neighbor partition cell corresponding to the codevector $\mathbf{s}_j$ when the source n is being quantized:

$$R_{jn} = \{\mathbf{x} \in \mathcal{R}^k : d(\mathbf{x}, \mathbf{s}_j) \leq d(\mathbf{x}, \mathbf{s}_i) \quad \forall i : \mu_i(n) = \mu_j(n) = 1\}. \tag{4}$$

For the given selector vectors, and nearest neighbor partition R_{jn}, the optimal universal codevector s_j is given by

$$\hat{\mathbf{s}}_j = \arg\min_{s_j} D_j$$

which defines the centroid for the partition region R_{jn}. For the mean-square distortion measure, the explicit expression is

$$\hat{\mathbf{s}}_j = \frac{\sum_{n:\mu_j(n)=1} w_n E(\mathbf{X}_n \mid \mathbf{X}_n \in R_{jn})}{\sum_{n:\mu_j(n)=1} w_n}. \tag{5}$$

Since the centroid condition minimizes the distortion for the given selector vectors, this update of the universal codebook yields nonincreasing distortion.

B. Selector Function Optimization

Consider the extraction of C_n given the universal codebook. The problem at hand is that of optimally choosing a subset of l_n codevectors from the set of M universal codevectors. There are obviously too many $(\frac{M!}{(M-l_n)!l_n!})$ possibilities, for brute-force exhaustive approaches to be practical. Thus, a simpler, but possibly suboptimal method is required to determine the selector vector. A similar problem has been addressed heuristically by Nasrabadi *et al.* [15]. They calculated the relative frequency with which each of the universal codevectors would have been used by a particular source, had that source been coded with the entire universal codebook. Then the l_n most frequently used universal codevectors were selected as members of C_n.

In this work, we introduce an iterative procedure for optimizing the selector vectors, that unlike the heuristic of [15] ensures *convergence to a local optimum.* For a particular source $\mathbf{X}_n$, we assume we have a training set representing the source statistics. At each iteration of the algorithm, we first update the encoding partition, $\{R_{jn}\}$, and then we recompute the codevectors. This is almost identical to the GLA except that here we add the necessary constraint that *each of the updated codevectors be a member of the universal codebook.* Hence, the selector function optimization algorithm proceeds as follows.

1) Compute a nearest-neighbor partition.
2) Find the universal codevector which best represents each partition region.

It is easily shown that each of these steps is nonincreasing in distortion: It is well known from the GLA that Step 1 cannot increase the distortion. Step 2 tries to find the best universal codevector to represent the partition computed in Step 1. Switching from the current universal codevector to the best universal codevector can never increase the distortion.

For Step 2, one would generally need to try all universal codevectors and select the one minimizing the distortion. But for the squared error distortion, we can equivalently compute the "unconstrained" centroid, and then quantize it using the universal codebook, as will be shown below. Thus a more efficient algorithm for the squared-error case performs the following steps at each iteration.

1) Compute a nearest-neighbor partition using the current codebook.
2) a) Compute the partition centroids.
   b) Quantize the centroids using the universal codebook.

(Note that quantizing centroids to codevectors from a predefined set appeared in Rao and Pearlman's work on alphabet constrained quantizer design [21].) We now show that Steps 2a and 2b above, are equivalent to selecting the best universal codevector for the given partition cell. We need to find $\mathbf{s}$ in the universal codebook to minimize $D_{jn} = E[(\mathbf{X}_n - \mathbf{s})^2 \mid \mathbf{X}_n \in R_{jn}]$. Using the properties of second moments we rewrite D_{jn} as

$$D_{jn} = E[(\mathbf{X}_n - \mathbf{m}_{jn})^2 \mid \mathbf{X}_n \in R_{jn}] + (\mathbf{m}_{jn} - \mathbf{s})^2 \quad (6)$$

where $\mathbf{m}_{jn} = E[\mathbf{X}_n \mid \mathbf{X}_n \in R_{jn}]$ is the centroid of the cell R_{jn}. Since only the last term in (6) depends on $\mathbf{s}$, the best universal codevector to choose is the one nearest to $\mathbf{m}_{jn}$. Thus, finding the optimal codevector in the universal codebook is equivalent to quantizing m_{jn} with the nearest universal codevector.

To summarize, the basic iteration is guaranteed not to increase the distortion, hence it will decrease the distortion until convergence is achieved. This procedure can be performed on each source independently. At convergence, the algorithm produces an (at least locally) optimal selector function and encoding partition for the given fixed universal codebook.

C. Practical Design Algorithm

In Section III-B, we derived an iterative algorithm for optimizing the selector function given a fixed universal codebook. As we mentioned earlier, performing such optimization as a single step in the "super iteration" of the design algorithm, might be highly impractical. The reason is that much computation would be spent on finding the "perfect" selector function for universal codebooks that are still suboptimal. Here we modify the basic algorithm to produce a practical method which effectively embeds the universal codebook optimization within the selector optimization iterations. More specifically, after each iteration of the selector function optimization of Section III-B we perform a universal codebook update as described in Section III-A. Note that while we do find the optimal universal codebook for the given mapping, we only take one step toward finding the optimal mapping for the given universal codebook. It is easy to see that each step

TABLE I
PSEUDOCODE FOR THE CSVQ DESIGN ALGORITHM

```
    **Given:** a training set for each source.
    *Compute* an initial universal codebook $S^{(0)}$ by applying GLA to the entire
training
     data.
     *Compute* initial codebooks $C_n^{(0)}$ by applying GLA to the source training
sets.
    *Set* $m = 1$.
    do {
        *Update* the mapping :
            For $n = 1$ to $N$ {
                Perform nearest-neighbor partitioning of the $\mathbf{X}_n$ training set
                using codebook ${C_n}^{(m-1)}$ to obtain $\{R_{jn}^{(m)}, j = 1, \ldots, M\}$.
                Compute centroids for every partition cell $E[\mathbf{X}_n | \mathbf{X}_n \in R_{jn}]$
                Quantize centroids to obtain ${C_n}^{(m)}$ and store the selector vector:
          $\mu_j^{(m)}(n) = \begin{cases} 1 & \text{if } \mathbf{s}_j^{(m-1)} \in {C_n}^{(m)} \\ 0 & \text{if } \mathbf{s}_j^{(m-1)} \notin {C_n}^{(m)} \end{cases}, j = 1, \ldots, M$
            }
        *Update* the universal codebook :
                Compute each universal codevector $\mathbf{s}_j^{(m)}$ by (**??**).
         *Increment* $m$.
      } while not converged
```

in the algorithm is monotonically nonincreasing in distortion. Thus, this practical algorithm does not sacrifice optimality, yet significantly reduces the computational complexity of the design. The algorithm is presented in pseudocode in Table I, and a flow-diagram is given in Fig. 3.

Similar to the GLA, the CSVQ method is applicable to sources given by probability distributions, and to sources represented by training sets. In the latter case, one simply assumes that the source distribution is well approximated by the discrete distribution induced by a uniform probability over the samples of the training set. Thus, all our results can be restated within the context of a training algorithm in a straightforward manner.

IV. Experimental Results

A. Finite-State VQ

The proposed technique was applied to FSVQ of images. An FSVQ encoder, as shown in Fig. 4, consists of a finite set of subcodebooks where each subcodebook belongs to a distinct state of the encoder. The current state of the encoder is determined by the previously encoded blocks and the previous state. An input vector X_n is encoded by searching the corresponding subcodebook of the current encoder state for the best representative codevector. Synchronization between the encoder and the decoder is achieved by employing the same next-state function and starting with the same initial state at the transmitter and receiver.

For a given fixed rate, FSVQ's performance can be improved by increasing the number of states. However, since a codebook is assigned to each state, this results in increasing memory requirements. By viewing the different states as "different sources," we can apply the CSVQ design procedure described above directly to the FSVQ problem. Thus,

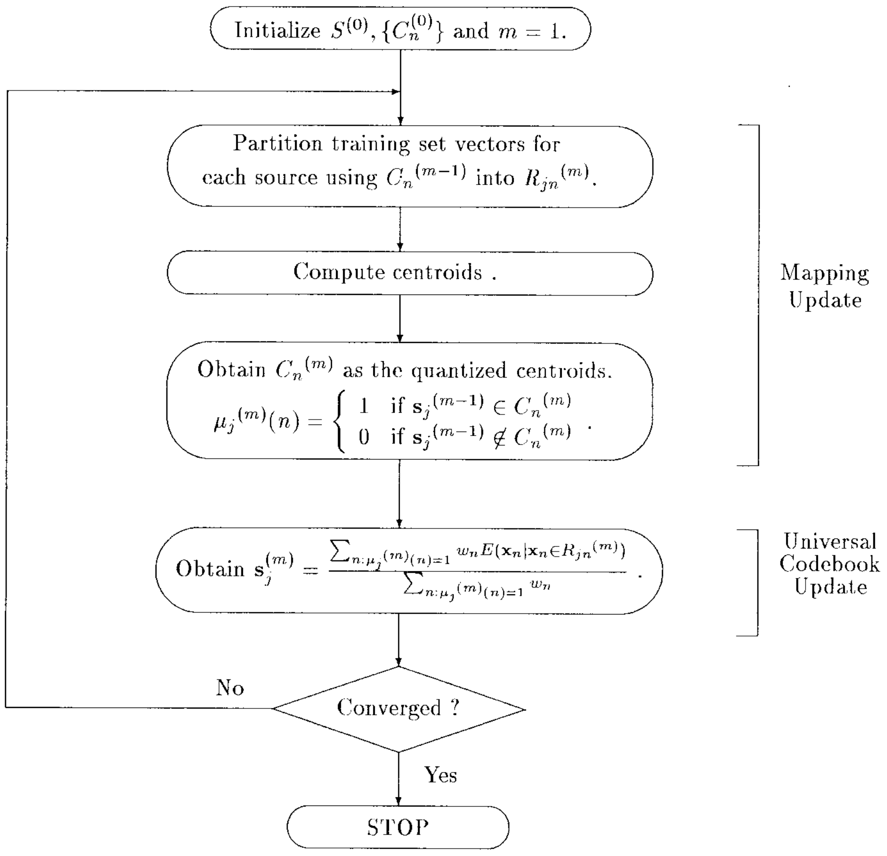

Initialize $S^{(0)}, \{C_n^{(0)}\}$ and $m = 1$.
Partition training set vectors for each source using ${C_n}^{(m-1)}$ into ${R_{jn}}^{(m)}$.
Compute centroids .
Obtain ${C_n}^{(m)}$ as the quantized centroids.
$\mu_j{}^{(m)}(n) = \begin{cases} 1 & \text{if } \mathbf{s}_j{}^{(m-1)} \in {C_n}^{(m)} \\ 0 & \text{if } \mathbf{s}_j{}^{(m-1)} \notin {C_n}^{(m)} \end{cases}$.
Obtain $\mathbf{s}_j^{(m)} = \frac{\sum_{n:\mu_j{}^{(m)}(n)=1} w_n E(\mathbf{x}_n | \mathbf{x}_n \in {R_{jn}}^{(m)})}{\sum_{n:\mu_j{}^{(m)}(n)=1} w_n}$.
Converged ?
No
Yes
STOP
Mapping Update
Universal Codebook Update

Fig. 3. Flow-diagram for the CSVQ design algorithm.

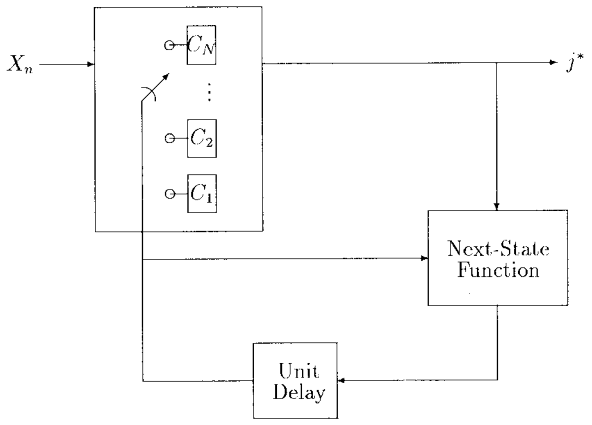


Fig. 4. FSVQ encoder.

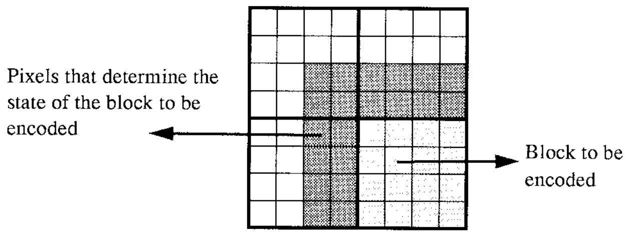


Fig. 5. Current state is determined from the neighboring pixels in the previously encoded blocks.

we apply our storage-constrained approach to optimize the overall FSVQ performance subject to the specified rate *and available memory*. The resulting algorithm is referred to as *constrained storage FSVQ* (CS-FSVQ). We emphasize the important advantage: by the mechanism of codevector sharing we can increase the number of states while respecting the given memory constraint.

For the simulations we selected a data set of images of differing types. Seven images were used for training, and ten other images were used for testing the performance. We segmented these images into 4×4 blocks to create the training and test sets. Given a fixed rate, we applied the omniscient design method [20] to the training set and designed five standard FSVQ systems with number of states $N = 4, 8, 16, 32, 64$, respectively. All codebooks in all systems were of the same size as determined by the fixed rate. The next state function was implemented as a vector quantizer operating on the pixels in the three neighboring blocks as shown in Fig. 5.

We next used the standard FSVQ points for $N = 16, 32, 64$ to facilitate initialization and applied CS-FSVQ to constrain the memory size. More precisely, the initial universal codebook of size M was obtained by applying standard GLA to the union of state codebooks of a standard FSVQ. We then encoded a test set using the previously designed codebooks to obtain the curves depicted in Fig. 6.

The results demonstrate that the CS-FSVQ approach improves over standard FSVQ in two respects. First it provides solutions at arbitrary levels of memory, while standard

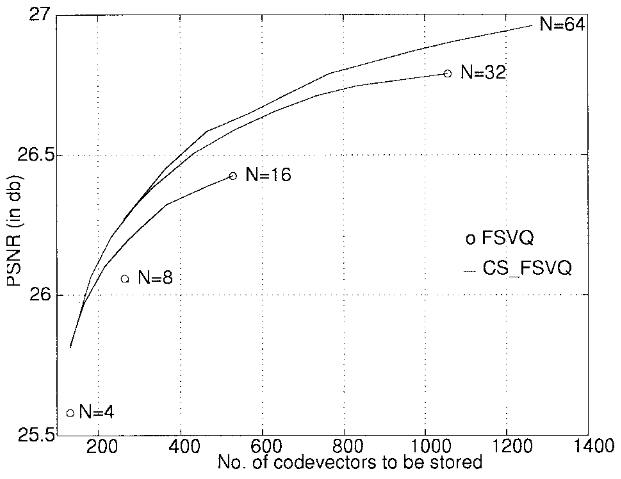


Fig. 6. Comparison of CS-FSVQ and standard FSVQ. The plot shows PSNR versus memory size, each curve corresponds to a different number of states. The rate is 0.3125 b/pixel.

FSVQ can only provide systems at distinct memory sizes corresponding to multiples of state codebook size. Secondly, and more importantly, CS-FSVQ consistently outperforms standard FSVQ at all memory sizes. Please note that the memory values used in Fig. 6 were adjusted to include the memory cost of implementing the next-state function. This gives a fair comparison and penalizes our CS-FSVQ results that allow more states. In fact, other forms of next-state function (instead of the VQ we used) may require less memory.

B. Tree-Structured VQ

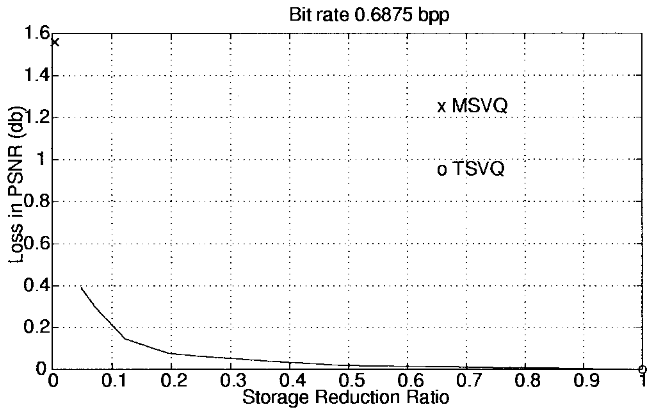
We begin by noting the similarity between tree-structured VQ (TSVQ) and multistage VQ (MSVQ) [18]. In MSVQ, the final reproduction vector is obtained by summing over the reproduction vectors at the different stages. At each stage the conventional MSVQ uses a single codebook. We refer to such a system as an MSVQ of *fanout* 1. Allowing more residual codebooks at each stage, will improve performance at the cost of increase in fanout [5], [19]. In this case, the codebook used in a certain stage would be dependent on the reproduction vector obtained in the previous stage. If there are as many codebooks in the current stage as there are reproduction vectors in the previous stage, then the structure is an MSVQ of *full fanout*. Such a structure is in fact equivalent to a TSVQ except for the technicality of operating on residuals at each layer instead of directly operating on the source vector at the leaf layer (standard TSVQ). We generalize our terminology and refer to MSVQ with fanout n as n-MSVQ. The standard MSVQ and the N-ary TSVQ are the special cases 1-MSVQ and N-MSVQ, respectively. While all n-MSVQ have the same encoding search complexity as TSVQ, their memory requirements differ. In fact, they provide many intermediate levels of memory between the least—standard MSVQ and the largest memory requirement of TSVQ. Thus, the n-MSVQ family provides us with a tool for trading memory for performance. However, we emphasize that within this family the memory requirements determine the structure to be used. CSVQ provides the means for restricting memory without compromising the structure.

Before describing the CSVQ approach to TSVQ deign, it is important to note that the TSVQ method of Lyons *et al.* [14] is closely related. There, the approach was based on the idea that the codebook of testvectors can be quantized itself. Since, in principle, quantization is a form of codevector sharing, the methods have some important similarities. Some important differences: the CSVQ approach offers joint optimization of the overall system; Lyons *et al.* recommend scalar quantization of the testvectors while CSVQ effectively “quantizes” them as full dimensional vectors; and CSVQ is generally derived for any mixture of sources regardless of the VQ structure.

In order to adapt our CSVQ approach to MSVQ/TSVQ, we consider the collection of vectors using the same codebook as an “independent source.” Hence, the m-ary balanced tree has one source at the first layer, m sources at the second layer and $m^{(p-1)}$ sources at the pth layer. We now require that all the codebooks be extracted from some universal codebook of given size M. The memory requirement of this constrained storage approach to TSVQ (CS-TSVQ) *is not dictated by the structure* but is a variable whose value can be chosen by the designer. Since the storage requirement is independent of the fanout used in this CS-TSVQ approach, a full fanout (i.e., a structure equivalent to TSVQ) might as well be used to best exploit the available design flexibility offered by a larger fanout. We thus design a TSVQ subject to storage constraints. By viewing our TSVQ as a full fanout MSVQ, where residuals are used at each layer rather than the original source vector itself, one can achieve equivalent overall distortion with a smaller universal codebook. This is due to the fact that “residual test vectors” are smaller in magnitude and are statistically similar. This significantly improves the performance of memory sharing.

Having specified the sources that need to be quantized while satisfying the overall memory constraint, we now have a CSVQ problem that we can solve as we have done in the preceding sections. It should be noted that although we have stated above that we could treat the sources using the different codebooks independently, the source index of a given vector at layer p is in fact dependent on the codebooks in the previous layers. This allows the training set for a given source to change from iteration to iteration. Hence, the algorithm does not guarantee a nonincreasing sequence of distortion values, and the universal codebook produced at termination is not necessarily optimal. What we practically observe in simulations is a fluctuation about the minimum point. This problem is practically handled by storing the best universal codebook and selector function during the fluctuations.

The proposed algorithm was applied to the tree-structured vector quantization of images. The same training set as that used for our FSVQ simulations was used. A binary tree with 11 layers was grown, where the codebook at each layer operates on the residue obtained in the previous level. The constrained-storage approach was used to design universal codebooks of different sizes. Fig. 7 shows the loss of performance in dB compared to a conventional (full memory) TSVQ versus the



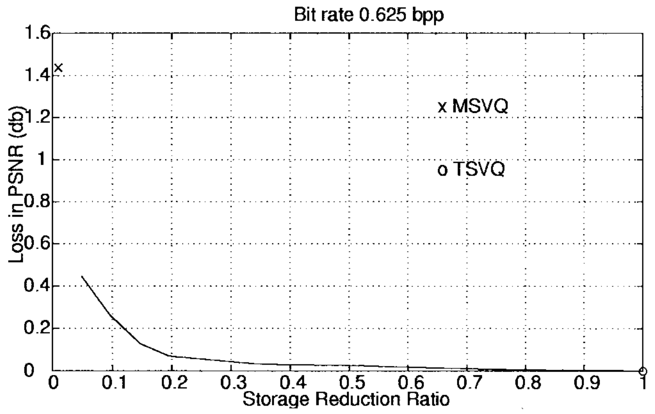


Fig. 7. Comparison of CS-TSVQ, TSVQ and MSVQ. The plot displays the loss in PSNR when compared to the performance of TSVQ. The end-point of the curve with a 0 dB loss is the TSVQ solution. The disconnected point with the maximum loss corresponds to the solution obtained by MSVQ. All intermediate points are the CS-TSVQ solutions.

storage reduction ratio (SRR) defined as

$$\frac{\text{Memory required by the constrained-storage approach}}{\text{Memory required by a standard tree}} = \frac{M}{\sum_{n=1}^{N} l_n}. \quad (7)$$

Thus, SRR $= 1$ corresponds to the conventional TSVQ and the loss associated with it is 0 db. Also, for an 11 layer binary tree (7) becomes SRR $= M/4094$ since $l_n = 2$ for binary trees.

A test set was encoded using two trees of 10 and 11 layers, corresponding to rates of 0.625 and 0.6875 b/pixel, respectively. The results obtained are displayed in the graphs of Fig. 7. The graphs show that storage reduction by a factor of up to five is obtained at the cost of less than 0.1 dB loss in PSNR, while storage reduction by factors of up to 20 requires less than 0.5 dB loss in PSNR. The point corresponding to standard 1-MSVQ is also shown on the graph. Note that we neglected the memory requirement for the selector function in the SRR calculation. Detailed calculation shows that it is less than 2% of the TSVQ memory cost in most cases considered. It should be noted all the points in the above graph have the same computational complexity as TSVQ. Hence CS-TSVQ offers the low computational complexity and almost the same performance as TSVQ, but with much reduced memory requirements. Also note that the CSVQ approach allows one to obtain intermediate solutions at any memory requirement between standard MSVQ and standard TSVQ.

V. Conclusion and Future Directions

We introduced a new codevector sharing approach to constrained-storage VQ, for quantizing multiple sources. The method uses a single universal codebook whose size is limited by the specific application. A number of sources may use the same universal codevector, but they are not constrained to use exactly the same set of codevectors for their codebooks. As confirmed by the experimental results reported here, this technique trades off very modest reductions in performance for very large savings in storage complexity, and provides an (at least locally) optimal design for a given memory size.

Besides the applications with which we have already experimented, there are a number of other interesting applications. Perhaps a less obvious possible application of the CSVQ approach is in fine-coarse VQ (FCVQ) [16] whose objective is to reduce the computational complexity of vector quantizers. When using a tree for this purpose, the design method suggested in [16] could be improved by using our approach, by associating the fine quantizer with the source codebooks of size one, and the coarse codebook with our definition of a universal codebook.

Thus, the proposed CSVQ technique is a fundamental one in the sense that it is directly applicable to many different problems, including a variety of structured VQ schemes, such as FSVQ, CVQ, and TSVQ, various problems of universal quantization, and other problems. The practical importance of the method is in the fact that it can be used to impose memory constraints on virtually any VQ-based system.

References

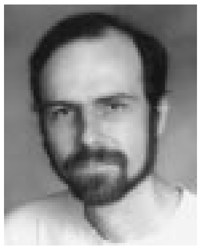
[1] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Boston, MA: Kluwer, 1991.
[2] A. Gersho, "Adaptive vector quantization," *Ann. Telecommun.*, vol. 41, pp. 470–480, Sept. 1986.
[3] M. Goldberg, P. Boucher, and S. Shlien, "Image compression using adaptive vector quantization," *IEEE Trans. Commun.*, vol. COMM-34, pp. 180–187, Feb. 1986.
[4] A. Buzo, A. H. Gray, R. M. Gray, and J. D. Markel, "Speech coding based upon vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-28, pp. 562–574, Oct. 1980.
[5] W. Y. Chan and A. Gersho, "Generalized product code vector quantization: A family of efficient techniques for signal compression," *Digital Signal Process.*, vol. 4, pp. 95–126, Apr. 1994.
[6] B. Ramamurthi and A. Gersho, "Classified vector quantization of images," *IEEE Trans. Commun.*, vol. COM-34, pp. 1105–1115, Nov., 1986.
[7] R. Aravind and A. Gersho, "Image compression based on vector quantization with finite memory," *Opt. Eng.*, vol. 26, pp. 570–580, July 1987.
[8] W. Y. Chan and A. Gersho, "Constrained-storage quantization of multiple vector sources by codebook sharing," *IEEE Trans. Commun.*, vol. 39, pp. 11–13, Jan. 1991.
[9] S. Ramakrishnan, K. Rose, and A. Gersho, "Constrained-storage vector quantization with a universal codebook," in *Proc. Data Compression Conf.*, Apr. 1995, pp. 42–51.
[10] W. Y. Chan and A. Gersho, "Constrained-storage vector quantization in high fidelity audio transform coding," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Toronto, Ont., Canada, May 1991, pp. 2497–3600.

[11] K. Zeger, A. Bist, and T. Linder, "Universal source coding with codebook transmission," *IEEE Trans. Commun.*, vol. 42, pp. 336–346, Feb./Mar./Apr. 1994.
[12] K. Zeger and A. Bist, "Universal adaptive vector quantization using codebook quantization with application to image compression," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, San Francisco, CA, Mar. 1992, pp. 381–384.
[13] T. Kim, "Side match and overlap match vector quantizers for images," *IEEE Trans. Image Processing*, vol. 1, pp. 170–185, Apr. 1992.
[14] D. F. Lyons, D. L. Neuhoff, and D. Hui, "Reduced storage tree-structured vector quantization," in *Proc. IEEE Conf. Acoustics, Speech, Signal Processing* , Minneapolis, MN, Apr. 1993, vol. 5, pp. 602–605.
[15] N. M. Nasrabadi, C. Y. Choo, and Y. Feng, "Dynamic finite-state vector quantization of digital images," *IEEE Trans. Commun.*, vol. 42, pp. 2145–2154, May 1994.
[16] N. Moayeri, D. L. Neuhoff, and W. E. Stark, "Fine-coarse vector quantization," *IEEE Trans. Signal Processing*, vol. 39, pp. 1503–1515, July 1991.
[17] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COMM-28, pp. 84–95, Jan. 1980.
[18] B.-H. Juang, "Multiple stage vector quantization for speech coding," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Apr. 1982, pp. 597–600.
[19] W. Y. Chan and A. Gersho, "Enhanced multistage vector quantization with constrained storage," in *Proc. Asilomar Conf. Signals, Systems, Computers*, Nov. 1990, pp. 659–663.
[20] J. Foster, R. M. Gray, and M. O. Dunham, "Finite-state vector quantization for waveform coding," *IEEE Trans. Inform. Theory*, vol. IT-31, pp. 348–359, May 1985.
[21] R. P. Rao and W. A. Pearlman, "Alphabet-constrained vector quantization," *IEEE Trans. Inform. Theory*, vol. 39, pp. 1167–1179, July 1993.

Sangeeta Ramakrishnan received the B.S. degree in electronics and communications from the College of Engineering, Anna University, Madras, India, in June 1993, and the M.S. degree in electrical and computer engineering from the University of California, Santa Barbara, in March 1995.

She is currently with Chromatic Research, Sunnyvale, CA. Her research interests include image and video compression, vector quantization and digital signal processing.

Ms. Ramakrishnan was a recipient of the UC Regents Fellowship.



Kenneth Rose (S'85–M'91) received the B.Sc. (summa cum laude) and M.Sc. (magna cum laude) degrees in electrical engineering from Tel-Aviv University, Israel, in 1983 and 1987, respectively, and the Ph.D. degree in electrical engineering from the California Institute of Technology (Caltech), Pasadena, in 1990.

From July 1983 to July 1988, he was employed by Tadiran Ltd., Israel, where he carried out research in the areas of image coding, image transmission through noisy channels, and general image processing. From September 1988 to December 1990, he was a graduate student at Caltech. In January 1991, he joined the Department of Electrical and Computer Engineering, University of California, Santa Barbara, where he is currently an Associate Professor. His research interests are in information theory, source and channel coding, pattern recognition, image coding, and processing, and nonconvex optimization in general.

Dr. Rose was co-recipient of the William R. Bennett Prize Paper Award of the IEEE Communications Society (1990).

Allen Gersho (S'58–M'64–SM'78–F'82) received the B.S. degree from the Massachusetts Institute of Technology in 1960, and the Ph.D. from Cornell University, Ithaca, NY, in 1963.

He was at Bell Laboratories, Murray Hill, NJ, from 1963 to 1980. He is currently Professor of Electrical and Computer Engineering at the University of California, Santa Barbara (UCSB). His current research activities are in signal compression methodologies and algorithm development for speech, audio, image and video coding.

He is co-author, with R. M. Gray, of the book, *Vector Quantization and Signal Compression* (Boston, MA: Kluwer, 1992), and co-editor of two books on speech coding. He served as a member of the Board of Governors of the IEEE Communications Society from 1982 to 1985, and is a member of various IEEE technical, award, and conference management committees. He has served as Editor of *IEEE Communications Magazine* and Associate Editor of the IEEE TRANSACTIONS ON COMMUNICATIONS. He received NASA "Tech Brief" Awards for technical innovation in 1987, 1988, and 1992. In 1980, he was co-recipient of the Guillemin–Cauer Prize Paper Award from the IEEE Circuits and Systems Society. He received the Donald McClennan Meritorious Service Award from the IEEE Communications Society in 1983, and in 1984 he was awarded an IEEE Centennial Medal. In 1992, he was co-recipient of the 1992 Video Technology Transactions Best Paper Award from the IEEE Circuits and Systems Society. He holds patents on speech coding, quantization, adaptive equalization, digital filtering, and modulation and coding for voiceband data modems.