LINEAR-TRANSLATE CONSTRAINED STORAGE VQ FOR VSPIHT WAVELET IMAGE COMPRESSION*

Debargha Mukherjee

Hewlet Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94304.
Email: debargha@hpl.hp.com

Sanjit K. Mitra

Department of Electrical and Computer Engineering,
University of California, Santa Barbara, CA 93106.
Email: mitra@ece.ucsb.edu

ABSTRACT

A new Constrained Storage VQ (CSVQ) structure based on linear transforms and translates of a common root codebook is proposed. The new VQ structure, named LT-CSVQ (Linear Translate CSVQ), acts as a building block for multistage VQ implementations (LT-CS-MSVQ), and significantly reduces storage requirements from that required in tree-multistage VQ implementations. LT-CS-MSVQ is most appropriate for medium rate multistage VQ implementations, and is applied to the recently proposed vector enhancement of Said and Pearlman's Set Partitioning in Hierarchical Trees (SPIHT) image coder, named VSPIHT.

1. INTRODUCTION

Vector extensions to popular scalar wavelet image coding schemes like EZW [1] and SPIHT [2], have recently been proposed, and named Vector EZW and Vector SPIHT (VSPIHT) respectively. While lattice VQ based schemes [3]-[5] have fast algorithms, trained VQ VSPIHT schemes [6]-[8] are superior in rate-distortion performance. The main idea in VSPIHT is to classify groups of coefficients over concentric hyperspheres of decreasing radii, and progressively quantize them using classified tree-multistage VQ (TS-MSVQ) structures. A key factor affecting the efficiency of vector based successive approximation encoders, such as VSPIHT, is the effectiveness of successive refinement VQ systems used for quantization. *Multistage VQ* (MSVQ), alternatively known as *Residual VQ* (RVQ) [9], [10], is a natural candidate for such progressive refinement VQ applications. Unfortunately, the performance of MSVQs degrade substantially with the increase in the number of stages due to *tree-entanglement*, that effectively reduces the codebook size from the allocated rate. Although the suboptimalities introduced by RVQ with multiple stages may be alleviated by the use of joint codebook design procedures, in conjunction with more complex multiple path search procedures [11], [12], such schemes are less effective for embedded coding applications, where the number of stages to which a vector will be decoded is not known at the time of encoding.

Considering the above factors, the only reasonable way to enhance the efficiency of trained successive refinement VQs is to increase the storage. MSVQ and *Tree-Structured VQ* (TSVQ) [13], [10], constitute two ends of the storage spectrum. While codebook fan-out for MSVQ is unity at every stage, that for TSVQ increases exponentially from stage to stage. For most applications including VSPIHT, the codebook storage requirement as well as the training set size requirement for TSVQ is untenable, while the performance of MSVQ is unacceptable. Therefore, successive refinement VQ structures that achieve intermediate storage-efficiency trade-offs between TSVQ and MSVQ, is of vital importance. The VSPIHT implementations presented in [6]-[8] use tree-multistage VQs to achieve a certain intermediate trade-off. Unfortunately, they require a large storage, are very rigid, and do not allow fine adjustments in storage and rate.

Chan and Gersho [14] developed a more sophisticated scheme called Constrained-Storage VQ (CSVQ), that allows multiple sources to be vector quantized with fewer codebooks by *codebook sharing*. CSVQ can be readily combined with MSVQ to obtain Constrained Storage Multistage VQs (CS-MSVQ) with arbitrary fan-outs. These structures attempt reducing the suboptimalities of MSVQ by incremental incorporation of memory into the stages. In order to improve the storage-efficiency trade-off of CSVQ [14], Ramakrishnan *et. at.* [15] recently proposed a generalized framework for CSVQ, using shared codebooks chosen as subsets of a universal codebook. In this paper, an alternative Linear Transformation (LT) constraint for CSVQ is proposed, which for CS-MSVQ chains (LT-CS-MSVQ) with a large number of stages, achieve an even better storage-efficiency trade-off. The eventual goal is to replace the TS-MSVQ structures in VSPIHT with these CS-MSVQ structures that are more flexible and require less storage. Results are presented comparing LT-CS-MSVQ with Chan and Gersho's Unstructured CS-MSVQ (referred to as U-CS-MSVQ in this paper), while possible combination with the universal codebook approach [15] is left as a future extension.

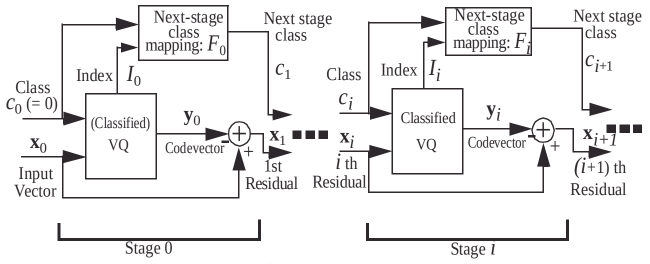

Figure 1. The CS-MSVQ structure

2. THE CS-MSVQ STRUCTURE

In a CS-MSVQ chain, all possible fanned out codebooks after each stage, are merged in a controlled manner so that a desired effective fan-out is obtained. Since fan-out determines storage, which in turn determines efficiency, an arbitrary trade-off between storage and efficiency of successive refinement is achieved. The CS-MSVQ structure is shown in Figure 1. Each stage uses a classified VQ (CVQ) system, where the class of a vector residual is determined by the indices transmitted at the previous stages. Further, the mapping function that yields the class of the residual at the next stage is obtained from the training data itself, and is considered an integral part of the stage codebooks. In the diagram, $\mathbf{x}_0$ represents an input vector. If the VQ itself is not classified, then the class index c_0 of $\mathbf{x}_0$ is taken as 0. For any stage i, *the input* consists of the ith residual $\mathbf{x}_i$ and the corresponding class index c_i. The classified VQ associated with the stage encodes $\mathbf{x}_i$ to generate codevector $\mathbf{y}_i$ and index I_i while the next class mapping function F_i generates class index c_{i+1} from c_i and I_i. The residual $\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{y}_i$ is passed on to the next stage along with class index c_{i+1}.

The design effort in CS-MSVQ consists of designing the stage CVQs as well as the mapping functions. While a truly optimal design over all stages must account for all dependencies between the CVQs and the mapping functions, in practice such schemes are hard to design under reasonable complexity constraints. The design procedures proposed in [14] are of a stage-wise greedy nature where the objective is to do the best encoding at each stage. This methodology also fits well with the objectives of embedded coding. Depending on the nature of the CVQs, several types of CS-MSVQs can be defined. The standard CS-MSVQ structure [14] is referred to as Unstructured CS-MSVQ (U-CS-MSVQ), because unstructured CVQs are used. In this paper, additional structural constraints are imposed on the CVQ to reduce storage without sacrificing efficiency. The next section describes the proposed CVQ and resultant multistage chains.

* This work was supported by the University of California MICRO grant with matching support from Lucent Technologies, Raytheon Missile Systems, Tektronix Corporation and Xerox Corporation.

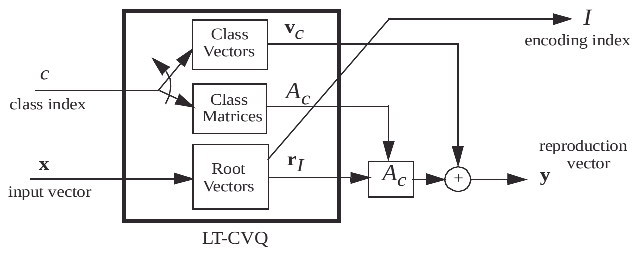

Figure 2. A Linear Translate-Classified VQ Structure

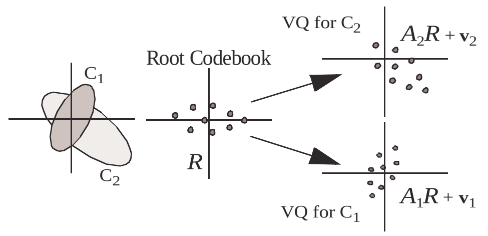

Figure 3. A pictorial description of a 2-class LT-CVQ

3. LINEAR-TRANSLATE CLASSIFIED VQ

We first consider an isolated Linear-Translate Classified VQ (LT-CVQ) system that forms the basic module in a LT-CS-MSVQ chain. Let the number of classes in a LT-CVQ be N, and the number of codevectors for each class be fixed at M. Then, M root vectors, $\mathbf{r}_k,\ k = 0, 1, \ldots, M-1$, along with N class matrices A_j, and N class vectors $\mathbf{v}_j,\ j = 0, 1, \ldots, N-1$, need to be designed, so that the effective size-M codebook used for class j is given by the set $\{A_j\mathbf{r}_k + \mathbf{v}_j | k = 0, 1, \ldots, M-1\}$.

Figure 2 shows the LT-CVQ structure diagrammatically. Given an input vector $\mathbf{x}$, and associated class index c, the encoding rule is to find the root vector $\mathbf{r}_k$ that minimizes the distance $\|\mathbf{x} - A_c\mathbf{r}_k - \mathbf{v}_c\|^2$ over all k. In other words, the codebooks for the individual classes are obtained by linear transformations and translations of a common set of root vectors. Figure 3 shows an example 2-class LT-CVQ. The classes are distributed as shown on the left. However, a single root codebook R is used to generate the codebooks for both the classes, after a class-specific linear transformation and translation.

The rationale for the use of the LT-constraint is as follows. If the individual class distributions are such that one can be approximately transformed into the other with linear transformations and translations, then significant savings in storage may be achieved by storing a single root codebook for all classes, with the individual codebooks for each being generated by class specific linear transformations and translations. The CSVQ approach [14] needs duplicate storage for overlapped regions in class distributions. The universal codebook based approach to CSVQ [15] on the other hand, while being very flexible in terms of rate, can only exploit similarities in the overlapped regions of the class distributions. So if the overlapped regions are small, due to substantial difference either in scale, orientation, or centroids of the individual distributions, then universal codebooks lose their storage efficiency. LT-CVQ, on the contrary, can effectively exploit similarities both in the overlapped regions and non-overlapped regions, by rotation/scaling etc. of a common set of root codevectors. It is to be mentioned, that the LT-CVQ approach can be readily combined with the universal codebook CSVQ approach [15], to reap the same benefits of flexibility in rate. This may be done by choosing the root codebook for each class as subsets of a single *universal root codebook*. In this paper however, we deal only with the case where the full root codebook is used for all classes.

3.1 LT-CVQ Design Procedure

Let us assume that there is a separate training set $\chi_j,\ j = 0, 1, \ldots, N-1$, for each of the N classes of vectors of dimension d. Also assume the number of root vectors to be designed to be M. The LT-CVQ design objective then is to find M optimal root vectors $\mathbf{r}_k,\ k = 0, 1, \ldots, M-1$, along with N optimal class matrices A_j, and N optimal class vectors $\mathbf{v}_j,\ j = 0, 1, \ldots, N-1$, so as to minimize the distortion for all the training sets. We present an iterative sequential update technique for the three codebooks, where, after each encoding pass, only one of the three quantities $\mathbf{r}_k, A_j, \mathbf{v}_j$ are updated.

Given the root vectors, class matrices and class vectors at any phase of the iterative process, all the N training sets are first encoded by the nearest neighbor encoding rule. Based on the encoding results, each training set χ_j is partitioned into M smaller sets χ_{jk}, where χ_{jk} is the subset of χ_j that chooses $\mathbf{r}_k$ as the root vector. The overall distortion for the training set is then given by:

$$D = \sum_{j=0}^{N-1}\sum_{k=0}^{M-1}\left(\sum_{\mathbf{x}\in\chi_{jk}} \|\mathbf{x} - A_j\mathbf{r}_k - \mathbf{v}_j\|^2\right) \qquad (1)$$

The centroid condition can be separately written for each of the three quantities $\mathbf{r}_k, A_j, \mathbf{v}_j$, (while holding the other two fixed) by taking partial derivatives of Eq. (1) w.r.t. one of $\mathbf{r}_k, A_j, \mathbf{v}_j$, and equating to zero. This leads to the following sequential update rules:

$$\mathbf{r}_k = \left(\sum_{j=0}^{N-1}|\chi_{jk}|A_j^TA_j\right)^{-1}\left(\sum_{j=0}^{N-1}A_j^T\cdot\sum_{\mathbf{x}\in\chi_{jk}}(\mathbf{x}-\mathbf{v}_j)\right) \quad k = 0, \ldots M-1 \qquad (2)$$

$$A_j = \left(\sum_{k=0}^{M-1}\sum_{\mathbf{x}\in\chi_{jk}}(\mathbf{x}-\mathbf{v}_j)\mathbf{r}_k^T\right)\left(\sum_{k=0}^{M-1}|\chi_{jk}|\mathbf{r}_k\mathbf{r}_k^T\right)^{-1} \quad j = 0, \ldots, N-1 \qquad (3)$$

$$\mathbf{v}_j = \frac{1}{|\chi_j|}\sum_{k=0}^{M-1}\left(\sum_{\mathbf{x}\in\chi_{jk}}(\mathbf{x}-A_j\mathbf{r}_k)\right) \quad j = 0, \ldots N-1 \qquad (4)$$

where the $|\chi|$ notation refers to the number of elements in a set χ. We assume that the number of root vectors M is sufficiently large, and the training sets do not form pathological distributions, so that the matrix inversions in Eq. (2) and Eq. (3) are good.

The sequential update procedure starts from an initial set of codebooks $\mathbf{r}_k, A_j, \mathbf{v}_j$, and refines them one at a time in that order. A single iteration therefore comprises three sub-iterations in succession, the first for $\mathbf{r}_k$, the second for A_j, and the third for $\mathbf{v}_j$. Note that each sub-iteration reduces the distortion in Eq. (1). The distortion is checked after every full iteration for a standard stopping criterion.

The initial codebooks $\mathbf{r}_k, A_j, \mathbf{v}_j$, are chosen as follows. For each class j, A_j and $\mathbf{v}_j$ are so chosen that the second order distributions for the individual training sets each warp to the zero-centered average second order distribution of all training sets taken together after the inverse transformation (*i.e.* translation by $-\mathbf{v}_j$ followed by linear transformation with A_j^{-1}). In order to do so, the means $\mathbf{m}_j$ and the autocovariance matrices K_j for all classes χ_j are computed, along with the autocovariance matrix K of all the combined training sets. Then the following assignment for the initial A_j and $\mathbf{v}_j$ are made:

$$\mathbf{v}_j = \mathbf{m}_j, \quad A_j = X_j\Lambda_j^{1/2}\Lambda^{-1/2}X^T \qquad (5)$$

where $K_j = X_j\Lambda_jX_j^T$ and $K = X\Lambda X^T$ are the diagonalizations of positive definite matrices K_j and K respectively with unitary X_j and X. Each training set χ_j is then translated by $-\mathbf{v}_j$ and linear transformed with non-singular A_j^{-1}, before combining them all into a common pool. Note that the second order distribution of the common pool would be K since the individual distributions making up the common pool each have the same second order distribution K. This common pool is then used to design M root vectors $\mathbf{r}_k$ by the GLA. These root vectors, along with the class matrices and vectors given by Eq. (5) form the initial guesses before the iterative sequential update design commences.

3.2 LT-CVQ Encoding Complexity

The storage advantage of LT-CVQ over Unstructured Classified VQ is not achieved without a price. For every input vector, the actual codevectors in the class to which it belongs, can only be computed with Md^2 multiplications and Md^2 additions, where the meaning of

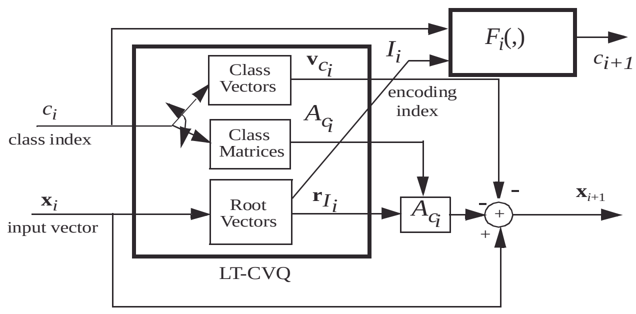

Figure 4. A stage of a LT-CS-MSVQ chain

the quantities M and d are as in the previous section. For a practical LT-CVQ reduced memory implementation, the encoder would typically store a few of the effective class codebooks in a cache at any given time, while swapping and recomputing the codevectors for a new class as and when required. Depending on the frequency of cache updates required, determined by the cache size, the computational complexity involved may be significantly more than that needed for Unstructured CVQ.

One approach to solving the complexity problem would be to impose orthogonality constraints on the linear transform matrices by suitable modifications of the design process. With orthogonal matrices, a lower complexity *weighted* nearest-neighbor encoding rule can be used on the inverse transformed input with untransformed root codevectors, in lieu of the standard nearest neighbor encoding rule with transformed root codevectors. We call this Orthogonal LT-CVQ (OLT-CVQ), but do not investigate them further in this work.

3.3 LT-CS-MSVQ

LT-CS-MSVQ is simply a CS-MSVQ chain where the classified VQ in each stage is a LT-CVQ. Figure 4 shows a module of a LT-CS-MSVQ chain. For high resolution successive quantization, as the number of stages in a CS-MSVQ structure increases, the initial distribution of the input vectors tend to lose their character. While the residuals remain confined within polytopic Voronoi regions around codevectors of the previous encoding stage, their distributions become more and more uniform within these regions. Further, the distributions tend to differ substantially in scale so that the overlap regions for the classes reduce substantially, leading to loss of storage efficiency even for a MS-CSVQ designed using the universal codebook approach [15]. Under these circumstances, choosing codevectors as linear transformations and translations of a common set of root vectors enable more effective sharing of codevectors, both in overlapped and non-overlapped regions, and the storage requirement is substantially reduced for the same efficiency.

A stagewise joint design procedure similar to [14] may be used to jointly design the mapping function for the current stage and the LT-CVQ for the next stage, using an iterative alternate update rule. For a given mapping function, the next stage CVQ is first updated using the LT-CVQ design procedure in Section 3.1 operating on merged residual clusters. Next, for a given next stage LT-CVQ, the current mapping function is updated to map each small product residual cluster to the next stage class yielding the lowest overall distortion. The initial mapping function is designed based on similarity of distributions of product residual clusters.

Alternatively, a decoupled design procedure based on separate clustering and design operations may be used if the rate and fan-out required is too high for the joint design to remain practical. Given the training the residual classes at a given stage, the LT-CVQ design procedure is followed by a clustering operation on the product residual clusters based on similarily of their distributions, to generate the next-stage mapping functions as well as the new training sets for the next stage.

3.4 Storage Complexity

We consider the storage requirement in each stage of a LT-CS-MSVQ chain, where all stages are assumed to be similar in parameters (except possibly the first). Let the number of input classes in a stage be N, the number of root vectors be M, and the vector dimension be d. Then, if f be the number of bits needed to store a floating point number, the storage requirement in bits for the N class matrices and vectors, is given by $fN(d^2+d)$, while that for the M root vectors is given by fMd. The storage required in bits for storing MN N-ary sumbols in the mapping function is given by $MN\lceil \log_2 N\rceil$. The overall storage per stage in bits, $S_{LT\text{-}CS\text{-}MSVQ}(N, M, d)$, is then given by adding the three:

$$S_{LT\text{-}CS\text{-}MSVQ}(N, M, d) = fN(d^2+d) + fMd + MN\lceil \log_2 N\rceil \quad (6)$$

Note that storage required by unstructured U-CS-MSVQ is given by:

$$S_{U\text{-}CS\text{-}MSVQ}(N, M, d) = fNMd + MN\lceil \log_2 N\rceil \quad (7)$$

Comparing Eq. (6) and Eq. (7) for the same set of parameters (N, M, d), the following deduction can be made. Storage for LT-CS-MSVQ is less than that for U-CS-MSVQ as long as:

$$N > M/(M-d-1) \quad (8)$$

It is to be noted that due to the additional structural constraints on LT-CS-MSVQ, the same value of N as in U-CS-MSVQ for a given value of M and d, yields a lower SNR for LT-CS-MSVQ. To compensate for this difference, N has to be increased appropriately. It is still hoped however, that the overall storage-efficiency trade-off will be better in LT-CS-MSVQ than in U-CS-MSVQ.

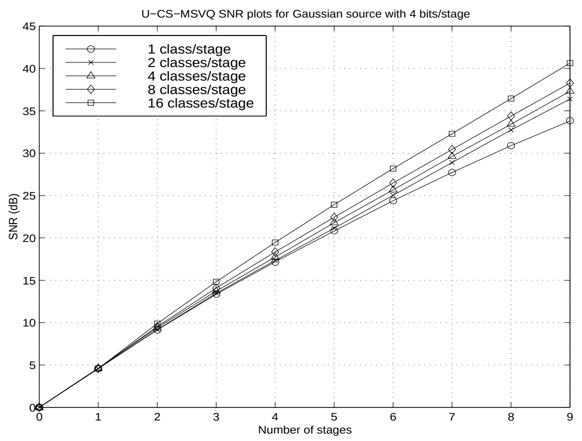

Figure 5. SNR progression with stages for Gaussian data, obtained by a U-CS-MSVQ chain with 4-dim vectors, at rate 4 bits/stage, for varying number of input classes in each stage.

4. IMPLEMENTATION AND RESULTS

4.1 Codebook Design for Gaussian Data

In order to test the U-CS-MSVQ and LT-CS-MSVQ structures, we designed them for a training set of Gaussian data. Both the structures quantize 4-dimensional vectors in 9 stages, with 4 bits allocated to each stage, *i.e.* $d = 4$, and $M = 16$. Figure 5 shows the SNR progression with stages for U-CS-MSVQ, obtained for various values of N (number of classes in each stage, except the first, which has a single input class). In particular, values of N = 1, 2, 4, 8, and 16 are used. Note the N = 1 corresponds to the standard multistage VQ. Figure 6 shows the corresponding results for a LT-CS-MSVQ chain.

Comparing the figures, we see that while SNR for LT-CS-MSVQ is lower than U-CS-MSVQ for the same N, its storage requirement is much lower too. As an example, we compare the storage/stage required by U-CS-MSVQ with N = 8 in the above situation, with that required for LT-CS-MSVQ with N = 16. If we assume that 4 bytes (32 bits) are used to store a floating point number, then using Eq. (7) with f = 32, N = 8, M = 16, d = 4, yields $S_{U\text{-}CS\text{-}MSVQ} = 16768$ bits/stage. On the other hand, using Eq. (6) with f = 32, N = 16, M = 16, d = 4, yields $S_{LT\text{-}CS\text{-}MSVQ} = 13312$ bits/stage. The latter however yields lower distortion than the former. Figure 7 compares explicitly for LT-CS-MSVQ and U-CS-MSVQ, the SNR obtained after 9 stages plotted against the required storage/stage, for the same Gaussian data as before. The graph clearly shows the superiority of LT-CS-MSVQ over U-CS-MSVQ for a modrately large number of stages.

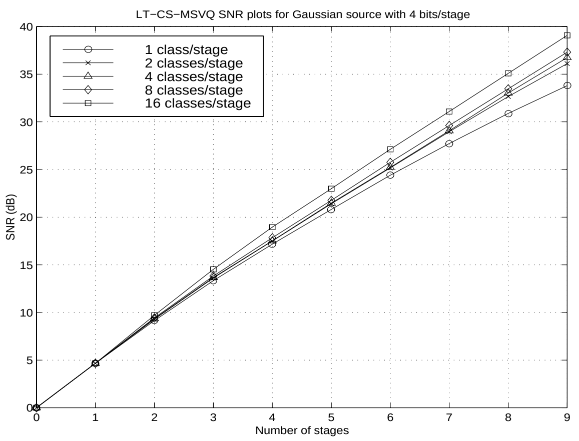

Figure 6. SNR progression with stages for Gaussian data, obtained by a LT-CS-MSVQ chain with 4-dim vectors, at rate 4 bits/stage, for varying number of input classes in each stage.

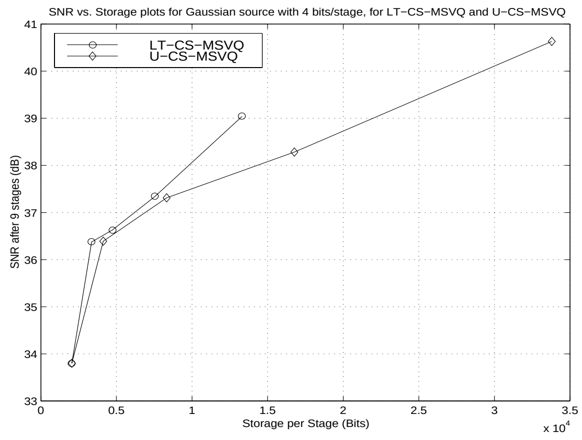

Figure 7. SNR vs. storage/stage for Gaussian data, for LT-CS-MSVQ and U-CS-MSVQ after 9 stages, with 4-dim vectors, at 4 bits/stage.

4.2 VSPIHT Image Coding

The CS-MSVQ successive refinement methodology is applied to a VSPIHT grayscale image coder. Magnitude classified U-CS-MSVQ and LT-CS-MSVQ chains are designed for 4-dim wavelet vectors in 2x2 blocks of each subband. All vectors are scaled uniformly before coding so that their maximum magnitude is a fixed at a parameter R_{-1}. The bit-allocation schedule, the codebook fan-out, and the thresholds for classification, for a total of 9 classes are shown in Table 1. The $b|\phi$ notation in the 3rd column of Table 1 refers to a bit-allocation of b bits for a stage, with a fan-out of ϕ to the next stage.

Table 1. Bit-allocation for 2 x 2 VSPIHT with CS-MSVQ

Class	Threshold R_i R_{-1} = 8192	U-CS-MSVQ and LT-CS-MSVQ Bit-allocation and Fan-out
0	4096	4|16, 6|32, 6|32, 5|32, 5|32, 5|32, 4|32, 4|32, 4|32
1	2048	5|32, 6|32, 6|32, 5|32, 5|32, 4|32, 4|32, 4|32
2	1024	5|32, 6|32, 5|32, 5|32, 4|32, 4|32, 4|32
3	512	5|32, 6|32, 5|32, 4|32, 4|32, 4|32
4	256	6|32, 6|32, 4|32, 4|32, 4|32
5	128	6|32, 5|32, 4|32, 4|32
6	64	5|32, 5|32, 4|32
7	32	5|32, 4|32
8	16	4|16

The results obtained by 2 x 2 VSPIHT with U-CS-MSVQ, for the *Baboon* image are shown in Table 2, compared against those obtained by scalar SPIHT, and VSPIHT with tree-multistage VQ (TS-MS-VQ) [6]-[8]. The corresponding results for the *Barbara* image are shown in Table 3. Examination of the results reveal that for VSPIHT coding there is no significant loss in efficiency by the use of CS-MSVQ with uniform bit-allocation and smaller codebooks, over TS-MSVQ with staggered bit-allocation and larger codebooks. In fact, the results with CS-MSVQ are better than TS-MSVQ for the *Baboon* image. The storage requirements are significantly reduced with both CS-MSVQ structures. Also, between U-CS-MSVQ and LT-CS-MSVQ, the latter requires significantly lower storage than the former, for the same rate and fan-out. Furthermore, these structures are very flexible, and a broad range of trade-offs between storage and efficiency are possible.

Table 2. PSNR (dB) vs. Bitrate (BPP) results for *Baboon* with 2x2 VSPIHT

BPP	SPIHT	VSPIHT (TS-MSVQ)	VSPIHT (U-CS-MSVQ)	VSPIHT (LT-CS-MSVQ)
0.2	22.69	22.77	22.80	22.79
0.4	24.65	24.88	24.91	24.88
0.6	26.50	26.57	26.67	26.64
0.8	27.84	27.94	28.04	28.00
1.0	29.15	29.30	29.33	29.30

Table 3. PSNR (dB) vs. Bitrate (BPP) results for *Barbara* with 2x2 VSPIHT

BPP	SPIHT	VSPIHT (TS-MSVQ)	VSPIHT (U-CS-MSVQ)	VSPIHT (LT-CS-MSVQ)
0.2	26.64	27.34	27.28	27.27
0.4	30.08	30.58	30.53	30.51
0.6	32.50	33.03	32.75	32.70
0.8	34.63	34.80	35.00	34.93
1.0	36.37	36.59	36.49	36.40

5. REFERENCES

[1] J. M. Shapiro, “Embedded image coding using zerotrees of wavelet coefficients,” *IEEE Trans. Signal Processing*, vol. 41, no. 12, pp. 3445-62, Dec. 1993.

[2] A. Said and W. A. Pearlman, “A new, fast, and efficient image codec based on set partitioning in hierarchical trees,” *IEEE Trans. Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243-50, June 1996.

[3] E. A. B. Da Silva, D. G. Sampson, M. Ghanbari, “A successive approximation vector quantizer for wavelet transform image coding,” *IEEE Trans. Image Processing*, vol. 5, no. 2, pp. 299-310, Feb. 1996.

[4] J. Knipe, X. Li, and B. Han, “An improved lattice vector quantization scheme for wavelet compression,” *IEEE Trans. Signal Processing*, vol. 46, no. 1, pp. 239-43, Jan 1998.

[5] D. Mukherjee and S. K. Mitra, “Vector set partitioning with successive refinement Voronoi lattice VQ for embedded wavelet image coding,” *Proc. IEEE Int. Conf. on Image processing*, Chicago, Illinois, vol. 1, pp. 107-11, Oct 1998.

[6] D. Mukherjee and S. K. Mitra, “Vector set partitioning with classified successive refinement VQ for embedded wavelet image and video coding,” *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Seattle, Washington, vol. 5, pp. 2809-12, May 1998.

[7] D. Mukherjee and S. K. Mitra, “Arithmetic Coded Vector SPIHT with classified tree-multistage VQ for color image coding,” *IEEE Workshop on Multimedia Signal Processing*, Los Angeles, California, Dec 1998.

[8] D. Mukherjee, *Vector Set Partitioning and Successive Refinement VQ for Wavelet Image and Video Compression*, Ph.D. Thesis, University of California, Santa Barbara, Aug 1999.

[9] C. F. Barnes, S. A. Rizvi, N. M. Nasrabadi, “Advances in residual vector quantization: A review,” *IEEE Trans. Image Processing*, vol. 5, no. 2, Feb 1996.

[10] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Boston, MA: Kluwer, 1992.

[11] C. F. Barnes, R. L. Frost, “Vector quantizers with direct sum codebooks,” *IEEE Trans. on Information Theory*, vol. 39, no. 2, pp. 565-80, March 1993.

[12] W.-Y Chan, S. Gupta, A. Gersho, “Enhanced multistage vector quantization by joint codebook design,” *IEEE Trans. Communications*, vol. 40, no. 11, pp. 1693-97, Nov. 1992.

[13] A. Buzo, A. H. Gray, R. M. Gray, J. D. Markel, “Speech coding based upon vector quantization,” *IEEE Trans. Acoustics, Speech and Signal Processing*, vol. ASSP-28, pp. 562-74, Oct. 1980.

[14] W.-Y Chan, A. Gersho, “Constrained-storage quantization of multiple vector sources by codebook sharing,” *IEEE Trans. Communications*, vol. 39, no. 1, pp. 11-13, January 1991.

[15] S. Ramakrishnan, K. Rose, A. Gersho, “Constrained-storage vector quantization with a universal codebook,” *IEEE Trans. Image Processing*, vol. 7, no. 6, pp. 785-93, June 1998.