3D WEATHER RADAR IMAGE COMPRESSION USING MULTISCALE RECURRENT PATTERNS

Athayde L. V. Frauche[1]*, Murilo B. de Carvalho*[1]*, Eduardo A. B. da Silva*[2]

1. TET/CTC, Universidade Federal Fluminense. R. Passo da pátria, 156,
Niterói - RJ, 24210-240, BRAZIL
2. Universidade Federal do Rio de Janeiro, Cx. P. 68504,
Rio de Janeiro, RJ, 21941-972, BRAZIL
e-mails: frauche@hotmail.com, murilo@telecom.uff.br, eduardo@lps.ufrj.br

ABSTRACT

In this work we use the *Multidimensional Multiscale Parser* (MMP) algorithm to compress three-dimensional data from weather radar. MMP is based on *approximate multiscale pattern matching*. MMP encodes segments of an input signal using expanded and contracted versions of patterns stored in a dictionary. The dictionary is updated using concatenated and displaced versions of previously encoded segments, therefore MMP builds its own dictionary while the input data is being encoded. MMP can be adapted to compress signals of any number of dimensions, and has been successfully applied to compress two-dimensional image data and one-dimensional ECG and EMG signals. We show simulation results where our 3D version of MMP outperforms some of the best encoders in the literature.

Keywords: *Recurrent Pattern Matching, Multiscale Decomposition, Vector Quantization, 3D radar data*

1. INTRODUCTION

In this paper we report the results of the application of a universal lossy data compression scheme referred to as MMP (multidimensional Multiscale Parser) [1] to the compression of three-dimensional data from Doppler weather radar.

The Doppler Weather Radar (DWR) is a valuable tool for generation of meteorologic data which are essential for hydrology, hydrometeorology and weather forecasting among other applications. However, the huge amount of data generated by weather radar networks poses serious difficulties concerning storage, retrieval and real time transmission of DWR data. Therefore, a compression scheme that could deliver high-quality images while attaining high compression ratios is highly desirable. In some applications, the quality needed may be so high that lossless compression is mandatory. On the other hand, some band-limited applications, such as air-ground communications between ground bases and aircraft, require the best possible quality at a given bit budget. In this scenario, MMP emerges as a good candidate for a radar data compression tool, as its universal flavor has proved very useful when compressing many different kinds of data [2, 3]. Also MMP can be gracefully controlled, shifting from lossless to any degree of lossy operation points depending on the choice of a single parameter.

This paper is organized as follows. In section 2, the basic segmentation procedure used in the MMP algorithm is presented. In section 3, the 3D extension of MMP is described. Next, in section 4, we present experimental results and comparisons to other state-of-the-art image and video encoders. Finally, in section 5 we present our conclusions.

2. THE MMP ALGORITHM

The MMP is a lossy compression scheme based on multiscale pattern matching. It has a dictionary $\mathcal{D} = \{\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{L-1}\}$ of L variable length vectors $\mathbf{v}_i$ that it uses to encode variable-sized segments of an input vector $\mathbf{X}^0 = \begin{pmatrix} x(0) & x(1) & \ldots & x(N-1) \end{pmatrix}$, where the dimension N is a power of two. When attempting to encode $\mathbf{X}^0$, MMP breaks the input sequence in several non-overlapping segments $\mathbf{X}^j$, and represents each one by an element $\mathbf{v}_{i_j}$ in the dictionary $\mathcal{D}$ that will be used to replace $\mathbf{X}^j$. In the spirit of multiscale pattern matching, any vector $\mathbf{v}_i$ in $\mathcal{D}$ can be used to replace $\mathbf{X}^j$, regardless of its size. The segmentation is associated with a binary segmentation tree $\mathcal{T}$, where each node n_j corresponds to an input segment $\mathbf{X}^j$, as illustrated in figures 1 and 2.

Therefore, MMP's representation of the input vector is composed by the binary segmentation tree $\mathcal{T}$ as well as the dictionary indexes i_j of the elements $\mathbf{v}_{i_j}$ chosen to replace each input segment $\mathbf{X}^j$. For example, considering the segmentation illustrated in figure 1, the output generated by MMP would be the sequence $0, 0, 1, i_3, 0, 1, i_9, 1, i_{10}, 1, i_2$. The flag sequence 0010111 defines the chosen segmentation tree in a top-down fashion. As long as one knows which are the vectors associated with each index i_j (that is, both the encoder and the decoder have the same dictionary $\mathcal{D}$),

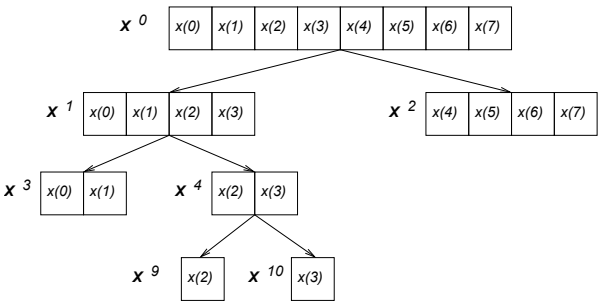

Fig. 1. A segmentation of the input vector $\mathbf{X}^0$

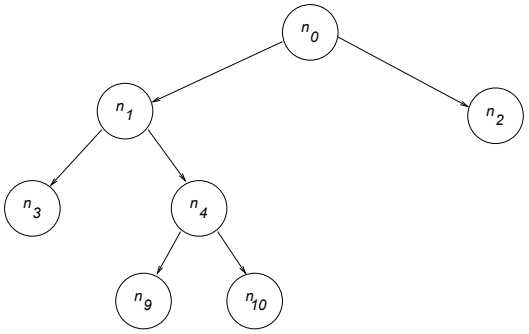

Fig. 2. A segmentation tree $\mathcal{T}$

the output sequence generated by MMP is easily decoded as follows: the decoder begins by reconstructing the segmentation tree from the segmentation flags. After that, the decoder knows the lengths associated with each index i_j, since each node n_j corresponds to a vector of known size. Then the decoder proceeds by replacing each node n_j by a correctly scaled version of the vector $\mathbf{v}_{i_j}$ in the dictionary $\mathcal{D}$.

The choice of the best vector and the best segmentation tree are based on the minimization of the Lagrangian cost $J(\mathcal{T})$, defined as:

$$J(\mathcal{T}) = \lambda R_{\mathcal{T}} + \sum_{n_j \in \mathcal{T}_{\mathcal{L}}} \left(D\left((n_j\right) + \lambda R\left((n_j\right)\right) \qquad (1)$$

where $\mathcal{T}_{\mathcal{L}}$ is the set of leaf nodes of $\mathcal{T}$, $D(n_j)$ is the distortion (usually the mean squared error) incurred by using $\mathbf{v}_{i_j}$ to replace $\mathbf{X}^j$, $R(n_j)$ is the rate needed to encode the index i_j and $R_{\mathcal{T}}$ is the rate needed to encode the segmentation tree $\mathcal{T}$.

It can be shown that the segmentation tree that minimizes the cost given by equation (1) also leads to the rate-distortion optimum solution [1].

The MMP algorithm as described so far operates with a fixed database, that is, the dictionary is static. A much better performance can be achieved if we allow the dictionary to adapt itself to the data being encoded. This is accomplished by updating the dictionary with concatenations of previously encoded segments. This resembles the way that the lossless Lempel-Ziv compression scheme [4] works. For example, referring to figure 1, as soon as the indexes i_9 and i_{10} are determined, we can build $\hat{\mathbf{X}}^9$ and $\hat{\mathbf{X}}^{10}$, the reconstruction versions of the input segments $\mathbf{X}^9$ and $\mathbf{X}^{10}$, by properly scaling the dictionary vectors $\mathbf{v}_{i_9}$ and $\mathbf{v}_{i_{10}}$. Then we have the reconstructed version of $\mathbf{X}^4$ by the concatenation of $\hat{\mathbf{X}}^9$ and $\hat{\mathbf{X}}^{10}$, that is $\hat{\mathbf{X}}^4 = \begin{pmatrix} \hat{x}(2) & \hat{x}(3) \end{pmatrix}$. We can then safely include $\hat{\mathbf{X}}^4$ in the dictionary, since the information used to generate this new vector is available both to the encoder and to the decoder, ensuring the proper synchronization of the dictionaries at both ends. The next update of the dictionary in this example will be done by including in it the concatenations of $\hat{\mathbf{X}}^3$ and $\hat{\mathbf{X}}^4$, as soon as the two segments are available. When MMP tries to represent an input segment $\mathbf{X}^j$ by one of the vectors in its dictionary $\mathcal{D}$, it first has to apply scale transformations to adjust the length of each vector in $\mathcal{D}$ so that they are equal to the length of $\mathbf{X}^j$. If the length of the input vector is N, the segmentation procedure can create vectors of lengths $N/2, N/4, \ldots, 1$. This implies that there are at most $1 + \log_2(N)$ different lengths or scales. Therefore, to save time, we could keep $1 + \log_2(N)$ copies of the dictionary, one at each scale, to avoid the computation of the scale transformation each time a match is attempted. This way, we only need to use the scale transformation when we are including a new vector in the dictionary. We denote a copy of the dictionary at scale $2^{-p}N$ as $\mathcal{D}^p$. If we are using this multiple codebook scheme, to include a new vector $\hat{\mathbf{X}}^j$ in the dictionary we must actually include $T_{2^{-p}N}[\hat{\mathbf{X}}^j]$ in $\mathcal{D}^p$ for $p = 0, 1, \ldots, \log_2(N)$.

3. THREE-DIMENSIONAL MMP

The segmentation procedure described in section 2 can be extended in many different ways when processing higher-dimensional input data. In [1], a simple two-dimensional segmentation is described, where an $N \times N$ input block $\mathbf{X}^0$ can be split in two $N/2 \times N$ sub-blocks $\mathbf{X}^1$ and $\mathbf{X}^2$. Each sub-block can be independently subdivided in $N/2 \times N/2$ sub-sub-blocks $\mathbf{X}^3$ and $\mathbf{X}^4$, $\mathbf{X}^5$ and $\mathbf{X}^6$, and so on. With this rule, there are $1 + 2\log_2(N)$ possible scales, namely $N \times N$, $N/2 \times N$, $N/2 \times N/2$, $N/4 \times N/2$, ..., 1×1. This simple rule could be extended to the three-dimensional case, yielding for $N \times N \times N$ input blocks the following different scales: $N \times N \times N$, $N/2 \times N \times N$, $N/2 \times N/2 \times N$, $N/2 \times N/2 \times N/2$, $N/4 \times N/2 \times N/2$, ..., $1 \times 1 \times 1$. However, there are other possibilities that can be explored. For instance, we could alternatively use the following possible scales: $N \times N \times N$, $N \times N \times N/2$, $N \times N/2 \times N/2$, $N/2 \times N/2 \times N/2$, $N/2 \times N/2 \times N/4$, ..., $1 \times 1 \times 1$.We implemented a three-dimensional version of MMP using these segmentation rules and found out that the performance was very sensitive to the specific choice we made. We concluded that a much better approach would be to allow MMP to choose, in a sub-block-by-sub-block basis, which direction should be split. In other words, when attempting to encode

an input block at scale $N \times N \times N$, MMP tries 3 splitting possibilities: two $N/2 \times N \times N$ blocks, two $N \times N/2 \times N$ blocks and two $N \times N \times N/2$ blocks. The best performer is chosen. In this case, the segmentation tree will be described in a top-down fashion by a sequence of quaternary flags: M, representing a match, A representing splitting in the first direction, B representing splitting in the second direction and C representing splitting in the third direction. For example, figure 3 illustrates the segmentation of a $4 \times 4 \times 4$ block in one $4 \times 4 \times 2$ plus two $4 \times 2 \times 2$ blocks. In this case, The segmentation three is described by the sequence of flags C,M,B,M,M.

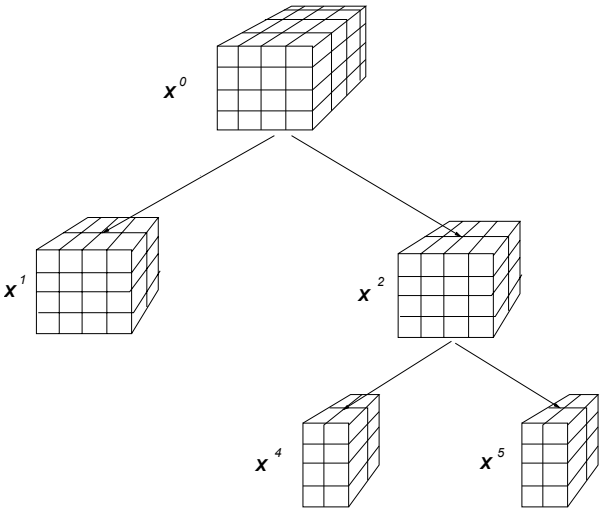

Fig. 3. A three-dimensional segmentation of the input block $\mathbf{X}^0$

The rate-distortion optimized segmentation tree can be obtained by the following recursive procedure:
$(J, \mathcal{T}_s) = \texttt{BestSeg}\left(\mathbf{X}^j, \mathcal{D}, \mathcal{T}, \lambda\right)$

`step 1` find $\mathbf{v}_{i_j} \in \mathcal{D}$ such that $J_0 = d\left(\mathbf{v}_{i_j}, \mathbf{X}^j\right) + \lambda R\left(i_j\right)$ is minimum. Here, $d\left(\mathbf{v}_{i_j}, \mathbf{X}^j\right)$ is the distortion when $\mathbf{v}_{i_j}$ replaces $\mathbf{X}^j$ and $R\left(i_j\right)$ is the rate needed to encode i_j.

`step 2` if scale is $1 \times 1 \times 1$ return J_0. Else, go to `step 3`.

`step 3` if n in current scale $n \times m \times p$ is greater than one, split $\mathbf{X}^j$ in two sub blocks $\mathbf{X}^{2j}$ and $\mathbf{X}^{2j+1}$ of scale $n/2 \times m \times p$. Evaluate
$(J_1, \mathcal{T}_1) = \texttt{BestSeg}\left(\mathbf{X}^{2j}, \mathcal{D}, \mathcal{T}, \lambda\right)$ and
$(J_2, \mathcal{T}_2) = \texttt{BestSeg}\left(\mathbf{X}^{2j+1}, \mathcal{D}, \mathcal{T}, \lambda\right)$.

`step 4` if m in current scale $n \times m \times p$ is greater than one, split $\mathbf{X}^j$ in two sub blocks $\mathbf{X}^{2j}$ and $\mathbf{X}^{2j+1}$ of scale $n \times m/2 \times p$. Evaluate
$(J_3, \mathcal{T}_3) = \texttt{BestSeg}\left(\mathbf{X}^{2j}, \mathcal{D}, \mathcal{T}, \lambda\right)$ and
$(J_4, \mathcal{T}_4) = \texttt{BestSeg}\left(\mathbf{X}^{2j+1}, \mathcal{D}, \mathcal{T}, \lambda\right)$.

`step 5` if p in current scale $n \times m \times p$ is greater than one, split $\mathbf{X}^j$ in two sub blocks $\mathbf{X}^{2j}$ and $\mathbf{X}^{2j+1}$ of scale $n \times m \times p/2$. Evaluate
$(J_5, \mathcal{T}_5) = \texttt{BestSeg}\left(\mathbf{X}^{2j}, \mathcal{D}, \mathcal{T}, \lambda\right)$ and
$(J_6, \mathcal{T}_6) = \texttt{BestSeg}\left(\mathbf{X}^{2j+1}, \mathcal{D}, \mathcal{T}, \lambda\right)$.

`step 6` Evaluate the total costs to encode the subtree of $\mathcal{T}$ with n_j as the root taking into account the cost required for the representation of the segmentation tree: $J_M = J_0 + R_M$, $J_A = J_1 + J_2 + R_A$, $J_B = J_3 + J_4 + R_B$, $J_C = J_5 + J_6 + R_C$; where R_M, R_A, R_B and R_C are the rates needed to encode the flags.

`step 7` Evaluate $J_{ST} = \min\left(J_A, J_B, J_C\right)$. If $J_A = J_{ST}$ make $\mathcal{T}_l = \mathcal{T}_1$ and $\mathcal{T}_r = \mathcal{T}_2$. If $J_B = J_{ST}$ make $\mathcal{T}_l = \mathcal{T}_3$ and $\mathcal{T}_r = \mathcal{T}_4$. If $J_C = J_{ST}$ make $\mathcal{T}_l = \mathcal{T}_5$ and $\mathcal{T}_r = \mathcal{T}_6$.

`step 8` If $J_0 \leq J_{ST}$, then the procedure returns $J = J_0$. Otherwise, the procedure updates $\mathcal{T}$ by including $\mathcal{T}_l$ and $\mathcal{T}_r$ as the two subtrees descending from node n_j, and then returns $J = J_{ST}$.

The performance gains we have obtained with this new segmentation rule came at the expense of an increased computational complexity. The optimization procedure described above performs at most six Vector Quantization operations to evaluate the Lagrangian cost of the node n_j. However, some of these calculations can be avoided, since there are different paths in the recursion that lead to the same input sub-block. For instance, when attempting to find the best option to encode the node n_3, the algorithm will perform two VQ using the same dictionary at scale $\frac{N}{2} \times \frac{N}{2} \times N$. For example, the procedure reaches the same input sub-block when it first segments the $N \times N \times N$ input block in two $\frac{N}{2} \times N \times N$, followed by a subdivision in four $\frac{N}{2} \times \frac{N}{2} \times N$ blocks, or alternatively, when it first segments the $N \times N \times N$ input block in two $N \times \frac{N}{2} \times N$, followed by a subdivision in four $\frac{N}{2} \times \frac{N}{2} \times N$ blocks. If the costs are calculated before the recursion begins, the repetitions can be avoided, and the number of VQ operations becomes proportional to the number of scales. The rule presented in [1] leads to $1 + 3\log\left(N\right)$ different scales, for input blocks of size $N \times N \times N$. Using our new segmentation, the number of different scales increases to $\left(1 + \log\left(N\right)\right)^3$ for the same block-size. Therefore, the number of vector quantization operations performed by the algorithm, which are responsible for the major part of the computational complexity [3], increases in the same proportion.

4. EXPERIMENTAL RESULTS

We have implemented MMP in software and applied it to compress three-dimensional data from the ground based U.S. NEXRAD (WSR-88D) Doppler weather radars. The three-dimensional input data was initially segmented in $8 \times 8 \times 4$

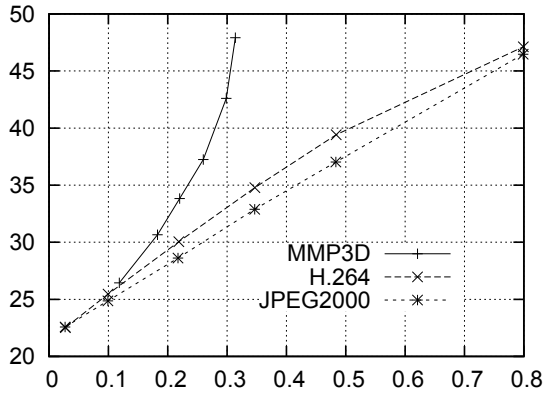

Fig. 4. Performance of 3D-MMP (PSNR versus bits/sample)

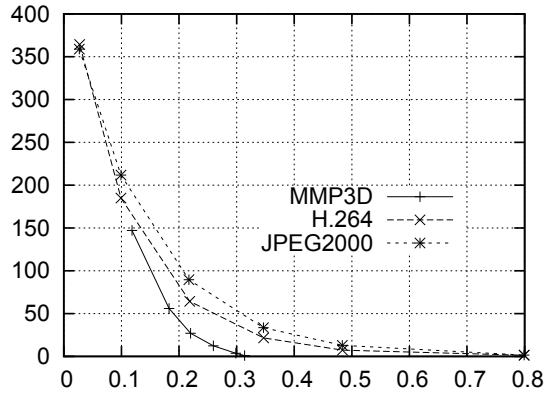

Fig. 5. Performance of 3D-MMP (MSE versus bits/sample)

blocks that were sequentially processed by our 3D-MMP algorithm. We used the dictionary obtained after the processing of a block as the initial dictionary of the next block. For the first block, we used an initial dictionary at scale $1 \times 1 \times 1$ composed by all the integers in the range $[0, 255]$. The initial dictionaries at the other 63 scales were obtained from the one at scale $1 \times 1 \times 1$ by appropriate scale transformations.

Figure 4 shows the simulation results, $PSNR$ versus rate, of the three-dimensional MMP for a three-dimensional volume of $512 \times 512 \times 4$ reflectivity samples obtained from the 4 lowest elevations of the record number 6500KHGX20000610_000110 of the NEXRAD database. Also shown are the results for the video coder H.264 [5], and for the image coder JPEG2000 [6]. The H.264 results were obtained using JM.96 reference software, configured for best performance using: GOP IPPP, rate-distortion optimization on, CABAC, variable bit rate, YUV format 4:0:0, Search range 400, using Haddamard transform. The input data was interpreted by H.264 as four frames of 512×512 pixels each (each frame corresponding to one elevation of the three dimensional radar data). Figure 5 shows the MSE versus rate.

As can be seen, the results are very good, our encoder outperforming the second best by 10 dB at 0.3 bits/sample.

5. CONCLUSIONS

We have applied a recently developed universal lossy compression algorithm called MMP to the problem of three-dimensional radar data coding. The MMP algorithm is based on approximate multiscale pattern matching, an extension of ordinary pattern matching. It uses a dictionary of patterns, that is adaptively built while the data is being coded, and a simple segmentation procedure that can be extended to operate on multidimensional data. We proposed a new three-dimensional segmentation procedure to be used in MMP that performed very well to encode three-dimensional radar data. Although our current implementation of MMP presents a very high computational complexity, its rate-distortion performance is state-of-the-art, outperforming the best encoders known.

6. REFERENCES

[1] M. de Carvalho, E. da Silva, and W. Finamore, "Multidimensional signal compression using multiscale recurrent patterns," *Elsevier Signal Processing*, , no. 82, pp. 1559–1580, November 2002.

[2] E. B. L. Filho, E. A. B. da Silva, M. B. de Carvalho, and F. S. Pinagé, "Universal image compression using multiscale recurrent patterns with adaptive probability model," *IEEE Transactions on Image Processing*, vol. 17, pp. 512–527, April 2008.

[3] E. B. L. Filho, E. A. B. da Silva, M. B. de Carvalho, W. S. S. Júnior, and J. Koiller, "Electrocardiographic signal compression using multiscale recurrent patterns," *IEEE Transactions on Circuits and Systems I*, vol. 52, no. 12, pp. 2739–2753, December 2005.

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

[5] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG (ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6), *Draft of Version 4 of H.264/AVC (ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 part 10) Advanced Video Coding)*, March 2005.

[6] D. S. Taubman and M.W. Marcelin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*, Kluwer Academic Publishers, 2001.