RATE DISTORTION OPTIMIZED ADAPTIVE MULTISCALE VECTOR QUANTIZATION

Murilo B. de Carvalho

Depto. de Eng. de Telecomunicações
Universidade Federal Fluminense
R. Passos da Pátria, 156
Niteroi - RJ, 24210-240, BRASIL
murilo@lps.ufrj.br

Eduardo A. B. da Silva

PEE/COPPE/DEL/EE
Universidade Federal do Rio de Janeiro
Cx. P. 68504,
Rio de Janeiro, RJ, 21945-970, BRASIL
eduardo@lps.ufrj.br

Weiler Alves Finamore

CETUC, PUC-Rio
Rio de Janeiro, RJ, BRASIL
weiler@cetuc.puc-rio.br

ABSTRACT

We present a new rate-distortion optimized algorithm for adaptive vector quantization. It is based in approximate matching of recurrent patterns. In our approach, the input vector is segmented in variable-sized blocks. The blocks are encoded using a set of codebooks, one for each block size. The codebooks are updated while the data is encoded, with no need for any side information. Also, no prior training is required. We use dynamic programming techniques to optimize the segmentation tree. It performs well for a wide class of sources, with very good results for highly non-stationary sources, like compound documents.

1. INTRODUCTION

In an earlier work [1], we described a new class of universal multi-dimensional lossy data compression algorithms, the UMMP (Universal Multiscale Matching Pursuits). These algorithms employ a dictionary of vectors of different lengths and a recursive parsing procedure to encode variable-length segments of the input vector. This can be seen as adaptive variable dimension vector quantization (VQ). Previous work on this include [2, 3, 4]. The distinctive features of UMMP are its codebook updating method, that requires no side information, and its multiscale approach. The UMMP algorithm attempts to encode an input vector using one element with the same length in the dictionary. If the distortion in the approximation is above a given threshold, the input vector is split in two segments and the whole procedure is recursively repeated, with each new segment interpreted as a new input vector, until the distortion falls below a threshold. The dictionary is updated by the inclusion of the concatenation of previously encoded vectors, in the spirit of the lossless Lempel-Ziv algorithm (LZ)[5]. Differently from LZ however, the segmentation of UMMP can be easily performed in multi-dimensional arrays instead of vectors. Also, whenever a new vector of length N is obtained by concatenation, UMMP makes predictions of what vectors should be included in the dictionary at all other lengths.

One weakness of UMMP is that the segmentation created using decisions based on threshold comparisons is sub-optimal. In this work, we apply rate-distortion concepts to optimize the segmentation in an adaptive VQ closely related to UMMP. Our adaptive quantizer uses a set of $K = \log_2(N)$ dictionaries $\mathcal{D}^{(k)}$ to encode the data. An input vector $\mathbf{X} = (\ X_0 \quad \ldots \quad X_{N-1}\ )$ is parsed in L segments, $\mathbf{X} = (\ \mathbf{X}^{l_0} \quad \ldots \quad \mathbf{X}^{l_{L-1}}\ )$ of lengths $\ell(\mathbf{X}^{l_m})$, $m = 0, 1, \ldots, L-1$. These lengths are specified by a binary segmentation tree $\mathcal{S}$ as in figure 1.

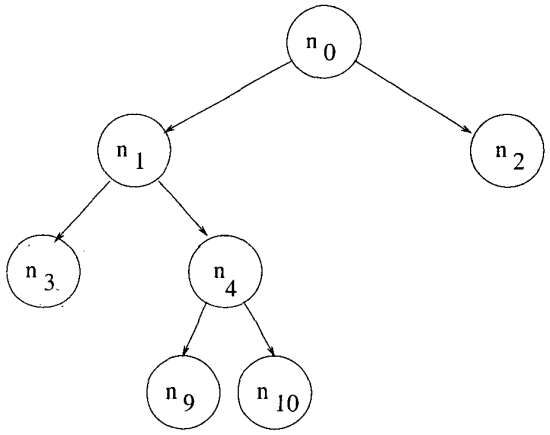


Fig. 1. A binary segmentation tree.

A node of the tree is denoted by n_l. It can have the two children nodes n_{2l+1} and n_{2l+2}, or no child at all. A node with no child is a leaf. The root node n_0 of the segmentation tree corresponds to a segment of length N. Its two children, n_1 and n_2 are associated with the two segments of length $N/2$. A node at depth p represents a segment of length $2^{-p}N$. Therefore, the segmentation is given by the leaves, and the length $\ell(\mathbf{X}^{l_m})$ of the segment $\mathbf{X}^{l_m}$ is the length of the corresponding leaf node in the tree. For

0-7803-6725-1/01/$10.00 ©2001 IEEE

example, the segmentation represented by the tree in figure 1 is $\mathbf{X} = \left(\mathbf{X}^3 \quad \mathbf{X}^9 \quad \mathbf{X}^{10} \quad \mathbf{X}^2 \right)$ and the lengths of the segments are respectively: $N/4, N/8, N/8, N/2$.

Each segment $\mathbf{X}^{l_m}$ is encoded using one element of the corresponding dictionary $\mathcal{D}^{(k_m)} = \left\{ \mathbf{S}_0^{(k_m)}, \ldots, \mathbf{S}_{M_{k_m}-1}^{(k_m)} \right\}$, where $k_m = \log_2 \left(\ell(\mathbf{X}^{l_m}) \right)$. That is, the input vector is approximated as $\hat{\mathbf{X}} = \left(\mathbf{S}_{i_0}^{(k_0)} \quad \ldots \quad \mathbf{S}_{i_{L-1}}^{(k_{L-1})} \right)$. The algorithm updates its dictionaries while encoding the input data as follows: Whenever the segments corresponding to the two children of node n_j have been encoded, the vector resulting from their concatenation $\hat{\mathbf{X}}^j = \left(\hat{\mathbf{X}}^{2j+1} \quad \hat{\mathbf{X}}^{2j+2} \right)$ is included in the dictionaries. This is in the spirit of the lossless Lempel-Ziv algorithm [5]. In order to update all dictionaries, the length of this vector is changed by a scale transformation $T_{2^k}^{\ell(\hat{\mathbf{X}}^j)}[\hat{\mathbf{X}}^j]$ to fit in each dictionary $\mathcal{D}^{(k)}$. The scale transformation is a function $T_N^M : \mathbb{R}^M \rightarrow \mathbb{R}^N$ that maps a vector of length M into a vector of length N. The algorithm outputs an integer sequence consisting of the dictionary indexes i_m and a sequence of binary flags b_n that specify the segmentation tree. The flags represent $\mathcal{S}$ as a series of binary decisions, in a top-down fashion. We use the binary flag 0 to indicate splitting and the flag 1 to indicate a leaf node. For example, the tree in figure 1 is represented by the sequence of flags 0,0,1,0,1,1,1.

2. THE OPTIMIZATION OF THE SEGMENTATION TREE

Each leaf node n_l is associated with a segment of the input vector $\mathbf{X}^l$ that is represented by an element $\mathbf{S}_{i_l}^{(k_l)}$, where $k_l = \log_2(\ell(\mathbf{X}^l))$. Therefore we can evaluate the distortion:

$$\begin{aligned} D(n_l) &= \|\mathbf{X}^l - \mathbf{S}_{i_l}^{(k_l)}\|, \\ k_l &= \log_2(\ell(\mathbf{X}^l)) \end{aligned} \tag{1}$$

The rate $R(n_l)$ is the rate needed to specify the index i_l, that is:

$$R(n_l) = -\log_2(Pr(i_l|k_l)) \tag{2}$$

where $Pr(i_l|k_l)$ is the probability of occurrence of index i_l in the dictionary of scale k_l.

The overall distortion is:

$$D(\mathcal{S}) = \sum_{n_l \in \mathcal{S}_\mathcal{L}} D(n_l) \tag{3}$$

where $\mathcal{S}_\mathcal{L}$ is the set of leaf nodes of $\mathcal{S}$.

The number of bits needed to encode this approximation is the rate $R(\mathcal{S})$, and is given by:

$$R(\mathcal{S}) = R_t(\mathcal{S}) + \sum_{n_l \in \mathcal{S}_\mathcal{L}} R(n_l) \tag{4}$$

where $R_t(\mathcal{S})$ is the rate required to specify the segmentation tree.

The best segmentation $\mathcal{S}^*$, in an R-D sense, leads to the minimum rate $R(\mathcal{S})$ given that the distortion $D(\mathcal{S})$ is no greater than a target distortion D^* or, alternatively, the minimum distortion at rate R^*. This is a constrained minimization problem stated as:

$$\begin{aligned} \mathcal{S}^* &= \arg \min_{\mathcal{S} \in \mathcal{S}_{R^*}} D(\mathcal{S}), \\ \mathcal{S}_{R^*} &= \{\mathcal{S} : R(\mathcal{S}) = R^*\} \end{aligned} \tag{5}$$

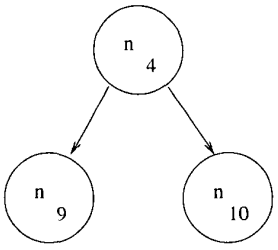


Fig. 2. The sub-tree $\mathcal{S}(n_4)$.

To find $\mathcal{S}^*$ we can find the solution to a related unconstrained problem introducing a Lagrange multiplier λ. It is well known that if we find the minimum of the Lagrangian cost $J(\mathcal{S}) = D(\mathcal{S}) + \lambda R(\mathcal{S})$, we also find the solution to the constrained problem when we choose $R(\lambda) = R^*$ [6]. That is:

$$\begin{aligned} \mathcal{S}^* &= \arg \min_{\mathcal{S}} J(\mathcal{S}) \\ &= \arg \min_{\mathcal{S}} \left(\sum_{n_l \in \mathcal{S}_\mathcal{L}} D(n_l) \right) + \lambda \left(R_t(\mathcal{S}) + \sum_{n_l \in \mathcal{S}_\mathcal{L}} R(n_l) \right) \\ &= \arg \min_{\mathcal{S}} \lambda R_t(\mathcal{S}) + \sum_{n_l \in \mathcal{S}_\mathcal{L}} (D(n_l) + \lambda R(n_l)) \\ &= \arg \min_{\mathcal{S}} \lambda R_t(\mathcal{S}) + \sum_{n_l \in \mathcal{S}_\mathcal{L}} J(n_l) \end{aligned} \tag{6}$$

where $J(n_l) = D(n_l) + \lambda R(n_l)$.

A sub-tree $\mathcal{S}(n_l)$ of $\mathcal{S}$ at node n_l is the binary tree with all the nodes of $\mathcal{S}$ having n_l as the root node. Figure 2 illustrates the sub-tree $\mathcal{S}(n_4)$ of the binary tree in figure 1. We denote $\mathcal{S} - \mathcal{S}(n_l)$ the tree obtained from $\mathcal{S}$ by pruning the sub-tree $\mathcal{S}(n_l)$.

If the Lagrangian costs $J(n_l)$, associated with the approximation of each segment $\mathbf{X}^l$, are independent, then the Lagrangian cost of two sub-trees $J(\mathcal{S}(n_l))$ and $J(\mathcal{S}(n_m))$ are also independent, as long as all nodes of both sub-trees are different. Then a fast search algorithm, similar to [4], can be implemented considering that if $J(n_l) \leq J(\mathcal{S}(n_{2l+1})) + J(\mathcal{S}(n_{2l+2}))$ then the sub-trees $\mathcal{S}(n_{2l+1})$ and $\mathcal{S}(n_{2l+2})$ must be pruned from $\mathcal{S}$ in order to decrease the cost. Unfortunately this is not the case with our VQ, since the costs $J(n_l)$ are coupled by the dictionary updating procedure. However, if the initial dictionaries are large enough, the contribution to the minimization of $J(n_l)$ due to the dictionary update can be negligible. In practical VQ implementations, we tend to use an upper limit to the size M of the input vector $\mathbf{X}$ and to the number of vectors in the dictionaries, to deal with the finite amount of memory available. Therefore, the input data is broken in blocks of size M that are sequentially processed by the VQ. Although not true for the first blocks, the dictionary eventually grows large enough for the Lagrangian costs $J(n_l)$ to be almost decoupled. In this sense, one could use the algorithm in [4] to find an approximate R-D-optimal solution.

However, if we want to use relatively large blocklengths or the dictionary is too small (as happens at very low rates), we must use an algorithm that takes into account the impact of the dictionary-updating procedure. The dictionary is updated by the inclusion of the concatenation of previously encoded segments. Therefore, if we choose to prune a sub-tree, the impact on the cost is not re-

stricted to that sub-tree, but can affect all nodes that are to the right of the sub-tree. That is, if we prune a sub-tree, we might remove from the dictionary an element that would otherwise be used later to approximate an input segment, therefore increasing the cost. The idea is to prune a sub-tree only if the potential increase in the cost of subsequent nodes, due to the removal of some vectors from the dictionary, is not greater than the reduction in the cost provided by the pruning. The algorithm is described below.

step 1 Initialize $\mathcal{S}$ as the full tree of depth $\log_2(M) + 1$.

step 2 Make $J_i = \infty, i = M-1, M, \ldots, 2M-2$.

step 3 Make $p = \log_2(M)$.

step 4 For each node $n_l \in \mathcal{S}$ at depth p, that is $l \in [2^{p-1} - 1, 2^p - 2]$, evaluate $J_l = J(n_l) + \lambda R_{1_l}$ where $J(n_l)$ is the cost to represent the input segment associated with the node n_l and R_{1_l} is the rate needed to indicate that the node n_l is a leaf. Evaluate $\Delta J_l = \sum_{n_r \in \mathcal{S} - \mathcal{S}(n_l)} J(n_r) - \sum_{n_r \in \mathcal{S} - \mathcal{S}(n_l)} J'(n_r)$, where $J'(n_l)$ is computed using the dictionary without $\hat{\mathbf{X}}^l = \begin{pmatrix} \hat{\mathbf{X}}^{2j+1} & \hat{\mathbf{X}}^{2j+2} \end{pmatrix}$. If $J_l - J_{2l+1} - J_{2l+2} - \lambda R_{0_l} \leq \Delta J_l$ then prune nodes n_{2l+1} and n_{2l+2} (R_{0_l} is the rate needed to indicate splitting, and J_{2l+1}, J_{2l+2} were evaluated in the previous iteration with $p+1$). Otherwise, the cost of node n_l is updated with $J_{2l+1} + J_{2l+2} + \lambda R_{0_l}$. Make $p = p - 1$.

step 5 Repeat step 4 until $p = 0$.

It is interesting to consider why ΔJ_l is evaluated for all the nodes, since only the leaf nodes contribute to the total cost. The idea of the algorithm is to prune only when we are sure that the cost will not increase. The computation of ΔJ_l must be conservative because we don't know, at the time we are making a decision at node n_l, which nodes will be the leaves (the current leaves may be pruned later). When we evaluate $\Delta J(n_l) > 0$, we decide not to prune. However, the nodes that affected the decision might be pruned later. Therefore, the whole procedure must be repeated to improve the segmentation, until convergence.

3. EXPERIMENTAL RESULTS

The algorithm for R-D optimization of the segmentation tree described in the previous section was implemented and applied to lossy compress gray-scale still images. The segmentation was adapted to the two-dimensional character of the source as follows: The node n_0 corresponds to a 16 × 16 block. The nodes at depth 1 correspond to blocks of size 8 × 16. The nodes at depth 2 correspond to 8 × 8 blocks and so on. A node at depth p corresponds to a block of size $2^{\lfloor(\frac{8-p}{2})\rfloor} \times 2^{\lfloor(\frac{9-p}{2})\rfloor}$ ($\lfloor x \rfloor$ is the largest integer that is smaller than or equal to x). The scale transformation was implemented using classical sampling rate conversion procedures [8]. The integer sequence of dictionary indexes i_n was coded by an adaptive arithmetic coder with an independent model for each scale. The sequence of flags b_n was coded by the arithmetic coder with different models for each depth. The rates $R(n_l)$, R_{0_l} and R_{1_l} were estimated using the logarithm of the relative frequencies of occurrence of the symbols, used by the arithmetic coder models. The arithmetic coder used is based on the one in [9].

The algorithm was applied to the image Lena 512 × 512 and to the image pp1209 512 × 512, shown in figure 3a (due to file size limitations only a 400 × 400 window of it is shown). This image was scanned from the *IEEE Transactions on Image Processing*,

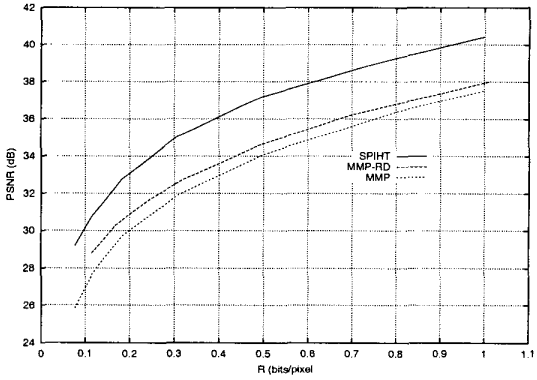


Fig. 4. R-D performance with LENA 512 × 512.

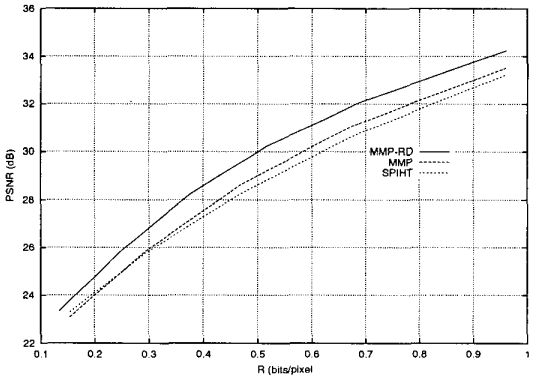


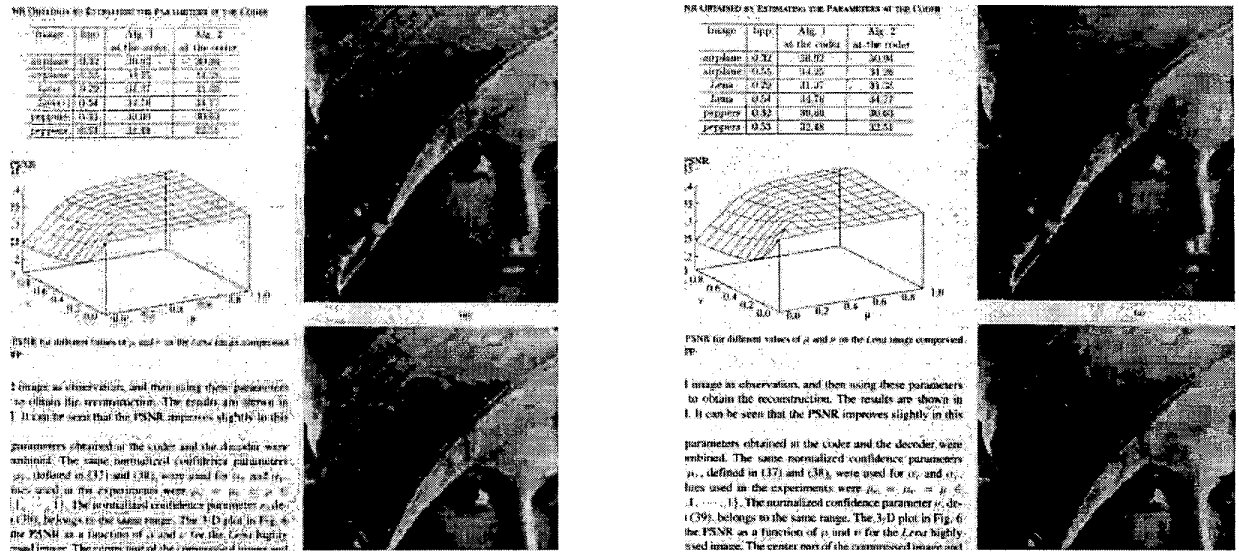
Fig. 5. R-D performance with pp1209 512 × 512.

Volume 9, number 7, July 2000, page 1209. It is a compound image of a compressed grayscale Lena with text and graphics. Figures 4 and 5 show the Peak Signal to Noise Rate (PSNR) versus the rate in bits/pixel obtained with these images for the original and enhanced algorithm. Figure 3b shows the reconstructed image using the proposed algorithm at 0.51 bits/pixel. The rendition of the text and the graphics is quite good. Results for the SPIHT [7] algorithms are also shown.

The improvement of the R-D optimized algorithm over the original one is clear, corresponding to a gain of about 1 dB in PSNR. The performance with Lena is 2 dB worse than that of the SPIHT algorithm with Lenna. However the propose algorithm outperforms SPIHT by 1 dB with the compound image. With text-only documents, our algorithm outperforms SPIHT by 5 dB.

4. CONCLUSION

We presented a new algorithm for adaptive vector quantization. It is similar to UMMP, an algorithm for universal lossy compression previously presented [1], but with an R-D optimized segmentation tree. Different from classical VQ approaches, it it has a universal flavor as it builds the dictionary while encoding the input data, obviating the need for previous codebook training. The dictio-



(a) (b)

Fig. 3. Image pp1209: (a) original; (b) compressed at 0.51 bits/pixel.

nary updating technique adopted is such that there is no need for any side information. The algorithm segments the input data in variable-sized blocks. It uses multiple dictionaries, one for each block length. Its performance is quite promising. For example, even being just a VQ applied directly to the image, it can encode compound documents outperforming the wavelet based SPIHT coder by more than 1 dB.

5. REFERENCES

[1] M. B. Carvalho and E. A. B. Silva, "A universal multi-dimensional lossy compression algorithm", *1999 IEEE International Conference on Image Processing*, October 1999, Kobe, Japan.

[2] M. Effros, P. A. Chou, and R. M. Gray, "One-pass adaptive universal vector quantization," *Procedings of ICASSP'94*, Vol. 5, pp. 625-628, Adelaide, 1994.

[3] C. Chan and M. Vetterli, "Lossy compression of individual signals based on string matching and one pass codebook design," *Procedings of ICASSP'95*, pp. 2491-2494, Detroit, 1995.

[4] G. J. Sullivan and R. L. Baker, "Efficient quadtree coding of images and video," *IEEE Transactions on Image Processing*, vol.3, No. 3, pp. 327-331, May 1994.

[5] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. it-24, No. 5, pp. 530-536, September 1978.

[6] R. E. Blahut, "Principles and Practice of Information Theory" Addison-Wesley publishing Company, 1988.

[7] A.Said and W.A. Pearlman, "A new, fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.6, pp.243–250, June 1996.

[8] P. P. Vaidyanathan, "Multirate Systems and Filter Banks," Prentice-Hall Inc., 1993.

[9] T. C. Bell, J. G. Cleary, I. H. Witten, "Text Compression", Prentice Hall, 1990.