TEMA Tend. Mat. Apl. Comput., **11**, No. 3 (2010), 193-203.

Methods to Accelerate a Competitive Learning Algorithm Applied to VQ Codebook Design

E.L. BISPO JR[1], Universidade de São Paulo, 05508-090, São Paulo, SP, Brasil.

C.R.B. AZEVEDO[2], Universidade Federal de Pernambuco, 50740-540, Recife, PE, Brasil.

W.T.A. LOPES[3], M.S. ALENCAR[4], Universidade Federal de Campina Grande, 58109-970, Campina Grande, PB, Brasil.

F. MADEIRO[5], Universidade Católica de Pernambuco, 50050-900, Recife, PE, Brasil.

Abstract. Codebook design plays a crucial role in the performance of signal processing systems based on vector quantization (VQ). This paper is concerned with methods for reducing the processing time spent by a competitive learning (CL) algorithm applied to VQ codebook design. Using analytical expressions for the number of operations (multiplications, additions, subtractions and comparisons) performed by the CL algorithm, it is shown that almost all the operations are due to the nearest neighbor search (NNS). Simulation results regarding image VQ show that simple modifications introduced in CL lead to considerable number clock cycles savings.

Keywords. Vector Quantization, Competitive Learning, Nearest Neighbor Search.

1. Introduction

Vector quantization (VQ) [5,6] has been widely used in signal compression [1,4,11]. It may be defined as a mapping Q from an input vector $\boldsymbol{x}$ belonging to the K-dimensional Euclidean space, $\mathbb{R}^K$, into a vector belonging to a finite subset W of $\mathbb{R}^K$, that is, $Q : \mathbb{R}^K \to W$.

The codebook $W = \{\boldsymbol{w}_i;\ i = 1,\ 2,\ \ldots,\ N\}$ is the set of codevectors (reconstruction vectors), K is the dimension of the codevectors and N is the codebook size. The mapping Q leads to a partitioning of $\mathbb{R}^K$ in N non-intercepting cells (Voronoi

[1]bispojr@ime.usp.br.
[2]crba@cin.ufpe.br.
[3]waslon@ieee.org
[4]malencar@iecom.org.br
[5]madeiro@dei.unicap.br

Recebido em 04 Maio 2009; Aceito em 30 Dezembro 2010.

regions) S_i, $i = 1,\ 2,\ \ldots,\ N$, such that $S_i = \{\boldsymbol{x} :\ Q(\boldsymbol{x}) = \boldsymbol{w}_i\} = \{\boldsymbol{x} :\ d(\boldsymbol{x}, \boldsymbol{w}_i) < d(\boldsymbol{x}, \boldsymbol{w}_j), \forall j \neq i\}$, in which $d(\cdot)$ is a distortion measure.

The code rate, which measures the number of bits per vector component, is given by $R = \frac{1}{K}\log_2 N$. In image coding, R is expressed in bits per pixel (bpp).

The purpose of the techniques for codebook design is to reduce (for a given rate R) the distortion introduced when the input vectors $\boldsymbol{x}$ are represented by their corresponding quantized versions $Q(\boldsymbol{x})$. The Linde-Buzo-Gray (LBG) algorithm [9] is the most widely used technique for codebook design. In the literature, a number of unsupervised learning algorithms has been applied to codebook design [3, 7, 8].

This paper presents two methods to reduce the time spent by a competitive learning (CL) algorithm in the process of VQ codebook design. Using analytical expressions for the number of operations (multiplications, additions, subtractions and comparisons) performed by the competitive algorithm, it is shown that almost all the operations are due to the nearest neighbor search (NNS). In other words, the determination of the winner consumes almost all the operations of the CL algorithm. Simulations results regarding image VQ show that considerable number of clock cycles savings can be obtained by simple modifications in the CL algorithm.

2. Computational Complexity of the VQ

In VQ the computational complexity of the minimum distortion encoding phase is a relevant problem. For encoding an input vector, the encoder must determine its distance to each of the N codevectors and must compare the distances in order to find the codevector closest to the input vector, that is, the nearest neighbor. In the conventional full search (FS) method, the encoding of an input vector requires N distance (distortion) computations and $N-1$ comparisons. When using the squared Euclidean distortion, that is,

$$d(\boldsymbol{x}, \boldsymbol{w}_i) = \sum_{j=1}^{K} (x_j - w_{ij})^2, \tag{2.1}$$

in which w_{ij} is the j-th component of the codevector $\boldsymbol{w}_i$ and x_j is the j-th component of the input vector $\boldsymbol{x}$, each distance computation requires K multiplications, K subtractions and $K-1$ additions. Thus, to encode each input vector, KN multiplications, KN subtractions, $(K-1)N$ additions and $N-1$ comparisons must be computed.

2.1. PDS algorithm

The partial distance search (PDS) algorithm, proposed by Bei and Gray in [2], is a method for reducing the computational complexity of the nearest neighbor search (encoding phase). In the PDS algorithm, the encoder takes the decision before completing the computation of the distance between the input vector (vector to be encoded) and a codevector.

The encoder decides that a codevector is not the nearest neighbor if, for some $j < K$, the accumulated distance (that is, the partial distance) for the first j samples of the input vector is greater than the smallest distance previously computed in the search. Then, the encoder stops the distance computation for that codevector and starts the distance computation for the next codevector. With this approach, the number of multiplications per sample is dramatically reduced. The PDS algorithm also leads to a reduction in the number of subtractions/additions per sample. Although PDS increases the number of comparisons, the global complexity of the nearest neighbor search is reduced.

The basic structure of the PDS algorithm is presented in Figure 1(a), in which *dist* denotes the distance (distortion), *distmin* denotes the minimum distance (that is, the smallest distance previously found), x_j is the j-th component of the K-dimensional input vector $\boldsymbol{x}$ and w_{ij} is the j-th component of the i-th codevector $\boldsymbol{w}_i \in \mathbb{R}^K$, $i = 1, \cdots, N$.

2.2. Torres-Huguet algorithm

Another method for reducing the computacional complexity of NNS is Torres-Huguet algorithm [12] which is discussed as follows. The Euclidean distance (see Equation 2.1) can be written as $d(\boldsymbol{x}, \boldsymbol{w}_i) = \|\boldsymbol{x}\|^2 + \|\boldsymbol{w}_i\|^2 - 2\sum_{j=1}^{K} x_j w_{ij}$.

In Torres-Huguet method, $\|\boldsymbol{w}_i\|^2$ $(1 \leq i \leq N)$ is precalculated and stored. The term $\sum_{j=1}^{K} x_j w_{ij}$ is upper bounded as $\sum_{j=1}^{K} x_j w_{ij} \leq x_{max} \sum_{j=1}^{K} w_{ij}$ (for $x_j \geq 0$ and $y_j \geq 0$), in which x_{max} is the maximum vector component of vector $\boldsymbol{x}$. In Torres-Huguet, $\sum_{j=1}^{K} w_{ij}$ is also calculated and stored. Hence, Torres-Huguet defines an auxiliar distance

$$d_1(\boldsymbol{x}, \boldsymbol{w}_i) = \|\boldsymbol{x}\|^2 + \|\boldsymbol{w}_i\|^2 - 2x_{max} \sum_{j=1}^{K} w_{ij}. \tag{2.2}$$

It is seen that $d_1(\boldsymbol{x}, \boldsymbol{w}_i) \leq d(\boldsymbol{x}, \boldsymbol{w}_i)$. Additionaly, $d_1(\boldsymbol{x}, \boldsymbol{w}_i)$ requires a smaller number of arithmetic operations when compared to $d(\boldsymbol{x}, \boldsymbol{w}_i)$. The algorithm is described in Figure 1(b).

3. Competitive Learning Algorithm

Let n_{tot} be the number of iterations (number of entire passages of the training set) of the competitive learning algorithm. Let M be the number of training vectors. After an initialization of the VQ codebook (that is, after initializing the K components of the N weight vectors), the competitive learning (CL) algorithm may be described as follows.

```
distmin = ∞; (a very large number)
for 1 ≤ i ≤ N
    dist = 0;
    for 1 ≤ j ≤ K
        dist = dist + (x_j − w_ij)^2;
        if (dist > distmin)
            break;
    next j;
    if(dist < distmin)
        index = i;
        distmin = dist;
next i;
```

(a) PDS algorithm.

```
Evaluate and store ‖w_i‖^2 of every codevector;
Evaluate and store 2 Σ_{j=1}^{K} w_ij of every codevector;

Evaluate x_max;
distmin = ∞;
for 1 ≤ i ≤ N
    dist = d_1(x, w_i);
    if(dist > distmin) continue;
    dist = d(x, w_i);
    if(dist > distmin) continue;
    index = i;        (codevector selected w_i)
    distmin = dist;
next i;
```

(b) Torres-Huguet algorithm.

Figure 1: Algorithms for nearest neighbor search.

In the CL algorithm (Figure 2), $\boldsymbol{x}(m)$ is the m-th vector from the training set, while $\boldsymbol{w}_i(n,m)$ and $\boldsymbol{w}_{i^*}(n,m)$ denote, respectively, the i-th codevector and the winner for the m-th training vector in the n-th iteration.

In the description of the algorithm,

$$d[\boldsymbol{x}(m), \boldsymbol{w}_i(n,m)] = \sum_{j=1}^{K} [x_j(m) - w_{ij}(n,m)]^2 \tag{3.1}$$

is the Euclidean distance between the vectors $\boldsymbol{x}(m)$ and $\boldsymbol{w}_i(n,m)$, in which $x_j(m)$ is the j-th component of vector $\boldsymbol{x}(m)$ and $w_{ij}(n,m)$ is the j-th component of vector $\boldsymbol{w}_i(n,m)$. In the winner updating expression, Δw_{i^*j} is the modification introduced in the j-th component of the winner, $\eta(n)$ is the learning rate or adaptation gain in the n-th iteration (with $0 < \eta(n) < 1$), w_{i^*j} is the j-th component of the winner and $\tilde{w}_{i^*j}$ is the updated version of the j-th component of the winner.

The learning rate decreases linearly as the iteration n increases and is kept constant during each iteration, that is, during each entire passage of the M training vectors. It is expressed as $\eta(n) = \eta(1) + (n-1)\frac{\eta(n_{\text{tot}}) - \eta(1)}{n_{\text{tot}} - 1}$, in which $\eta(1)$ and $\eta(n_{\text{tot}})$ denote two parameters of the competitive algorithm: the initial learning rate and the final learning rate, respectively. The other parameters are K (dimension), N (codebook size) and n_{tot} (number of iterations).

for $1 \leq n \leq n_{\text{tot}}$

for $1 \leq m \leq M$

find the winner $\boldsymbol{w}_{i^*}(n,m)$:

$i^* = \arg\min_i d[\boldsymbol{x}(m), \boldsymbol{w}_i(n,m)]$;

update the winner according to

$$\tilde{w}_{i^*j}(n,m) = w_{i^*j}(n,m) + \Delta w_{i^*j}(n,m), \quad (3.2)$$

in which

$$\Delta w_{i^*j}(n,m) = \eta(n)[(x_j(m) - w_{i^*j}(n,m)]. \quad (3.3)$$

Figure 2: CL algorithm.

4. Computational Load of NNS in CL Algorithm

A detailed analysis of the computational complexity of the CL algorithm is presented in [10]. Tables 1 and 2 summarize the expressions for the number of multiplications, additions, subtractions and comparisons performed by the algorithm.

This section examines the *computational load* of the NNS (which corresponds to the search for the winner) in the CL algorithm. For each operation (multiplication, subtraction, addition, comparison), the *computational load* of the NNS is

$$\text{Load} = \frac{\#\text{NNS}}{\#\text{TOT}}, \quad (4.1)$$

in which #NNS and #TOT denote, respectively, the number of operations performed in the nearest neighbor search (see Table 1, determination of the winner) and the total number of operations performed by the CL algorithm (see Table 2).

Table 1: Datailed description for the number of operations required by the CL algorithm.

Operation	**Determination of the winner**	**Updating of the winner**	**Computation of the learning rate**
mult.	$KNMn_{\text{tot}}$	KMn_{tot}	$n_{\text{tot}} - 1$
sub.	$KNMn_{\text{tot}}$	KMn_{tot}	$n_{\text{tot}} + 1$
ad.	$(K-1)NMn_{\text{tot}}$	KMn_{tot}	$n_{\text{tot}} - 1$
comp.	$(N-1)Mn_{\text{tot}}$	-	-
div.	-	-	1

Hence, the computational load of the number of multiplications is

$$\text{Load(mul)} = \frac{KNMn_{\text{tot}}}{[1 + (1+N)KM]n_{\text{tot}} - 1}. \quad (4.2)$$

Table 2: Analytical expressions for the number of multiplications, subtractions, additions, divisions and comparisons used by the CL algorithm.

Operation	**Analytical expression**
mult.	$[1+(1+N)KM]n_{\text{tot}}-1$
sub.	$[1+(1+N)KM]n_{\text{tot}}+1$
ad.	$[1+(K-1)NM+KM]n_{\text{tot}}-1$
div.	1
comp.	$(N-1)Mn_{\text{tot}}$

Typical codebook design uses sufficiently large training sets. In other words, high values of M are considered in practical codebook design. Thus, $[1+(1+N)KM]n_{\text{tot}} \gg 1$. Hence, Load(mul) $\approx \frac{KNMn_{\text{tot}}}{[1+(1+N)KM]n_{\text{tot}}}$. Observing that $[1+(1+N)KM]n_{\text{tot}} \approx (1+N)KMn_{\text{tot}}$, it follows that Load(mul) $\approx \frac{N}{1+N}$. It is observed that Load(mul) goes to 1 as N increases.

The computational load of the number of subtractions is given by

$$\text{Load(sub)} = \frac{KNMn_{\text{tot}}}{[1+(1+N)KM]n_{\text{tot}}+1}. \tag{4.3}$$

In practical codebook design, sufficiently large training sets (large M) are used. As a consequence, $[1+(1+N)KM]n_{\text{tot}} \gg 1$. Thus, Expression (4.3) can be written as Load(sub) $\approx \frac{KNMn_{\text{tot}}}{[1+(1+N)KM]n_{\text{tot}}}$. Since $[1+(1+N)KM]n_{\text{tot}} \approx (1+N)KMn_{\text{tot}}$, it follows that Load(sub) $\approx \frac{N}{1+N}$. Thus, Load(sub) goes to 1 as N increases.

The computational load of the number comparisons is given by

$$\text{Load(comp)} = \frac{(N-1)Mn_{\text{tot}}}{(N-1)Mn_{\text{tot}}}. \tag{4.4}$$

Hence, 100% of the comparisons performed by the CL algorithm are due to the nearest neighbor search.

The computational load of the number of additions is given by

$$\text{Load(add)} = \frac{(K-1)NMn_{\text{tot}}}{[1+(K-1)NM+KM]n_{\text{tot}}-1}. \tag{4.5}$$

For practical codebook design (for large M), it follows that $[1+(K-1)NM+KM]n_{\text{tot}}-1 \approx [1+(K-1)NM+KM]n_{\text{tot}}$. Then, Load(add) $\approx \frac{(K-1)NM}{1+(K-1)NM+KM}$. Since $1+(K-1)NM+KM \approx (K-1)NM+KM$, one has

$$\text{Load(add)} \approx \frac{(K-1)N}{(K-1)N+K}. \tag{4.6}$$

As an example, for $K=16$ and $N=32$ the computational load of additions is 99.79%. For typical values of K in image coding, it is observed that the computational load of additions goes to 100% as N increases.

5. Incorporating Fast NNS into the CL Algorithm

It was shown in the previous section that almost all the operations performed by the CL algorithm are due to the NNS. The CL algorithm performs NNS to determine the winner. Therefore, the execution time of the CL algorithm can be reduced if fast NNS methods, such as PDS or Torres-Huguet, are incorporated with the purpose of winner determination, in substitution to the conventional full-search (exhaustive search).

It is seen in Figure 3(a) how PDS is incorporated to the CL algorithm. In Figure 3(b), one can see Torres-Huguet incorporated to the CL algorithm. It is worth mentioning that $\|\boldsymbol{w}_{index}\|^2$ and $2\sum_{j=1}^{K} w_{index,\,j}$ (in which $index$ is the winner vector index) must be calculated every time the winner is determined, that is, each time a training vector is presented

```
for 1 ≤ n ≤ n_tot
   for 1 ≤ m ≤ M
      distmin = ∞;
      for 1 ≤ i ≤ N
         dist = 0;
         for 1 ≤ j ≤ K
            dist = dist + (x_j − w_ij)^2;
            if(dist > distmin)
               break;
            next j;
         if(dist < distmin)
            index = i;   (w_i selected)
            distmin = dist;
      next i;
      update the winner (w_index);
   next m;
next n;
```

(a) PDS incorporated to CL.

```
Evaluate and store ||w_i||^2 of every
codevector;
Evaluate and store 2 Σ_{j=1}^{K} w_ij of every
codevector;

for 1 ≤ n ≤ n_tot
   for 1 ≤ m ≤ M
      Evaluate x_max;
      distmin = ∞;
      for 1 ≤ i ≤ N
         dist = d̃_1(x, w_i);
         if(dist > distmin) continue;
         dist = d̃(x, w_i);
         if(dist > distmin) continue;
         index = i;   (w_i selected)
         distmin = dist;
      next i;
      update the winner (w_index);
      Evaluate and replace ||w_index||^2;
   Evaluate and replace 2 Σ_{j=1}^{K} w_index,j;
   next m;
next n;
```

(b) Torres-Huguet incorporated to CL.

Figure 3: PDS and Torres-Huguet incorporated to CL.

It is necessary to emphasize that $d(\boldsymbol{x}, \boldsymbol{w}_i)$ and $d_1(\boldsymbol{x}, \boldsymbol{w}_i)$ are modified in algorithm of Figure 3(b). Since the term $\|\boldsymbol{x}\|^2$ depends only on the input vector $\boldsymbol{x}$, it does not need to be computed. The modified $d(\boldsymbol{x}, \boldsymbol{w}_i)$ is $\tilde{d}(\boldsymbol{x}, \boldsymbol{w}_i) =$

$\|\boldsymbol{w}_i\|^2 - 2\sum_{j=1}^{K} x_j w_{ij}$ and the modified $d_1(\boldsymbol{x}, \boldsymbol{w}_i)$ is $\tilde{d}_1(\boldsymbol{x}, \boldsymbol{w}_i) = \|\boldsymbol{w}_i\|^2 - 2x_{max}\sum_{j=1}^{K} w_{ij}$.

6. Results

The following results concern codebook design using a training set corresponding to the Lena image (512×512 pixels). Vector quantization with $K = 16$ (corresponding to blocks of 4×4 pixels) and codebook size $N = 32$, 64, 128, 256 and 512 was considered. The corresponding coding rates are 0.3125 bpp, 0.375 bpp, 0.4375 bpp, 0.5 bpp and 0.5625 bpp, respectively.

Simulations were performed using an AMD Athlon 64, 2.01 GHz with 1 GB RAM. The source codes were implemented using C++ language.

Tables 3, 4, 5 and 6 present the number of multiplications, additions, subtractions and comparisons performed by the CL algorithm, respectively, when FS, PDS and Torres-Huguet are used for the task of winner determination. In all simulations, PDS and Torres-Huguet algorithms are more efficient than FS algorithm. It is observed that the PDS algorithm is more efficient when compared to the Torres-Huguet algorithm with respect to multiplications and additions. However, concerning the number of subtractions and comparisons, Torres-Huguet is more efficient than PDS. As an example, for $N = 512$, Torres-Huguet achieves subtractions savings of 90.39% with respect to FS while the corresponding savings of PDS is 86.17%.

Table 7 indicates the benefits introduced by the PDS and Torres-Huguet algorithms in terms of reducing the number of clock cycles spent for codebook design. It is assumed[6] that each multiplication is performed in nine clock cycles and each one of the remaining considered operations (addition, subtraction and comparison) in two clock cycles. Is is observed in Table 7 that the PDS algorithm overperforms the Torres-Huguet algorithm in terms of number of clock cycles savings. As as example, for $N = 32$, the substitution of FS by PDS leads to an acceleration in terms of clock cycles about a 2.6 factor, while the corresponding substitution by Torres-Huguet leads to an accelaration about a 1.8 factor. It is also observed in Table 7 that the acceleration factor of the CL algorithm, obtained by substituting FS by PDS, increases as the codebook size N increases. Indeed, acceleration factors of 2.6, 3.7, 4.6, 5.5 and 6.2 are obtained respectively for codebook sizes 32, 64, 128, 256 and 512. Table 7 also reveals that the acceleration factor of the CL algorithm, obtained by substituting FS by Torres-Huguet, is in the range 1.8-1.9 for all codebook sizes considered.

It is important to note that the purpose of incorporating PDS and Torres-Huguet to CL algorithm is to reduce the computational complexity of the nearest neighbor search (winner determination). Thus, the codebooks obtained by using PDS, Torres-Huguet and FS for NNS in the CL algorithm are identical.

[6]According to http://www.intel.com/Assets/PDF/manual/248966.pdf

Table 3: Number of multiplications performed by the CL algorithm, considering FS, PDS and Torres-Huguet for NNS. The savings with respect to FS is within parentheses.

N	**Number of multiplications**		
	FS	PDS	Torres-Huguet
32	2.6×10^7	9.0×10^6 (66.69%)	1.7×10^7 (33.10%)
64	5.1×10^7	1.2×10^7 (77.18%)	3.2×10^7 (37.55%)
128	1.0×10^8	1.9×10^7 (81.10%)	6.0×10^7 (40.51%)
256	2.7×10^8	4.3×10^7 (83.99%)	1.6×10^8 (41.27%)
512	6.7×10^8	9.3×10^7 (86.17%)	4.1×10^8 (39.43%)

Table 4: Number of additions performed by the CL algorithm, considering FS, PDS and Torres-Huguet for NNS. The savings with respect to FS is within parentheses.

N	**Number of additions**		
	FS	PDS	Torres-Huguet
32	2.4×10^7	7.0×10^6 (70.99%)	1.5×10^7 (39.45%)
64	4.8×10^7	9.0×10^6 (82.24%)	2.6×10^7 (45.33%)
128	9.5×10^7	1.3×10^7 (86.47%)	4.9×10^7 (48.99%)
256	2.5×10^8	2.6×10^7 (89.56%)	1.3×10^8 (50.16%)
512	6.3×10^8	5.1×10^7 (91.90%)	3.2×10^8 (48.65%)

Table 5: Number of subtractions performed by the CL algorithm, considering FS, PDS and Torres-Huguet for NNS. The savings with respect to FS is within parentheses.

N	**Number of subtractions**		
	FS	PDS	Torres-Huguet
32	2.6×10^7	9.0×10^6 (66.69%)	3.0×10^6 (87.70%)
64	5.1×10^7	1.2×10^7 (77.18%)	6.0×10^6 (89.18%)
128	1.0×10^8	1.9×10^7 (81.10%)	1.0×10^7 (89.98%)
256	2.7×10^8	4.3×10^7 (83.99%)	2.6×10^7 (90.34%)
512	6.7×10^8	9.3×10^7 (86.17%)	6.5×10^7 (90.39%)

Table 6: Number of comparisons performed by the CL algorithm, considering FS, PDS and Torres-Huguet for NNS. The increase with respect to FS is within parentheses.

N	**Number of comparisons**		
	FS	PDS	Torres-Huguet
32	2.0×10^6	8.6×10^6 (460.87%)	2.6×10^6 (70.82%)
64	3.0×10^6	1.3×10^7 (324.28%)	5.0×10^6 (59.52%)
128	6.0×10^6	2.4×10^7 (281.12%)	9.6×10^6 (53.35%)
256	1.7×10^7	5.8×10^7 (245.30%)	2.5×10^7 (50.51%)
512	4.2×10^7	1.3×10^8 (215.71%)	6.3×10^7 (51.55%)

Table 7: Number of clock cycles spent by the CL algorithm for codebook design, considering FS, PDS and Torres-Huguet for NNS.

N	**Number of clock cycles**		
	FS	PDS	Torres-Huguet
32	3.4×10^8	1.3×10^8	1.9×10^8
64	6.6×10^8	1.8×10^8	3.6×10^8
128	1.3×10^9	2.8×10^8	6.8×10^8
256	3.5×10^9	6.4×10^8	1.8×10^9
512	8.7×10^9	1.4×10^9	4.6×10^9

7. Conclusion

This paper showed, by means of analytical expressions, that most of the operations performed by the competitive learning (CL) algorithm applied to vector quantization (VQ) codebook design are due to the nearest neighbor search (NNS). With the purpose of reducing the processing time spent by the CL algorithm, two methods of fast NNS were considered: the partial distance search (PDS) algorithm and Torres-Huguet algorithm.

Results regarding codebook design for image VQ have shown that the PDS as well as the Torres-Huguet algorithm lead to a reduction in the number of operations (multiplications, additions and subtractions) performed by the CL algorithm. Results obtained for image VQ showed that simple modifications performed to accommodate PDS in the CL algorithm lead to consirable savings in number of clock cycles. As an example, for a codebook size 512, CL with PDS is about six times faster than its conventional version (CL with full search NNS).

Acknoledgment

The authors would like to thank Eduardo Lundgren for his valuable comments and the anonymous referees for the relevant suggestions and concerns.

References

[1] M. Antonini, M. Barlaud, P. Mathieu, I. Daubechies, Image coding using wavelet transform. *IEEE Trans. Image Process.*, **1**, No. 2 (1992), 205–220.

[2] C.-D. Bei, R. M. Gray, An improvement of the minimum distortion encoding algorithm for vector quantization, *IEEE Trans. Commun.*, **33**, No. 10 (1985), 1132–1133.

[3] O.T.-C. Chen, B.J. Sheu, W.-C. Fang, Image compression using self-organization networks, *IEEE Trans. Circuits Syst. Video Technol.*, **4**, No. 5 (1994), 480–489.

[4] P.C. Cosman, R.M. Gray, M. Vetterli, Vector quantization of image subbands: A survey, *IEEE Trans. Image Process.*, **5**, No. 2 (1996), 202–225.

[5] A. Gersho, R.M. Gray, "Vector Quantization and Signal Compression", Kluwer Academic Publishers, Boston, MA, 1992.

[6] R.M. Gray, Vector quantization, *IEEE ASSP Magazine*, (1984), 4–29.

[7] T. Kohonen, The self-organizing map, *Proceedings of the IEEE*, **78**, No. 9 (1990), 1464–1480.

[8] A.K. Krishnamurthy, S.C. Ahalt, D.E. Melton, P. Chen, Neural networks for vector quantization of speech and images, *IEEE J. Sel. Areas Commun.*, **8**, No. 8 (1990), 1449–1457.

[9] Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantizer design, *IEEE Trans. Commun.*, **28**, No. 1 (1980), 84–95.

[10] F. Madeiro, W.T.A. Lopes, B.G. Aguiar Neto, M.S. Alencar, Complexidade computacional de um algoritmo competitivo aplicado ao projeto de quantizadores vetoriais, *Learning and Nonlinear Models*, **1**, No. 3 (2004), 172–186.

[11] K.K. Paliwal, B.S. Atal, Efficient vector quantization of LPC parameters at 24 bits/frame, *IEEE Trans. Speech Audio Process.*, **1**, No. 1, (1993), 3–14.

[12] L. Torres, J. Huguet, An improvement on codebook search for vector quantization, *IEEE Trans. Commun.*, **42**, No. 2/3/4 (1994), 208–210.