RATE DISTORTION OPTIMIZED ADAPTIVE MULTISCALE VECTOR QUANTIZATION

Murilo B. de Carvalho  
Depto. de Eng. de Telecomunicações  
Universidade Federal Fluminense  
R. Passos da Pátria, 156  
Niterói - 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 UMMMP (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 UMMMP are its codebook updating method, that requires no side information, and its multiscale approach. The UMMMP 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].

Fig. 1. A binary segmentation tree.

A node of the tree is denoted by $n_i$. It can have the two children nodes $n_{i+1}$ and $n_{i+2}$, 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 $f(X^m)$ of the segment $X^m$ is the length of the corresponding leaf node in the tree. For 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 UMMMP. Our adaptive quantizer uses a set of $K = \log_2(N)$ dictionaries $\mathcal{D}^{(k)}$ to encode the data. An input vector $X = (X_0 \ldots X_{N-1})$ is parsed in $L$ segments, $X = (X^{10} \ldots X^{1N-1})$ of lengths $f(X^m)$, $m = 0, 1, \ldots, L - 1$. These lengths are specified by a binary segmentation tree $S$ as in figure 1.
example, the segmentation represented by the tree in figure 1 is
\[ \mathbf{X} = (\mathbf{X}^1, \mathbf{X}^9, \mathbf{X}^{10}, \mathbf{X}^5) \]
and the lengths of the segments are respectively: \( N/4, N/8, N/8, N/2 \).

Each segment \( \mathbf{X}^m \) is encoded using one element of the corresponding dictionary \( \mathcal{D}^{(km)} = \{ s_{(k_m)} \} \) where \( k_m = \log_2(\ell(\mathbf{X}^m)) \). That is, the input vector is approximated as
\[ \hat{\mathbf{X}} = (s_{(k_0)} \ldots s_{(k_{l-1})}) \]  

The algorithm updates its dictionaries while encoding the input data as follows: Whenever the segments corresponding to the two children of node \( n_i \) have been encoded, the vector resulting from their concatenation \( \mathbf{X}^j = (\mathbf{X}^{j+1}, \mathbf{X}^{j+4}) \) 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 \( T_N^{(k)}[\mathbf{X}^j] \) to fit in each dictionary \( \mathcal{D}^{(k)} \). The scale transformation is a function \( T_N^{(k)}: \mathbb{R}^M \rightarrow \mathbb{R}^N \) that maps a vector of length \( M \) into a vector of length \( N \). The algorithm outputs an integer sequence consisting of the dictionary indexes \( k_m \) and a sequence of binary flags \( b_k \) 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 0,1,0,1,1,1.

2. THE OPTIMIZATION OF THE SEGMENTATION TREE

Each leaf node \( n_i \) is associated with a segment of the input vector \( \mathbf{X}^j \) that is represented by an element \( s_{(k_i)} \), where \( k_i = \log_2(\ell(\mathbf{X}^j)) \). Therefore we can evaluate the distortion:
\[
D(n_i) = ||\mathbf{X}^j - s_{(k_i)}||,
\]
\[
k_i = \log_2(\ell(\mathbf{X}^j)) \tag{1}
\]

The rate \( R(n_i) \) is the rate needed to specify the index \( i \), that is:

\[
R(n_i) = -\log_2(Pr(i|k_i)) \tag{2}
\]

where \( Pr(i|k_i) \) is the probability of occurrence of index \( i \) in the dictionary of scale \( k_i \).

The overall distortion is:

\[
D(S) = \sum_{n_i \in S} D(n_i) \tag{3}
\]

where \( S \) is the set of leaf nodes of \( S \).

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

\[
R(S) = R_l(S) + \sum_{n_i \in S} R(n_i) \tag{4}
\]

where \( R_l(S) \) is the rate required to specify the segmentation tree.

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

\[
S^* = \arg \min_{S \in S_{R^*}} D(S),
\]

\[
S_{R^*} = \{ S : R(S) = R^* \} \tag{5}
\]

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

\[
S^* = \arg \min_{S \in S_{R^*}} J(S) \rightarrow \arg \min_{S \in S_{R^*}} \left( \sum_{n_i \in S} D(n_i) + \lambda \left( R_l(S) + \sum_{n_i \in S} R(n_i) \right) \right) \]

\[
= \arg \min_{S \in S_{R^*}} \lambda R_l(S) + \sum_{n_i \in S} (D(n_i) + \lambda R(n_i)) \tag{6}
\]

where \( J(n_i) = D(n_i) + \lambda R(n_i) \).

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

If the Lagrangian costs \( J(n_i) \), associated with the approximation of each segment \( \mathbf{X}^j \), are independent, then the Lagrangian cost of two sub-trees \( J(S(n_i)) \) and \( J(S(n_m)) \) 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 \( J(n_i) \leq J(S(n_{i+1})) \) and \( J(S(n_{i+2})) \) then the sub-trees \( S(n_{i+1}) \) and \( S(n_{i+2}) \) must be pruned from \( S \) in order to decrease the cost. Unfortunately this is not the case with our VQ, since the costs \( J(n_i) \) are coupled by the dictionary updating procedure. However, if the initial dictionaries are large enough, the contribution to the minimization of \( J(n_i) \) 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 \( J(n_i) \) 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 $S$ 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 $n_t \in S$ at depth $p$, that is $l \in [2^{p-1} - 1, 2^p - 2]$, evaluate $J_l = J(n_t) + \lambda R_{\hat{l}_t}$, where $J(n_t)$ is the cost to represent the input segment associated with the node $n_t$ and $R_{\hat{l}_t}$ is the rate needed to indicate that the node $n_t$ is a leaf. Evaluate $\Delta J_l = \sum_n_c \Delta J(n_r) - \sum_n_c \Delta J(n_r) J'(n_r)$, where $J'(n_r)$ is computed using the dictionary without $X^l = (X^{l+1}, X^{l+2})$. If $J_l - J_{3l+1} - J_{3l+2} - \lambda R_{\hat{l}_t} \leq \Delta J_l$ then prune nodes $3l, 3l+1$, and $3l+2$. Otherwise, the cost of node $n_t$ is updated with $J_{3l+1} + J_{3l+2} + \lambda R_{\hat{l}_t}$. Make $p = p - 1$.

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

It is interesting to consider why $\Delta J_l$ 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 $\Delta J_l$ must be conservative because we don’t know, at the time we are making a decision at node $n_t$, which nodes will be the leaves (the current leaves may be pruned later). When we evaluate $\Delta J(n_t) > 0$, 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 grayscale still images. The segmentation was adapted to the two-dimensional character of the source as follows: The node $n_0$ corresponds to a $16 \times 16$ block. The nodes at depth 1 correspond to blocks of size $8 \times 16$. The nodes at depth 2 correspond to $8 \times 8$ blocks and so on. A node at depth $p$ corresponds to a block of size $2^{(2^p-1)/2} \times 2^{(2^p-2)/2}$ ($\lceil x \rceil$ 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 $R(n_t)$, $R_{\hat{l}_t}$ and $R_{\hat{l}_t}$ 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 $512 \times 512$ and to the image pp1209 $512 \times 512$, shown in figure 3a (due to file size limitations only a $400 \times 400$ window of it is shown). This image was scanned from the IEEE Transactions on Image Processing, 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 has a universal flavor as it builds the dictionary while encoding the input data, obviating the need for previous codebook training. The dictio-
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


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