ADAPTIVE TREE-STRUCTURED LATTICE VECTOR QUANTIZATION FOR VIDEO CODING

Vincent Ricordel † and Moncef Gabbouj ‡
†MS-GESSY/ISITV, Université de Toulon et du Var,
Av. Georges Pompidou BP56,
83162 La Valette du Var, FRANCE
‡DMI/SPL, Tampere University of Technology,
P.O. Box 553, 33101-Tampere, FINLAND

e-mail: ricordel@univ-tln.fr, gabbouj@cs.tut.fi

ABSTRACT

The purpose of this paper is to introduce a new adaptive vector quantization (AVQ) for the compression of digital image sequences. In our previous studies [13, 14] we proposed a lattice vector quantizer (LVQ) based on the hierarchical packing of embedded truncated lattices. Now we investigate the capability of this LVQ for AVQ, through a scheme taking account of its treestructure. Precisely, in order to fit the spatiotemporal statistics of the image sequences, the codebook is designed using a training procedure and in two parts, with: a stump from several types of training sequences; some branches added according to the sequence to be coded. Experimental results are given with the adaptive LVQ taking place in a subband coder.

1 INTRODUCTION

Predictive coding and subband coding [11] are unavoidable decorrelation methods for the compression of digital image sequence. Because an hybrid coder aims to apply precisely two main rules: the non-transmission of predictable information and, the non-transmission of the no perceivable information by the human visual system (HVS). Fig 1 shows the generic coder for which our adaptive LVQ is devoted. Note that we doesn’t study in the paper the quantization aspect of the information performed by the motion estimation [16] between each image pair of the input sequence, but the VQ ¹ of the transformed prediction errors.

The inter-intra decorrelation steps shape the signal before its quantization. Because in each subimage the data are classified according to their frequent orientation and resolution. A separate VQ of each subimage is well-suited in order to exploit the dependencies between the transformed coefficients [3], and the bit allocation is performed such as, taking account of the HVS propriety, the low frequency subimages receive more bits (but the resulting gain is subjective). The monodimensional pdf of such subimage is commonly mapped by a Gaussian

¹From now VQ means vector quantizer as well as vector quantization

Generalized function whose narrowest highlights the correlation between the coefficients [3]. However the multidimensional signal obtained after the hybrid decorrelation chain is always nonstationary [6, 1]. Only an adaptive VQ [9, 7] is capable of adapting to changing source statistics as the coding progresses. Such as, from a representative training sequence, a very fast updating of the VQ codebook is achieved.

2 SUMMARY OF THE PREVIOUS STUDY

2.1 Context : Lattice Vector Quantization

The LVQ [5, 3] has been successfully introduced in order to overcome the LBG-type algorithm drawbacks [8]. The encoding, based on rounding and scaling operations, is simple and independant of the codebook size. There is no need to search among all the reproduction vectors, and practically no norms are computed. Because of the predefined structure of the lattice, there is no need to transmit the codebook and no training procedure is required to design it. But a lattice can only quantize an uniform source and, because of its infinite size, it must be truncate to index the codewords. The choice of the metric L² [10, 2] (resp. L¹ [5, 10, 3]) permits to shape the codebook into an hyper-sphere (resp. an hyper-square) and to count the points. As a result the basic LVQ is only adapted for symmetric Gaussian (resp. Laplacian) source distributions which map such a codebook. This restrictive modalization of the source offers some accurate and sophisticated methods to achieve the LVQ design, but the well-perform condition collapses considering a complex source coding at low bit rate. With the tree-structure LVQ (TSLVQ), based on embedding lattice strategy, simple procedures are implemented to overcome these drawbacks.

2.2 Tree-Structure Lattice Vector Quantization

Considering the lattices for which Conway and Sloane determined fast quantizing and decoding algorithms [4], the TSLVQ is based on the hierarchical packing of embedded truncated lattices. In [13, 14] we investigate its complete design with: the lattice truncation, the multistages procedure of quantization, the unbalanced
tree-structured design and the determination of the best lattice respectively to this method. The detection and
the processing of the probable outlying input vectors are defined too.

The TSLVQ offers some original solutions to usual LVQ
drawbacks, with a space partition according to the
source distribution, and a simple labelling of the lattice
points. The resulting codebook automatically obtained
is well suited for prediction error coding.

2.3 Binary Allocation Method

The bit allocation problem occurs when you have to share
the bit resource between the subbands [15]. We
present briefly our solution with the TSLVQ.

Precisely it consists in minimizing the distortion D
subject to the constraint that the global rate R is under
a threshold \( R_d \). The transformation is supposed orthog-
inal and there are \( M \) subbands:

\[
\min D = \min \sum_{j=0}^{M-1} d_{j,i} \quad \text{with} \quad R = \sum_{j=0}^{M-1} r_{j,i} \leq R_d
\]

For each subband \( j \) there are some potential quantizers
\( q_{j,i} \) (each of them corresponds to a configuration of the
TSLVQ tree), \( d_{j,i} \) is the distortion for the rate \( r_{j,i} \). For a
combination of \( M \) quantizers (i.e., a separate TSLVQ for
each subband) we get a bipoint \((R,D)\) in the distortion
vs. rate space, and all the \( N^M \) combinations produce a
cluster (fig 4 shows an example). The problem becomes
the determination on the convex hull of this cluster of
the bipoint whose rate is just lower than \( R_d \). In order
to reduce the amount of calculations the Lagragian multiplier
\( \lambda \) is introduced to solve:

\[
\min(D + \lambda R) \iff \sum_{j=0}^{M-1} \min(q_{j,i} + \lambda r_{j,i})
\]

The complexity decreases because the reduction of the
distortion is now achieved separately from each sub-
band. The general form of the algorithm is:

1. The convex hull for each subimage is calculated.
   With the TSLVQ we obtain directly it when grow-
ing the tree.

2. The point on the global convex hull is determin-
ed. Its rate is just below \( R_d \).

For this last step the Shoham’s method [15] can be ap-
plicated. It is based on the calculation of singular values of
\( \lambda \) (i.e., the slopes of the lines that pass through the
consecutive points of the convex hull). So, from a first
point of the hull and by successive calculations of sin-
gular values, we get the global convex hull. The fig 4a
shows an experimental result with the TSLVQ. A draw-
back appears because there are some large gaps between
some points of this global convex hull. So we want to
modify the algorithm in order to get the “optimal quant-
zers” (see figure 4), namely the points just above the
convex hull. We proceed in two steps:

1. The previous algorithm permits to get two points
   of the convex hull that surrounding \( R_d \).

2. From these two points, the Shoham’s method [15]
   is used again in order to get a local portion of
   convex hull.

The curve titled “intermediary quantizers” on the fig 4b,
is achieved by calculating the portions of convex hulls
between each bi-point of the global convex hull. As a
result a lot of optimal quantizers are got.

3 ADAPTIVE VQ DESIGN

For the adaptive TSLVQ the codebook is designed using
a training procedure and in two parts:

1. A stump from several types of training
   sequences (this stump construction is achieved
   outline).

2. Some branches added according to the sequence to
   be coded (these branches will constitute the only
   transmitted information to update the codebook).

Because of the binary allocation that implies a tree
pruning, there are precisely four steps in the algorithm
(see fig 2):

1. Construction of a large stump.
2. Binary allocation from this stump (the
   corresponding threshold \( R_d \) is very low).
3. Addition of large branches.
4. Final binary allocation with the desired bit rate.

In order to update the codebook, only the steps 3 and
4 are carried out. For the greedy tree growing method
of the steps 1 and 3, see [13].

4 EXPERIMENTAL RESULTS

The coder (see fig 1) is MPEG based because the tools
are a block matching for the motion estimation and a
DCT for the transformation. The block size is \( 2 \times 2 \) so
four subbands are set up (see fig 3). The simulations are
made using QCIF image sequences, and the computer
is a SparcStation 20 (75 Mhz).

Images from four different sequences are used for the
stump design, but we only use the images of one se-
quence to add branches (table 3 shows typical numerical
results). A particular codebook is then applied to
code the sequence that was used for the branches addi-
tion (see table 1). The CPU time for an image coding
is about 1.8 s.

For comparison, the table 4 shows codebook design
results with classical TSLVQ, and the table 2 the cor-
responding coding results.

5 CONCLUSION

With adaptive TSLVQ we get, with respect to our pre-
vious work, a more accurate binary allocation, and a bet-
ter regularity for the reconstruction when decoding the
sequences. The transmitted information to refresh the
codebook is not large, and the CPU time to design or
update the codebook is short.
Figure 1: Hybrid coder. Ip: input image from the sequence; Ir: reconstructed image; e: prediction error image; eq: quantized prediction error image.

![Diagram of a hybrid coder](image)

**Table 1:** Coding of sequences with the adaptive TSLVQ.

<table>
<thead>
<tr>
<th>sequence name</th>
<th>number of images</th>
<th>average entropy [bpp]</th>
<th>average PSNR [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Miss America&quot;</td>
<td>150</td>
<td>0.273</td>
<td>36.46</td>
</tr>
<tr>
<td>&quot;Salesman&quot;</td>
<td>180</td>
<td>0.174</td>
<td>36.38</td>
</tr>
<tr>
<td>&quot;Carphone&quot;</td>
<td>180</td>
<td>0.281</td>
<td>34.12</td>
</tr>
</tbody>
</table>

**Table 2:** Coding of sequences with the classical TSLVQ.

<table>
<thead>
<tr>
<th>sequence name</th>
<th>number of images</th>
<th>average entropy [bpp]</th>
<th>average PSNR [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Miss America&quot;</td>
<td>150</td>
<td>0.134</td>
<td>36.57</td>
</tr>
<tr>
<td>&quot;Salesman&quot;</td>
<td>180</td>
<td>0.177</td>
<td>35.58</td>
</tr>
<tr>
<td>&quot;Carphone&quot;</td>
<td>180</td>
<td>0.281</td>
<td>34.12</td>
</tr>
</tbody>
</table>

Figure 2: Adaptive TSLVQ, the four steps of the codebook design.

![Diagram of the four steps of the codebook design](image)

Figure 3: Codebook: Vector shapes and maximal entropies before bit allocation (note the subband labels).

**References**


Figure 4: (a,b): Cluster of bi-points \((R, D)\) and hulls calculated for the binary allocation.

<table>
<thead>
<tr>
<th>Subband label</th>
<th>Training sequence size [images]</th>
<th>CPU time for construction [s]</th>
<th>Number of codevectors</th>
<th>Codebook entropy [bpp]</th>
<th>Codebook size [byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>0.64</td>
<td>9</td>
<td>0.036</td>
<td>0.062</td>
</tr>
<tr>
<td>B</td>
<td>150</td>
<td>1.18</td>
<td>48</td>
<td>0.034</td>
<td>0.037</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>1.19</td>
<td>48</td>
<td>0.036</td>
<td>0.037</td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td>1.19</td>
<td>48</td>
<td>0.036</td>
<td>0.037</td>
</tr>
</tbody>
</table>

Table 3: Codebook design with adaptive TSLVQ: numerical results.

<table>
<thead>
<tr>
<th>Subband label</th>
<th>Training sequence size [images]</th>
<th>CPU time for construction [s]</th>
<th>Number of codevectors</th>
<th>Codebook entropy [bpp]</th>
<th>Codebook size [byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>0.64</td>
<td>9</td>
<td>0.036</td>
<td>0.062</td>
</tr>
<tr>
<td>B</td>
<td>150</td>
<td>1.18</td>
<td>48</td>
<td>0.034</td>
<td>0.037</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>1.19</td>
<td>48</td>
<td>0.036</td>
<td>0.037</td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td>1.19</td>
<td>48</td>
<td>0.036</td>
<td>0.037</td>
</tr>
</tbody>
</table>

Table 4: Codebook design with classical TSLVQ and using the sequence "Salesman": numerical results.