Applied Extended Associative Memories to High-Speed Search Algorithm for Image Quantization

Enrique Guzmán Ramírez[1], Miguel A. Ramírez[2] and Oleksiy Pogrebnyak[3]
[1]Electronic and Computer Science Dept., Universidad Tecnológica de la Mixteca
[2]Informatic Dept., Universidad de la Sierra Juárez
[3]Computer Science Dept., Centro de Investigación en Computación – IPN
México

1. Introduction

Vector quantization (VQ) has been used as an efficient and popular method in lossy image and speech compression (Gray, 1984; Nasrabadi & King, 1988; Gersho & Gray, 1992). In these areas, VQ is a technique that can produce results very close to the theoretical limits. The most widely used and simplest technique for designing vector quantizers is the LBG algorithm by Y. Linde et al. (1980). It is an iterative descent algorithm which monotonically decreases the distortion function towards a local minimum. Sometimes, it is also referred to as generalized Lloyd algorithm (GLA), since it is a vector generalization of a clustering algorithm due to Lloyd (1982).

New algorithms for VQ based on associative memories or artificial neuronal networks (ANNs) have arisen as an alternative to traditional methods. Within this approach, many VQ algorithms have been proposed. We mention some of them.

The self-organizing map (SOM) ANN, developed by Prof. Teuvo Kohonen in the early 1980s (Kohonen, 1981; Kohonen 1982), has been used with a great deal of success in creating new schemes for VQ. The SOM is a competitive-learning network. C. Amerijckx et al. (1998) proposed a lossy compression scheme for digital still images using Kohonen's neural network algorithm. They applied the SOM at both quantification and codification stages of the image compressor. At the quantification stage, the SOM algorithm creates a correspondence between the input space of stimuli, and the output space constituted of the codebook elements (codewords, or neurons) derived using the Euclidean distance. After learning the network, these codebook elements approximate the vectors in the input space in the best possible way. At the entropy coder stage, a differential entropy coder uses the topology-preserving property of the SOMs resulted from the learning process and the hypothesis that the consecutive blocks in the image are often similar. In (Amerijckx et al., 2003), the same authors proposed an image compression scheme for lossless compression using SOMs and the same principles.

Eyal Yair et al. (1992) provide a convergence analysis for the Kohonen Learning Algorithm (KLA) with respect to VQ optimality criteria and introduce a stochastic relaxation technique which produces the global minimum but is computationally expensive. By incorporating the

principles of the stochastic approach into the KLA, a new deterministic VQ design algorithm, called the soft competition scheme (SCS), is introduced. The new algorithm is an on-line method in which the codeword update is governed by a "soft" competition between the codevector, which are updated simultaneously for each presentation of a training vector.
A new VQ scheme based on competitive learning neural network and LBG vector quantization was presented by Basil and Jiang (1999). The algorithm integrates advantages presented in both LBG vector quantization and competitive learning neural networks to achieve an improved performance in comparison with their existing forms when it is applied to image compression.
Chin-Chuan Han et al. (2006; 2007) proposed an hybrid approach for VQ for obtaining the better codebook. It is modified and improved based on the centroid neural network adaptive resonance theory (CNN-ART) and the enhanced LBG (Linde-Buzo-Gray) approaches. Three modules, a neuronal net (NN) based clustering, a mean shift (MS) based refinement, and a principal component analysis (PCA) based seed assignment, are repeatedly utilized. Basically, the seed assignment module generates a new initial codebook to replace the low utilized codewords during the iteration. The NN-based clustering module clusters the training vectors using a competitive learning approach. The clustered results are refined using the mean shift operation.
A fuzzy Learning Vector Quantization (LVQ) which is based on the fuzzification of LVQ was proposed by Yong-Soo and Sung-Ihl (2007). The proposed fuzzy LVQ uses the different learning rate depending on whether classification is correct or not. When the classification is correct, it uses the combination of a function of the distance between the input vector and the prototypes of classes and a function of the number of iteration as the fuzzy learning rate. On the other hand, when the classification is not correct, it uses the combination of the fuzzy membership value and a function of the number of iteration as the fuzzy learning rate. The proposed FLVQ (fuzzy LVQ) is integrated into the supervised IAFC (Integrated Adaptive Fuzzy Clustering) neural network 5. The iris data set was used to compare the performance of the supervised IAFC neural network 5 with those of LVQ algorithm and back propagation neural network.
VQ is a process in which data to be encoded are divided into small blocks, forming vectors, which are then sequentially encoded vector by vector. The VQ process is divided in two phases: codebook design and encoding phases.
In the codebook design phase the idea is to identify a set, named codebook, of possible vectors (codewords) which are representative of the information to be encoded. The encoding phase searches for codeword with the closest match for every given input vector. The final encoding is then simply a process of sequentially listing the codeword indexes which were deemed to most closely match the vectors making up the original data.
One of the most serious problems for VQ, especially for high dimensional vectors, is the high computational complexity of searching for the closets codeword in the codebook design and encoding phases. Although quantizing high dimensional vectors rather than low dimensional vectors result in a better performance, the computation time needed for vector quantization grows exponentially with the vector dimension. This makes high dimensional vectors unsuitable for vector quantization. Thus, Given a data set (codebook) of m codewords, $\{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_i, ..., \mathbf{c}_m\}$, the problem is to find the i element that is closest to a input vector, $\mathbf{x}$, where $\mathbf{x}, \mathbf{c}_i \in \Re^d \forall i$.

In this study, to overcome the said problem an efficient high-speed search algorithm for image quantization based on Extended Associative Memories (EAM) is proposed. This new algorithm significantly reduces the computation needed without sacrificing performance.
A preliminary version of this work first appeared in (Guzmán et al., 2008). The present work is a widespread version of that one. The new results that this work includes are: 1) A mathematical analysis of the influence that each operator of our proposal has in the codebook design and encoding phase performance. 2) The results of the proposed algorithm when additional operators, **pmed** and **sum**, are used. 3) A complexity analysis of the proposed algorithm for additional operators. 4) The results that let us evaluate the influence that the proposed VQ algorithm has in the image compression performance.
The remaining sections of this work are organized as follows. In next Section, a brief theoretical background of extended associative memories is given. In Section 3 we describe our proposal, the high-speed search algorithm for image quantization based on EAM; furthermore, this sección include a complexity analysis of the proposed algorithm. Numerical simulation results obtained for the conventional image quantization techniques (LBG algorithm) and the proposed algorithm, when it uses **prom**, **med** and **pmed** operators, are provided and discussed in Section 4. Finally, Section 5 contains the conclusions of this study.

2. Extended associative memories model

An associative memory designed for pattern classification is an element whose fundamental purpose is to establish a relation of an input pattern $\mathbf{x} = [x_i]_n$ with the index i of a class $\mathbf{c}_i$ (see Fig. 1).

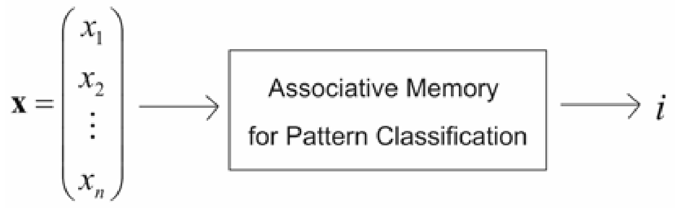

$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$
Associative Memory for Pattern Classification
i

Fig. 1. Associative memory scheme for pattern classification.

Let $\{(\mathbf{x}^1, c_1), (\mathbf{x}^2, c_2), \ldots, (\mathbf{x}^N, c_N)\}$ be N couples of a pattern and its corresponding class index forming the fundamental set of couples composed by patterns and their corresponding class indices:

$$\{(\mathbf{x}^\mu, c_\mu); \mu = 1,2,\ldots,N\} \tag{1}$$

where $\mathbf{x}^\mu \in \Re^n$ and $c_\mu = 1,2,\ldots,N$. The associative memory is represented by a matrix generated from the fundamental set of couples and denoted by $\mathbf{M}$.
H. Sossa et al. (2004) proposed an associative memory model for the classification of real-valued patterns. This model, named Extended Associative Memory (EAM), is an extension of the Lernmatrix model proposed by K. Steinbuch (1961). The EAM is based on the general concept of the associative memory learning function and presents a high performance in pattern classification of real value data by its components and with altered patterns (Vázquez, 2005; Barron, 2006).

In the EAM, being an associative memory utilized for pattern classification, all the synaptic weights that belong to output *i* are accommodated at the *i*-th row of a matrix $\mathbf{M}$. The final value of this row is a function ϕ of all the patterns belonging to class *i*. The function ϕ acts as a generalizing learning mechanism that reflects the flexibility of the memory (Sossa et al., 2004).

2.1 Training stage of the EAM

The training stage of the EAM consists of evaluating function ϕ for each class. Then, the matrix $\mathbf{M}$ can be structured as:

$$\mathbf{M} = \begin{bmatrix} \boldsymbol{\varphi}_1 & \cdots & \boldsymbol{\varphi}_N \end{bmatrix}^T \tag{2}$$

where $\boldsymbol{\varphi}_i$ is the evaluation of ϕ for all patterns of *i*-th class, $i = 1,2,\ldots,N$. The function ϕ can be evaluated in various manners. The arithmetical average operator (**prom**) is frequently used in the signals and images treatment. In (Sossa et al., 2004), the authors studied the performance of this operator and the median operator (**med**) to evaluate the function ϕ. Furthermore, in (Vázquez, 2005) the use of the average point operator (**pmed**) and sum operator (**sum**) were proposed to evaluate the function ϕ.

The goal of the training stage is to establish a relation between an input pattern $\mathbf{x} = [x_i]_n$, and the index *i* of a class c_i.

Considering that each class is composed for *q* patterns $\mathbf{x} = [x_i]_n$, and $\boldsymbol{\varphi}_i = [\phi_{i,1},\ldots,\phi_{i,n}]$. Then, the training stage of the EAM, when the **prom** operator is used to evaluate the function $\boldsymbol{\varphi}_i$, is defined as:

$$\phi_{i,j} = \frac{1}{q}\sum_{l=1}^{q} x_{j,l}, \quad j = 1,\ldots,n \tag{3}$$

The training stage of the EAM, when the **med** operator is used to evaluate the function $\boldsymbol{\varphi}_i$, is defined as:

$$\phi_{i,j} = \operatorname*{med}_{l=1}^{q} x_{j,l}, \quad j = 1,\ldots,n \tag{4}$$

When the **pmed** operator is used to evaluate the function $\boldsymbol{\varphi}_i$, the training stage of the EAM is defined as:

$$\phi_{i,j} = \frac{\gamma_{i,j} + \lambda_{i,j}}{2}, \quad j = 1,\ldots,n \tag{5}$$

When the **sum** operator is used to evaluate the function $\boldsymbol{\varphi}_i$, the training stage of the EAM is defined as:

$$\phi_{i,j} = \gamma_{i,j} + \lambda_{i,j} \tag{6}$$

where γ_{ij} and λ_{ij} represent the maximum and minimum vectors of the class *i* respectively. These vectors are defined as

$$\gamma_{i,j} = \overset{q}{\underset{l=1}{\vee}} \left(x_j^{i,l} \right) \tag{7}$$

$$\lambda_{i,j} = \overset{q}{\underset{l=1}{\wedge}} \left(x_j^{i,l} \right) \tag{8}$$

The operators max ($\vee$) and min ($\wedge$) perform the morphological operations on the patterns belongs to each class.
The associative memory, $\mathbf{M}$, is obtained after evaluating all functions $\boldsymbol{\varphi}_i$. In the case when N classes exist and the vectors to classify are n-dimensional, the resultant memory $\mathbf{M} = \left[m_{ij} \right]_{N \times n}$ is denoted as:

$$\mathbf{M} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \cdots & \phi_{1,n} \\ \phi_{2,1} & \phi_{2,2} & \cdots & \phi_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{N,1} & \phi_{N,2} & \cdots & \phi_{N,n} \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{N,1} & m_{N,2} & \cdots & m_{N,n} \end{bmatrix} \tag{9}$$

2.2 Classification stage of the EAM

The goal of the classification stage is the generation of the class index to which an input pattern belongs.

The pattern classification by EAM is done when a pattern $\mathbf{x}^{\mu} \in \Re^{n}$ is presented to the memory $\mathbf{M}$ generated at the training stage. The EAM possess the feature of classifying a pattern not necessarily one of those already used to build the memory $\mathbf{M}$.
When the **prom** or **pmed** operators are used, the class to which $\mathbf{x}$ belongs is given by

$$i = \underset{l}{\mathbf{arg}} \left[\overset{N}{\underset{l=1}{\wedge}} \overset{n}{\underset{j=1}{\vee}} \left| m_{lj} - x_j \right| \right] \tag{10}$$

In this case, the operators $\vee \equiv max$ and $\wedge \equiv min$ perform morphological operations on the difference of the absolute values of the elements m_{ij} of $\mathbf{M}$ and the component x_j of the pattern $\mathbf{x}$ to be classified.
When the EAM uses the **med** operator, the class to which $\mathbf{x}$ belongs is given by

$$i = \underset{l}{\mathbf{arg}} \left[\overset{N}{\underset{l=1}{\wedge}} \left| \underset{j=1}{\overset{n}{\operatorname{med}}} m_{lj} - \underset{j=1}{\overset{n}{\operatorname{med}}} x_j \right| \right] \tag{11}$$

In this case, the operator $\wedge \equiv min$ performs a morphological erosion over the absolute difference of the median of the elements m_{ij} of $\mathbf{M}$ and the median of the component x_j of the pattern $\mathbf{x}$ to be classified.
When the **sum** operator is used, the recovery matrix, $\mathbf{r} = \left[r_{ij} \right]_{N \times n}$, must be generated. The components of the recovery matrix can be computed of two forms:

$$r_{ij} = x_j + \gamma_{l,j} \tag{12}$$

$$r_{ij} = x_j + \lambda_{l,j} \tag{13}$$

Then, the class to which $\mathbf{x}$ belongs is given by

$$i = \mathbf{arg}_l \left[\overset{N}{\underset{l=1}{\wedge}} \overset{n}{\underset{j=1}{\vee}} \left| m_{lj} - r_{lj} \right| \right] \tag{14}$$

In this case, the operators $\vee \equiv max$ and $\wedge \equiv min$ perform morphological operations on the difference of the absolute values of the elements m_{ij} of $\mathbf{M}$ and the component r_{ij} of the recovery matrix.
The **Theorem 1** and **Corollaries 1-5** from (Sosa et al., 2004) govern the conditions that must be satisfied to obtain a perfect pattern classification; either the pattern may be from the fundamental set of couples or be an altered version of the pattern. Here we reproduce them.
Theorem 1. Let $d_i = \underset{\mathbf{x} \text{ class } i}{\vee} d(\mathbf{x}, \boldsymbol{\varphi}_i)$ and $\mathbf{R}_i = \{\mathbf{x} : d(\mathbf{x}, \boldsymbol{\varphi}_i) \le d_i\}$ be hyper cubes centred at $\boldsymbol{\varphi}_i$ and semi-sides d_i, $i = 1, ..., N$. If $d(\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_j) > 2\,\mathbf{max}\{d_i, d_j\}$ then:

i. $R_i \cap R_j = \varnothing,\ 1 \le i, j \le N,\ i \ne j.$

ii. $\mathbf{x} \in R_i$ implies $d(\mathbf{x}, \boldsymbol{\varphi}_i) \le d(\mathbf{x}, \boldsymbol{\varphi}_j)$.

iii. $\mathbf{x} \in R_j$ implies $d(\mathbf{x}, \boldsymbol{\varphi}_j) \le d(\mathbf{x}, \boldsymbol{\varphi}_i)$.

Corollary 1. If the conditions of Theorem 1 hold for all $1 \le i, j \le m,\ i \ne j$ then:

i. $R_i \cap R_j = \varnothing,\ 1 \le i, j \le N,\ i \ne j.$

ii. If $\mathbf{x} \in R_i$ then $d(\mathbf{x}, \boldsymbol{\varphi}_i) \le d(\mathbf{x}, \boldsymbol{\varphi}_j),\ 1 \le i, j \le N,\ i \ne j.$

Corollary 2. Perfect classification of the fundamental set of patterns is immediate due to any fundamental pattern $\mathbf{x}$, by constructions, belongs to its corresponding region R_i and if the conditions of Theorem 1 hold, then $d(\mathbf{x}, \boldsymbol{\varphi}_i) \le d(\mathbf{x}, \boldsymbol{\varphi}_j),\ 1 \le j \le N,\ i \ne j.$ Thus the memory assigns the fundamental pattern $\mathbf{x}$ to class *i* as is due.
Corollary 3. If $\tilde{\mathbf{x}}$ is an altered version of fundamental pattern $\mathbf{x} \in R_i$, $\tilde{\mathbf{x}}$ is correctly classified if $\tilde{\mathbf{x}} \in R_i$.
Corollary 4. The attraction basin of *i*-th class is at least R_i and the convergence of correct classification happens in one step.
Corollary 5. Let $\mathbf{x}$ be a pattern to be classified. If $\mathbf{x} \notin R_i$ and $\mathbf{x} \notin R_j$ then it must be classified at the class for which:

i. $i = \arg_l \left[\overset{N}{\underset{l=1}{\wedge}} \overset{n}{\underset{j=1}{\vee}} \left| m_{lj} - x_j \right| \right]$ **prom** and **pmed** operators.

ii. $i = \arg_l \left[\overset{N}{\underset{l=1}{\wedge}} \left| \overset{n}{\underset{j=1}{\text{med}}}\, m_{lj} - \overset{n}{\underset{j=1}{\text{med}}}\, x_j \right| \right]$ **med** operator.

iii. $i = \arg_l \left[\overset{N}{\underset{l=1}{\wedge}} \overset{n}{\underset{j=1}{\vee}} \left| m_{lj} - r_{lj} \right| \right]$ sum operator.

3. Image quantization based on extended associative memories

In this section the image quantization algorithm based on EAM is described. Our proposal uses the EAM in both codebook generation and in the encoding phase. In codebook generation, applying the learning stage of the EAM on a codebook generated by the LBG algorithm a new codebook, named EAM-codebook, is obtained. In encoding phase, we propose a High-Speed Search Algorithm Applied to Image Quantization based on EAM; the VQ process is performed by means of the recalling stage of EAM using as associative memory the EAM-codebook. This process generates a set of the class indices to which each input vector belongs.

3.1 Basic notations and preliminary definitions

Vector quantization can be viewed as a mapping **Q**, from a *n*-dimensional vector space R^n into a finite subset ***C*** of R^n

$$Q : R^n \rightarrow C \tag{15}$$

where $C = \{\mathbf{y}^i : i = 1,\ 2, \ldots,\ N\}$ is the set of reconstruction vectors and *N* is the number of vectors in *C*. Thus,

i. $\bigcup_{i=1}^{N} \mathbf{y}^i = C,$

ii. $\mathbf{y}^i \bigcap \mathbf{y}^j = \emptyset,\ \forall i, j, i \neq j.$

Each $\mathbf{y}^i$, in *C* is called a *codeword* and *C* is called the *codebook* for the vector quantizer. For every source vector $\mathbf{x}$, a codeword $\mathbf{y}^i$, in *C* is selected as the representation for $\mathbf{x}$. This process is called the *quantization phase* (or the *codebook search phase)* of the vector quantizer, denoted by $Q(\mathbf{x}) = \mathbf{y}^\mathbf{i}$. Then, the codeword $\mathbf{y}^i$, is represented by some symbols (normally the address of the codeword in the codebook) and transmitted through the channel. This process is called the *encoding phase* of the vector quantizer.

On the other side of the channel, the received symbols are used to select the codewords from the codebook to reproduce the source signals. This process is called the *decoding phase* of the vector quantizer. The average number of bits required to represent the symbols in the encoding phase is the *rate* of the quantizer, and the average quantization error between input source signals and their reproduction codewords is the *distortion* of the vector quantizer. Increasing the number of codewords in the codebook can decrease the distortion of a vector quantizer and, normally, will increase the rate also. One major concern for vector quantizer design is the trade-off between distortion and rate.

On the other hand, both codebook generation and encoding phases of the proposed VQ algorithm make use of individual blocks of the image, which must be converted to vectors. Let the image be represented by a matrix, $\mathbf{A} = [a_{ij}]_{hi \times wi}$, where *hi* is the image height and *wi* is the image width; and *a* represents the *ij*-th pixel value: $a \in \{0,1,2,\ldots,2^L - 1\}$, where L is the number of bits necessary to represent the value of a pixel.

Now, we define the *image block* and *image vector* terms.

Definition 1, image block (**ib**). Let $\mathbf{A} = [a_{ij}]$ be a $hi \times wi$ matrix representing an image, and let $\mathbf{ib} = [ib_{ij}]$ be a $d \times d$ matrix. The **ib** matrix is defined as a image block of the **A** matrix if the **ib** matrix is a subgroup of the **A** matrix such that

$$ib_{ij} = a_{\delta_i \tau_j} \tag{16}$$

where $i, j = 1,2,...,d$, $\delta = 1, d+1, 2d+1, ..., h-d+1$, $\tau = 1, d+1, 2d+1, ..., w-d+1$ and $a_{\delta_i \tau_j}$ represents the value of the pixel determined by the coordinates (*δ*+*i*, *τ*+*j*), where (*δ*, *τ*) and (*δ*+*d*, *τ*+*d*) are the beginning and the end of the image block, respectively.

Definition 2, image vector (**iv**). Let $\mathbf{ib} = [ib_{ij}]$ be an image sub-block and let $\mathbf{iv} = [iv_i]$ be a vector of size *d*. The *i*-th row of the **ib** matrix is said to be an image vector **iv** such that

$$iv_i = [ib_{i1}, ib_{i2}, ..., ib_{id}] \tag{17}$$

where $i = 1,2,3,...,d$. From each imageblock, *d* image vectors can be obtained:

$$\mathbf{iv}^{\mu} = [ib_{\mu 1}, ib_{\mu 2}, ..., ib_{\mu d}] \tag{18}$$

where $\mu = 1,2,3,...,d$.

A image block can be represented as a vector $\mathbf{x} = [x_j]_n$; thus, **ib** can be defined as a set of *d* row image vectors $\mathbf{iv} = [iv_j]_d$: $\{\mathbf{iv}^1, \mathbf{iv}^2, ..., \mathbf{iv}^d\}$, where, $iv_j^i = ib_{ij} | i, j = 1,2,...,d$; then

$$\mathbf{x} = \{\mathbf{iv}^i : i = 1,2,..d\} \tag{19}$$

where, $x_p = iv_j^i | i, j = 1,2,...,d$, $p = 1,2,...,n$, $n = d \times d$.

Finally, the new image representation is defined as

$$\mathbf{A} = X = \{\mathbf{x}^i : i = 1,2,...,M\} \tag{20}$$

where, $M = (hi/d)(wi/d)$.

In both EAM-codebook generation and encoding phases the new image representation is used.

3.2 Codebook generation

In this phase, an associative network is generated applying the learning stage of the EAM between a codebook generated by the LBG algorithm and a training set. This associative network is named EAM-codebook and establishes a relation between training set and the LBG codebook.

The codebook generation based on the LBG algorithm and the EAM is fundamental in obtaining a fast search algorithm for image VQ.

The EAM-codebook is computed in three steps:

Step 1. *LBG codebook generation*. This step applied the LBG algorithm on the image blocks (training set), $X = \{\mathbf{x}^i : i = 1, 2,..., M\}$, to generate an initial codebook: $C = \mathbf{y}^i : i = 1, 2,..., N$. This codebook is formed by a set of *n*-dimensional vectors $\mathbf{y} = [y_j]$, named *codeword*.

The Fig 2 shows the scheme of the LBG algorithm, where, $i = 1,...,n$, $\mu = 1,...,N$, $s_1, s_2, ..., s_N$ can have different value and $s_1 + s_2 + ... + s_N = M$.

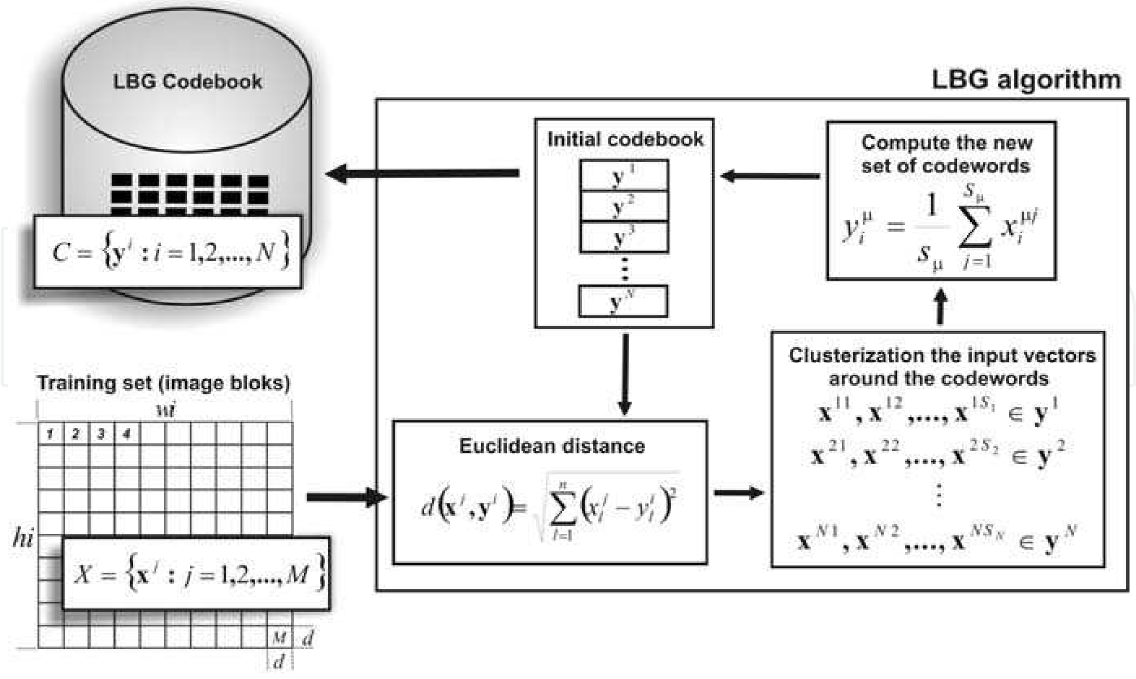

LBG Codebook
$C = \{\mathbf{y}^i : i = 1,2,...,N\}$
Training set (image bloks)
wi
hi
$X = \{\mathbf{x}^j : j = 1,2,...,M\}$
M d
d
LBG algorithm
Initial codebook
$\mathbf{y}^1$ $\mathbf{y}^2$ $\mathbf{y}^3$ $\mathbf{y}^N$
Compute the new set of codewords
$y_i^\mu = \frac{1}{s_\mu}\sum_{j=1}^{S_\mu} x_i^{\mu j}$
Clusterization the input vectors around the codewords
$\mathbf{x}^{11}, \mathbf{x}^{12}, ..., \mathbf{x}^{1S_1} \in \mathbf{y}^1$
$\mathbf{x}^{21}, \mathbf{x}^{22}, ..., \mathbf{x}^{2S_2} \in \mathbf{y}^2$
$\mathbf{x}^{N1}, \mathbf{x}^{N2}, ..., \mathbf{x}^{NS_N} \in \mathbf{y}^N$
Euclidean distance
$d(\mathbf{x}^j, \mathbf{y}^i) = \sqrt{\sum_{l=1}^{n}(x_l^j - y_l^i)^2}$

Fig. 2. LBG codebook generation.

Step 2. *Definition of the fundamental set of couples*. This step uses the codebook generated in the previous step and the training set.

This step designs a **Q** mapping and assigns an index i to each n-dimensional input pattern $\mathbf{x} = (x_1, x_2, ..., x_n)$, with $\mathbf{Q}(\mathbf{x}) = \mathbf{y}^i = (y_{i1}, y_{i2}, ..., y_{in})$. The **Q** mapping is designed to map $\mathbf{x}$ to $\mathbf{y}^i$ with $\mathbf{y}^i$ satisfying the following condition:

$$d(\mathbf{x}, \mathbf{y}^i) = \min_j d(\mathbf{x}, \mathbf{y}^j), \text{for } j = 1,2,3,...,N \tag{21}$$

where $d(\mathbf{x}, \mathbf{y}^j)$ is the distortion of representing the input pattern $\mathbf{x}$ by the codeword $\mathbf{y}^j$, measured by Euclidean distance.

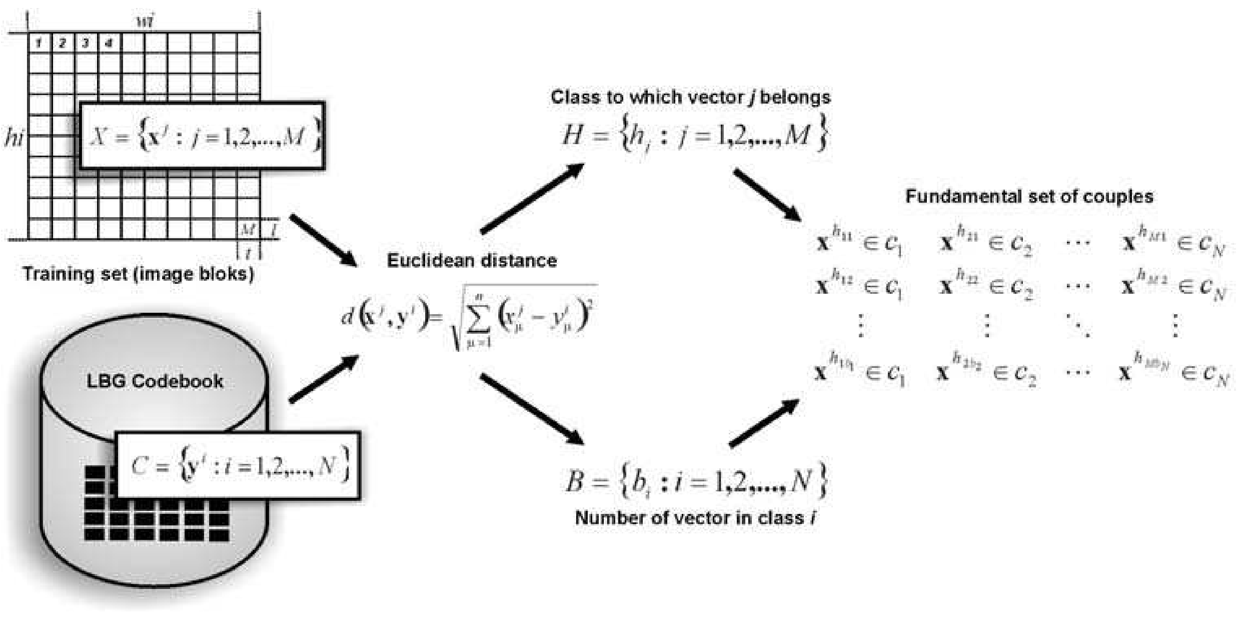
Each codeword $\mathbf{y}^i$ represents a class (existing N classes), then a set of M indices that indicates to which class, $\mathbf{y}^i$, each input pattern **x** belongs is generated. Let us to denote this set of indices as $H = \{h_i : i = 1, 2,..., M\}$. This step also generates a set of N indices that indicates the number of input patterns that integrate to each class. This set is denoted as $B = \{b_i : i = 1, 2,..., N\}$.

Based on the sets H and B and considering that the set of input patterns is integrated from M image blocks, $X = \{\mathbf{x}^i : i = 1, 2,..., M\}$, the fundamental set of couples is formed

$$\begin{matrix} \mathbf{x}^{h_{11}} \in c_1 & \mathbf{x}^{h_{21}} \in c_2 & \cdots & \mathbf{x}^{h_{M1}} \in c_N \\ \mathbf{x}^{h_{12}} \in c_1 & \mathbf{x}^{h_{22}} \in c_2 & \cdots & \mathbf{x}^{h_{M2}} \in c_N \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}^{h_{1b_1}} \in c_1 & \mathbf{x}^{h_{2b_2}} \in c_2 & \cdots & \mathbf{x}^{h_{Mb_N}} \in c_N \end{matrix} \tag{22}$$

where, $b_1, b_2, ..., b_N$ can have different value and $b_1 + b_2 + ... + b_N = M$.

The Fig. 3 shows the definition of the fundamental set of couples.



mi
hi
$X = \{\mathbf{x}^j : j = 1,2,...,M\}$
Training set (image bloks)
LBG Codebook
$C = \{\mathbf{y}^i : i = 1,2,...,N\}$
Euclidean distance
$d(\mathbf{x}^j,\mathbf{y}^i) = \sqrt{\sum_{\mu=1}^{n}(x_\mu^j - y_\mu^i)^2}$
Class to which vector j belongs
$H = \{h_j : j = 1,2,...,M\}$
$B = \{b_i : i = 1,2,...,N\}$
Number of vector in class i
Fundamental set of couples

Fig. 3. Definition of the fundamental set of couples.

Step 3. *Generation of a new codebook based on EAM.* The goal of the third step is to generate the EAM-codebook.

Finding an optimal solution of the feature vector quantization problem necessitates a training process which involves learning the probability distribution of the input data. For this purpose, the training stage of the EAM is applied on the fundamental set of couples, denoted by the equation (22).

Considering that each class can group $b_1,b_2,...,b_N$ input vectors, $\mathbf{x} = [x_i]_n$, the generalizing learning mechanism of the EAM is defined by the expressions (3), (4), (5) and (6) for the operators **prom**, **med**, **pmed** and **sum** respectively

$$m_{uv} = \frac{1}{q}\sum_{g=1}^{q} x_v^g$$

$$m_{uv} = \operatorname*{med}_{g=1}^{q} x_v^g$$

$$m_{uv} = \frac{\bigvee_{g=1}^{q} x_v^{u,g} + \bigwedge_{g=1}^{q} x_v^{u,g}}{2}$$

$$m_{uv} = \bigvee_{g=1}^{q} x_v^{u,g} + \bigwedge_{g=1}^{q} x_v^{u,g}$$

where q can take the value of $b_1, b_2, ..., b_N$, $v = 1, ..., n$, and $u = 1, 2, ..., N$.
The result of this step is an associative network that establishes a relation between the q input vectors and the class to which they belong.
Since N classes exist and the input vectors that integrate them are n-dimensional, then the associative network is represented by a matrix $\mathbf{M} = [m_{uv}]_{N\times n}$, which is the EAM-codebook:

$$\text{EAM-codebook} = \mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{N1} & m_{N2} & \cdots & m_{Nn} \end{bmatrix} = [m_{uv}]_{N\times n} \tag{23}$$

The column $(m_{u,1}, m_{u,2}, ..., m_{u,n})$ of $\mathbf{M}$ represents the centroid of the class u, $u = 1, 2, ..., N$. That is, the synaptic weight, m_{uv}, is a function ϕ of all elements belonging to class u. These synaptic weights determine the behaviour of the net.

3.3 High-speed search algorithm for image quantization based on EAM

In the encoding phase of VQ process each input pattern is replaced by the codeword index that presents the nearest matching based on the similarity criterion between input patterns and codewords. In our proposal, this criterion has been codified in the associative memory (EAM-codebook).
The EAM-codebook generation is based on the EAM training stage. Therefore, using the EAM classification stage, the index class to which each input vector belongs is obtained; at the end of this step the image VQ process is completed.
In a general case, when N classes exist, the input vectors, $\mathbf{x}$, are n-dimensional and the **prom** or **pmed** operator is used to generate the memory $\mathbf{M}$, the index class to which an input vector belongs is determined using the morphologic operators *max* and *min*

$$\text{index class} = \arg_h \left[\bigwedge_{h=1}^{N} \bigvee_{j=1}^{n} \left| m_{hj} - x_j \right| \right]$$

The morphologic operation $\bigvee_{j=1}^{n} \left| m_{hj} - x_j \right| \equiv d(\mathbf{x}, m_h)$ is the metrics that indicates the distance between each row of $\mathbf{M}$ and the feature vector **x**, that is to say, the degree of belonging of the feature vector $\mathbf{x}$ with each one of the N classes; that is

$$d(\mathbf{x}, m_h) = (|m_{h1} - x_1|) \vee (|m_{h2} - x_2|) \vee \ldots \vee (|m_{hn} - x_n|) \tag{24}$$

On the other hand, when the **med** operator is used to generate the memory $\mathbf{M}$, the region to which a feature vector $\mathbf{x}$ belongs is determined using the median and the morphologic operator *min.*

$$\text{index class} = \mathbf{arg}_h \left[\bigwedge_{h=1}^{N} \left| \operatorname*{med}_{j=1}^{n} m_{hj} - \operatorname*{med}_{j=1}^{n} x_j \right| \right]$$

Here, $\left| \operatorname*{med}_{j=1}^{n} m_{hj} - \operatorname*{med}_{j=1}^{n} x_j \right| \equiv d(\mathbf{x}, m_h)$ is a metric that indicates the degree of belonging of the feature vector **x** to each one of the *N* classes.
Considering that $m_{h1}, m_{h2}, ..., m_{hn}$ and $x_1, x_2, ..., x_n$ are sets of finite values in either ascending or descending order, the median has two cases: if *n* is an odd number, then the median is the value defined as $m_{h((n+1)/2)}$ and $x_{(n+1)/2}$; if *n* is an even number, then the median is the value defined as $\left(m_{h(n/2)} + m_{h((n/2)+1)}\right)/2$ and $\left(x_{n/2} + x_{(n/2)+1}\right)/2$, then:

$$
\begin{aligned}
d(\mathbf{x}, m_h) &= \left| m_{h((n+1)/2)} - x_{(n+1)/2} \right| \\
d(\mathbf{x}, m_h) &= \left| \left(m_{h(n/2)} + m_{h((n/2)+1)} \right) \Big/ 2 - \left(x_{n/2} + x_{(n/2)+1} \right) \Big/ 2 \right|
\end{aligned}
\tag{25}
$$

The computation of $d(\mathbf{x}, m_h)$ is simpler when the **prom** operator is used because the values of $\mathbf{x}$ and m_h have not to be ordered, this fact implies the highest processing speed. On the other hand, when the **med** operator is used, the extreme values of the set do not have important effects on the results.
The result of $d(\mathbf{x}, m_h)$ computation for three cases, **prom**, **med** and **pmed** operators, is a column vector where each element indicates the degree of belonging of the input vector, $\mathbf{x}$, with each one of the *N* classes

$$
\begin{bmatrix} d(\mathbf{x}, m_1) \\ d(\mathbf{x}, m_2) \\ \vdots \\ d(\mathbf{x}, m_N) \end{bmatrix}
\tag{26}
$$

Finally, to establish which class belongs to a new input vector, the operator *min* is applied to the vector in the expression (26). The class is indicated by the index of the row of $\mathbf{M}$ that presents the nearest matching with the input vector $\mathbf{x}$.

$$
\text{index class } = \arg_h \left[\bigwedge_{h=1}^{N} d(\mathbf{x}, m_h) \right]
\tag{27}
$$

The VQ process finishes when all the inputs vectors (image blocks) have been replaced by the index class to which it belongs.
In most of the data processing, the proposed algorithm makes use of morphological operations that is, sums and comparisons. Therefore, the proposed algorithm offered a high processing speed. The Fig. 4 shows the structure of the associative memory that implements the high-speed search algorithm.

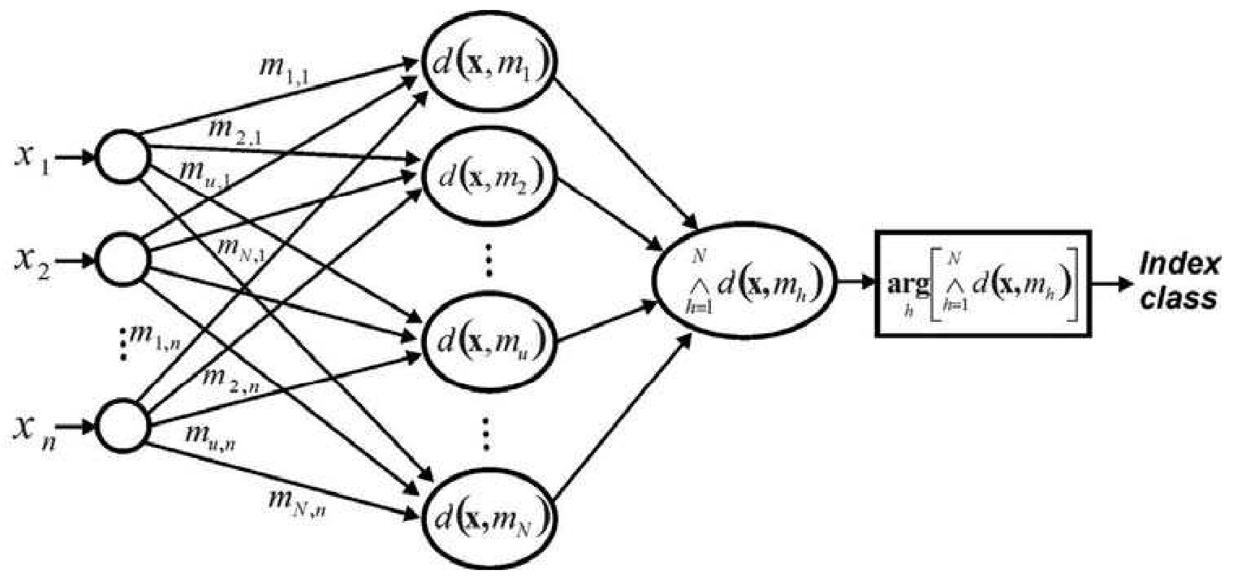


Fig. 4. Structure of the High-speed search algorithm based on EAM.

3.4 Complexity of the high-speed search algorithm

In this subsection, we analyze both the time and space complexity of the proposed algorithm. The algorithm complexity is measured by two parameters: the *time* complexity, or how many steps need the algorithm to solve the problem, and the *space* complexity, or how much memory it requires. To compute these parameters, we will use the pseudocodes of the **prom**, **med** and **pmed** operators that our algorithm uses to quantify a input vector (see Algorithm 1).

3.4.1 Time complexity

In order to measure the algorithm time complexity, we first obtain the run time based on the number of elementary operations (EO) that our proposal realizes to classify a input vector.

For **prom** and **pmed** operators, we consider pseudocode from Algorithm 1(a), thus in the *worst case*, the conditions of lines 9 and 16 will always be true. Therefore, the lines 10, 17 and 18 will be executed in all iterations, and then the internal loops realize the following number of EO:

$$\left(\sum_{j=1}^{n}(9+3)\right)+3=12\left(\sum_{j=1}^{n}1\right)+3=12n+3$$

$$\left(\sum_{l=1}^{N}(5+3)\right)+3=8\left(\sum_{l=1}^{N}1\right)+3=8N+3$$

```
(a)
   // prom and pmed operators
01 | subroutine prom()
02 | variables
03 |    x,k,j,l,aux: integer
04 |    arg[N]='0': integer
05 | begin
06 |    for k←1 to N [operations k=k+1] do
07 |       for j←1 to n [operations j=j+1]) do
08 |          aux=abs(prom_codebook[k][j]-x[j]);
09 |          if (aux>arg[k]) then
10 |             arg[k]=aux;
11 |          end_if
12 |       end_for
13 |    end_for
14 |    aux=max(x);
15 |    for l←1 to N [operations l=l+1] do
16 |       if(arg[l]<aux)
17 |          aux=arg[l];
18 |          index=l;
19 |       end_if
20 |    end_for
21 | end_subroutine
```

```
(b)
   // med operator
01 | subroutine med()
02 | variables
03 |    z,aux,w,aux2,index: integer
04 |    median: float
05 | begin
   // Arranges the vector input elements
06 |    for z←1 to n [operations z=z+1] do
07 |       aux=x[z];
08 |       w=z-1;
09 |       while (w>=0 && x[w]>aux) do
10 |          x[w+1]=x[w];
11 |          w=w-1;
12 |       end_while
13 |       x[w+1]=aux;
14 |    end_for
   // Compute the median
15 |    if (n%2==0) then
16 |       median=(x[n/2]+x[(n/2)-1])/2;
17 |    else
18 |       median=x[n/2];
19 |    end_if
   // Compute an element of M
20 |    aux2=max(x);
21 |    for k←1 to N [operations k=k+1] do
22 |       aux=abs(med_codebook[k]-median);
23 |       if(aux<aux2)
24 |          aux2=aux;
25 |          index=k;
26 |       end_if
27 |    end_for
28 | end_subroutine
```

Algorithm 1. Pseudocodes of the high-speed search algorithm: (a) **prom** and **pmed** operators, (b) **med** operator.

The next loop will repeat $12n+3$ EO at each iteration:

$$\left(\sum_{k=1}^{N}(12n+3)+3\right)+3=\left(\sum_{k=1}^{N}12n+6\right)+3=N(12n+6)+3=12Nn+6N+3$$

Thus, the equation (28) defines the total number of EO that the algorithm realizes.

$$T(n)=(12Nn+6N+3)+(8N+3)+1=12Nn+14N+7 \tag{28}$$

where N is a codebook size and n is a pattern dimension.

Now, for **med** operator, we consider the *worst case* of the pseudocode from Algorithm 1(b), thus, the conditions of lines 15 and 23 will always be true and the "while loop" will be executed z times in each iteration. From Algorithm 1(b), we can distinguish 3 phases: "arranges the vector input elements", "compute the median" and "compute an element of $\mathbf{M}$".

The "arranges the vector input elements" phase realizes the following number of EO:

$$\left(\sum_{z=1}^{n}(7+3)\right)+3=10\left(\sum_{z=1}^{n}1\right)+3=10n+3,\ \text{without "while loop"}$$

$$\frac{n(n+1)}{2}(12) = 6n(n+1) = 6n^2 + 6n, \text{ only the "while loop"}$$

The "compute the median" phase realizes 10 EO and the "compute an element of $\mathbf{M}$" phase realizes the following number of EO:

$$\left(\sum_{k=1}^{N}(7+3)+3\right)+1 = 10\sum_{k=1}^{N}1+4 = 10N+4$$

Finally, the equation (29) defines the total number of EO that the algorithm realizes.

$$T(n) = \left((10n+3)+\left(6n^2+6n\right)\right)+(10)+(10N+4) = 6n^2+16n+10N+17 \tag{29}$$

Also, we analyze the number and type of arithmetical operations used by our algorithm during the VQ process of an image. Then, for **prom** and **pmed** operators, we consider the pseudocode from Algorithm 1(a); furthermore, we know that this process is applied to image of $hi \times wi$ size. Thus, the number of operations that our algorithm, with **prom** or **pmed** operators, needs to quantify the image depends on the image size, the codebook size *N* and the pattern dimension *n*

$$\left(\frac{hi \times wi}{n}\right)\left[Nn(op2)+N(op1)\right] \tag{30}$$

where *op1*=1 comparison and *op2*=1 sum and 1 comparison; $(hi \times wi)/n$ is the number of patterns to quantify.
$Nn(op2)$ is the number and type of arithmetical operations used to perform a morphological dilation over the absolute difference of the EAM-codebook elements and the input pattern components; $N(op1)$ is the number and type of arithmetical operations used to perform a morphological erosion over the result of the previous process.
Now, for **med** operator, we consider pseudocode from Algorithm 1(b). Thus, the number of operations that our algorithm, with **med** operator, needs to quantify the image depends on the image size, the codebook size *N*, the pattern dimension *n* and the computation of the median

$$\left(\frac{hi \times wi}{n}\right)\left[n(op1)+(op2)+N(op3)\right] \tag{31}$$

where *op1* = 2 sums and 2 comparisons, *op2* = 2 sums, 1 comparison and 3 shift, *op2* = 1 sum and 1 comparison.
$n(op1)$ is the number and type of arithmetical operations used to arranges the vector input elements; $(op2)$ is the number and type of arithmetical operations used to compute the median and $N(op3)$ is the number and type of arithmetical operations used to compute an element of $\mathbf{M}$.

3.4.2 Space complexity

The algorithm space complexity is determined by the amount of memory required for its execution. To quantify an image of $hi \times wi$ size, which contains $M = (hi \times wi)/n$ *n*-dimensional patterns, the proposed algorithm implementation for **prom** and **pmed** operators, requires two vectors $\text{arg}[N]$ and $\text{indexes}[M]$, and one matrix $\text{prom\_codebook}[N][n]$. Thus, the number of memory units (*mu*) required for this process is:

$$mu\_\text{arg} + mu\_\text{indexes} + mu\_\text{prom\_codebook} = N{+}M{+}N(n) = M{+}N(n{+}1) \tag{32}$$

When the **med** operator is used, our algoritm only requires two vector $\text{indexes}[M]$ and $\text{med\_codebook}[N]$. Thus, the number of *mu* required is:

$$mu\_\text{indexes} + mu\_\text{med\_codebook} = M{+}N \tag{33}$$

For grayscale image (8 bits/pixel), the variables arg, prom_codebook and med_codebook are declared type *byte*, whereas the variable indexes depends on the codebook size, thus, if $N \le 256$ then indexes is type *byte* and if $N > 256$ then indexes is type *short* (16 bit integer signed numbers).
Then the total number of bytes required by the implementation of the algorithm with **prom** or **pmed** operatos is:

$$\begin{array}{l} M + N(n+1) \ \text{ if } \ N \le 256 \\ 2M + N(n+1) \ \text{ if } \ N > 256 \end{array} \tag{34}$$

and, the total number of bytes required by the implementation of the algorithm with **med** operato is:

$$\begin{array}{l} M + N \ \text{ if } \ N \le 256 \\ 2M + N \ \text{ if } \ N > 256 \end{array} \tag{35}$$

The number of memory units depends on the image size, codebook size and the codeword size chosen for the VQ process.

4. Experimental results

This section presents the experimental results obtained when our proposal is applied to an image quantization process. First, we show the proposed algorithm performance when it is using the **prom**, **med** and **pmed** operators. Then, we compare the proposed algorithm performance with the traditional LBG algorithm. The evaluated parameters are, the distortion generated on the reconstructed image and the influence that the VQ process has in the image compression when diverse codification methods are used. Finally, we analyze the number and type of operations and the amount of memory used by the proposed algorithm and the traditional LBG algorithm. For this purpose, a set of standard test images of size 512×512 pixels and 256 gray levels (Fig. 5) was used in simulations.
In order to measure the performance of both our proposal and LBG algorithm, we used a popular objective performance criterion named peak signal-to-noise ratio (PSNR), which is defined as

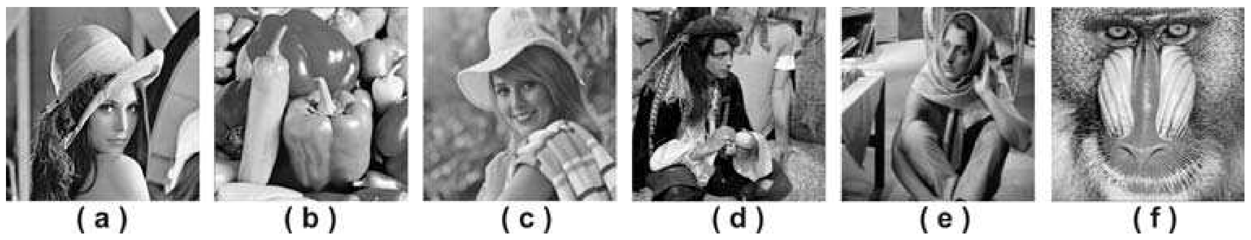


Fig. 5. Set of test images: (a) Lena, (b) Peppers, (c) Elaine, (d) Man, (e) Barbara, (f) Baboon.

$$PSNR = 10\log_{10}\left(\frac{\left(2^n-1\right)^2}{1/_{M}\sum_{i=1}^{M}\left(p_i-\tilde{p}_i\right)^2}\right) \tag{36}$$

where n is the number of bits per pixel, M is the number of pixels in the image, p_i is the i-th pixel in the original image, and $\tilde{p}_i$ is the i-th pixel in the reconstructed image.

The first experiment has the purpose to determine the performance that the algorithm proposed has with each of the operators (**prom**, **med** and **pmed**) when it is applied on the test images. The table 1 includes the distortion that each operator adds to the images during the quantization process when different sizes of codebook are used; the results show that the **prom** operator generates the minor distortion when the algorithm is applied on the test images.

In this experiment a codebook was generated for each test image to determine which image presents the best performance when it is used as training set; the best results were obtained when the image Lena was used to generate the EAM-codebook.

To minimize the distortion generated in the quantification process of images that were not used to obtain the EAM-codebook, it is possible to use an evolutionary method that allows adding information of new images to the book. This aspect is the subject of future work based on paper where the authors proposed the design of an evolutionary codebook using morphological associative memories (Guzmán et al., 2007).

The second experiment has the objective to compare the distortions that both the proposed algorithm (with **prom** operator) and LBG algorithm (the most widely quantization method used) add to the original image. For this purpose, the codebook was generated using the image Lena as the training set and the results were obtained using different sizes of the codebook. Then, VQ was applied to the set of the test images in order to determine the behavior of the algorithms with the patterns that do not belong to the training set. Table 2 shows the results of this experiment.

From Table 2, we can make the following observations: 1) the results obtained show that the proposed method is competitive with the LBG algorithm in the PSNR parameter; 2) the proposed algorithm replaced an input pattern by the index of the codeword that presents the nearest matching, not necessarily one of those already used to built the **M** memory. This property allows our algorithm to quantify efficiently the images that have not been used in the generation of the EAM-codebook.

In the third experiment, the performance of diverse standard coding methods applied to the quantified image with the proposed algorithm was evaluated. These methods included statistical modeling techniques, such as arithmetical, Huffman, range, Burrows Wheeler transformation, PPM, and dictionary techniques, LZ77 and LZP. The purpose of the third experiment is to analyze the proposed algorithm performance in image compression. For this purpose, a coder that includes our algorithm and diverse entropy coding techniques was developed. Table 3 shows the compression results obtained from applying the coder on test images expressed as bit per pixel (bpp).

Operator	Images	Proposed algorithm			
		Codebook size			
		64	128	256	512
		PSNR	PSNR	PSNR	PSNR
prom	Lena	26.4166	27.4702	28.3959	29.2524
	Peppers	25.2016	26.0691	26.5467	27.0489
	Elaine	28.3715	29.1042	29.6899	30.1218
	Man	23.2950	23.9506	24.5434	25.0942
	Barbara	21.3870	21.7271	22.0764	22.4347
	Baboon	18.1937	18.5259	18.8603	19.1508
med	Lena	18.2766	18.4610	18.5554	19.6281
	Peppers	18.1225	18.2008	17.9674	18.6897
	Elaine	18.8188	19.1405	19.1743	20.6147
	Man	17.5697	17.7426	17.7681	18.3887
	Barbara	17.0985	17.1972	17.1063	17.8108
	Baboon	14.5220	14.7361	14.5346	15.0243
pmed	Lena	24.5133	26.4137	27.6557	28.7976
	Peppers	23.5998	25.0989	26.1539	26.8024
	Elaine	25.6682	27.4973	28.7723	29.6578
	Man	22.0482	23.3018	24.3623	24.9216
	Barbara	20.8122	21.5167	22.0609	22.4012
	Baboon	18.1270	18.5977	18.9179	19.2329

Table 1. Proposed algorithm performance with **prom**, **med** and **pmed** operators (the image Lena was used to generate the codebook).

Images	LBG Algorithm				Proposed algorithm (prom operator)			
	Codebook size				Codebook size			
	64	128	256	512	64	128	256	512
	PSNR	PSNR	PSNR	PSNR	PSNR	PSNR	PSNR	PSNR
Lena	27.1448	28.2131	29.0855	29.9825	26.4166	27.4702	28.3959	29.2524
Peppers	26.3830	27.1805	27.6496	28.2132	25.2016	26.0691	26.5467	27.0489
Elaine	28.9191	29.6845	30.3035	30.7453	28.3715	29.1042	29.6899	30.1218
Man	24.2034	24.9395	25.4963	26.0266	23.2950	23.9506	24.5434	25.0942
Barbara	21.8144	22.2457	22.6960	23.1359	21.3870	21.7271	22.0764	22.4347
Baboon	18.8927	19.3438	19.6829	20.0105	18.1937	18.5259	18.8603	19.1508

Table 2. Performance comparison of the proposed algorithm and the LBG algorithm.

Images	Entropy encoding technique	LBG Algorithm				Proposed algorithm (prom)			
		Codebook size				Codebook size			
		64 bpp	128 bpp	256 bpp	512 bpp	64 bpp	128 bpp	256 bpp	512 bpp
Elaine	PPM	0.168	0.216	0.290	0.394	0.174	0.225	0.304	0.410
	SZIP	0.179	0.227	0.301	0.403	0.185	0.237	0.316	0.417
	Burrown	0.190	0.236	0.313	0.419	0.191	0.245	0.328	0.433
	LZP	0.191	0.239	0.312	0.452	0.197	0.248	0.328	0.470
	LZ77	0.211	0.268	0.352	0.520	0.218	0.279	0.366	0.538
	Range	0.285	0.340	0.409	0.624	0.284	0.340	0.410	0.621
Peppers	PPM	0.185	0.234	0.307	0.414	0.194	0.245	0.323	0.429
	SZIP	0.196	0.243	0.317	0.415	0.206	0.255	0.333	0.431
	Burrown	0.267	0.254	0.329	0.429	0.215	0.264	0.346	0.446
	LZP	0.207	0.254	0.326	0.473	0.218	0.267	0.342	0.492
	LZ77	0.225	0.279	0.363	0.523	0.237	0.292	0.379	0.546
	Range	0.313	0.364	0.429	0.636	0.310	0.363	0.430	0.635
Lena	PPM	0.209	0.262	0.339	0.462	0.214	0.269	0.348	0.470
	SZIP	0.217	0.269	0.347	0.464	0.224	0.276	0.356	0.472
	Burrown	0.225	0.276	0.359	0.475	0.227	0.284	0.368	0.483
	LZP	0.227	0.276	0.354	0.522	0.234	0.284	0.364	0.532
	LZ77	0.241	0.300	0.386	0.563	0.248	0.308	0.397	0.575
	Range	0.332	0.392	0.459	0.661	0.330	0.388	0.457	0.657
Barbara	PPM	0.229	0.292	0.366	0.474	0.228	0.291	0.374	0.488
	SZIP	0.246	0.307	0.379	0.480	0.244	0.306	0.387	0.491
	Burrown	0.251	0.312	0.389	0.490	0.251	0.311	0.398	0.503
	LZP	0.257	0.315	0.387	0.541	0.258	0.316	0.397	0.559
	LZ77	0.265	0.329	0.408	0.575	0.270	0.331	0.415	0.597
	Range	0.333	0.394	0.463	0.660	0.325	0.385	0.455	0.656
Man	PPM	0.240	0.307	0.386	0.493	0.244	0.312	0.393	0.499
	SZIP	0.255	0.320	0.395	0.493	0.259	0.325	0.402	0.499
	Burrown	0.259	0.322	0.402	0.502	0.261	0.328	0.410	0.507
	LZP	0.271	0.334	0.410	0.569	0.276	0.340	0.418	0.575
	LZ77	0.284	0.351	0.428	0.605	0.289	0.357	0.433	0.612
	Range	0.328	0.389	0.452	0.651	0.326	0.387	0.452	0.650
Baboon	PPM	0.284	0.356	0.443	0.536	0.281	0.360	0.449	0.544
	SZIP	0.302	0.371	0.450	0.536	0.299	0.375	0.455	0.542
	Burrown	0.301	0.373	0.473	0.547	0.298	0.376	0.477	0.552
	LZP	0.317	0.388	0.470	0.619	0.315	0.393	0.478	0.628
	LZ77	0.322	0.391	0.470	0.651	0.322	0.390	0.471	0.664
	Range	0.336	0.402	0.470	0.663	0.328	0.397	0.470	0.664

Table 3. Compression results obtained from applying several entropy encoding technique on the information generated from the proposed algorithm.

Images	Entropy encoding technique	Proposed algorithm (med)				Proposed algorithm (pmed)			
		Codebook size				Codebook size			
		64	128	256	512	64	128	256	512
		bpp	bpp	bpp	bpp	bpp	bpp	bpp	bpp
Elaine	PPM	0.238	0.280	0.315	0.372	0.181	0.230	0.296	0.391
	SZIP	0.254	0.297	0.331	0.382	0.193	0.242	0.309	0.399
	Burrown	0.258	0.298	0.337	0.392	0.201	0.250	0.320	0.413
	LZP	0.269	0.311	0.345	0.429	0.207	0.255	0.322	0.449
	LZ77	0.289	0.332	0.366	0.498	0.226	0.281	0.355	0.515
	Range	0.324	0.360	0.387	0.559	0.277	0.330	0.396	0.603
Peppers	PPM	0.232	0.274	0.308	0.344	0.198	0.242	0.309	0.412
	SZIP	0.245	0.287	0.322	0.354	0.209	0.252	0.320	0.413
	Burrown	0.250	0.292	0.330	0.360	0.216	0.260	0.330	0.424
	LZP	0.259	0.301	0.333	0.398	0.223	0.266	0.331	0.474
	LZ77	0.275	0.320	0.358	0.473	0.241	0.289	0.362	0.527
	Range	0.327	0.362	0.392	0.560	0.302	0.349	0.411	0.619
Lena	PPM	0.254	0.300	0.330	0.363	0.214	0.267	0.339	0.452
	SZIP	0.271	0.315	0.345	0.372	0.223	0.274	0.345	0.451
	Burrown	0.277	0.320	0.351	0.385	0.230	0.281	0.355	0.461
	LZP	0.283	0.327	0.355	0.419	0.234	0.283	0.356	0.512
	LZ77	0.292	0.339	0.370	0.487	0.248	0.305	0.382	0.559
	Range	0.338	0.375	0.398	0.560	0.314	0.371	0.436	0.641
Barbara	PPM	0.266	0.310	0.340	0.374	0.236	0.292	0.364	0.473
	SZIP	0.283	0.324	0.356	0.384	0.252	0.307	0.376	0.477
	Burrown	0.285	0.328	0.361	0.395	0.253	0.313	0.384	0.489
	LZP	0.299	0.341	0.371	0.437	0.266	0.318	0.388	0.545
	LZ77	0.303	0.346	0.376	0.501	0.276	0.331	0.401	0.585
	Range	0.334	0.370	0.396	0.558	0.322	0.377	0.440	0.644
Man	PPM	0.261	0.307	0.335	0.373	0.245	0.305	0.378	0.488
	SZIP	0.280	0.324	0.351	0.381	0.259	0.318	0.390	0.489
	Burrown	0.279	0.323	0.355	0.385	0.330	0.321	0.395	0.499
	LZP	0.300	0.344	0.370	0.436	0.274	0.333	0.404	0.565
	LZ77	0.301	0.345	0.373	0.500	0.288	0.347	0.418	0.604
	Range	0.323	0.361	0.385	0.556	0.320	0.376	0.438	0.641
Baboon	PPM	0.313	0.353	0.381	0.416	0.286	0.359	0.446	0.542
	SZIP	0.331	0.368	0.394	0.421	0.303	0.374	0.454	0.541
	Burrown	0.405	0.367	0.397	0.424	0.301	0.373	0.462	0.552
	LZP	0.350	0.391	0.418	0.488	0.320	0.392	0.476	0.627
	LZ77	0.339	0.373	0.397	0.544	0.324	0.388	0.470	0.662
	Range	0.334	0.368	0.392	0.560	0.330	0.395	0.469	0.663

Table 3. Compression results obtained from applying several entropy encoding technique on the information generated from the proposed algorithm (continuation).

These results show that when these coding methods are applied on the results obtained by our algorithm, the entropy coding technique that offers the best results in compression ratio is the PPM coding. The PPM is an adaptive statistical method; its operation is based on partial equalization of chains, that is, the PPM coding predicts the value of an element based on the sequence of previous elements.
Furthermore, these results show that the proposed algorithm remains competitive with the algorithm LBG in the compression parameter.
Finally, based on results obtained in section 3.4 and the study for LBG algorithm presented in (Guzmán et al., 2008), a complexity analysis of both algorithms was realized. Table 4 summarizes computation complexities of the encoding phase of both LBG and proposed algorithm in terms of the type and average number of operations per pixels. This experiment was performed for n = 16 and 64, which are the most popular pattern dimensions. With regard to time complexity, this Table shows that the proposed algorithm provides considerable improvement over the LBG algorithm. Table 4 also shows that the memory required by our algorithm is smaller than the memory needed by the LBG when a VQ process is performed.

Algorithm	**Pattern dimension** (n)	**Codebook size** (N)	**Required memory** (bytes)	**Average number of operations per pixel**				
				CMP	±	×	**SQRT**	**Shift**
LBG	16	128	41088	8	387	129	8.06	-
		256	49280	16	771	257	16.06	-
		512	65664	32	1539	513	32.06	-
	64	128	41472	2	387	129	2.015	-
		256	74240	4	771	257	4.015	-
		512	139776	8	1539	513	8.015	-
Our proposal with **prom** and **pmed** operators	16	128	18560	136	128	-	-	-
		256	20736	272	256	-	-	-
		512	41472	544	512	-	-	-
	64	128	12416	130	128	-	-	-
		256	20736	260	256	-	-	-
		512	37376	520	512	-	-	-
Our proposal with **med** operator	16	128	16512	10.062	10.125	-	-	0.1875
		256	16640	18.062	18.125	-	-	0.1875
		512	33280	34.062	34.125	-	-	0.1875
	64	128	4224	4.015	4.031	-	-	0.0468
		256	4352	6.015	6.031	-	-	0.0468
		512	8704	10.015	10.031	-	-	0.0468

Table 4. Computation complexities of the encoding phase of both proposed algorithm and the LBG algorithm.

5. Conclusions

In the present work, we have proposed the use of extended associative memories in a high-speed search algorithm applied to image quantization. With the purpose of evaluating the influence that the proposed VQ algorithm has in the image compression, it was integrated in a codec where the codification stage includes to several standard codification methods. The compression ratio and the signal to noise ratio obtained by our proposal show that the rate of compression and the decoding quality remain competitive with respect to the LBG algorithm.

Furthermore, the EAM has the property of substituting the input pattern by the class index that present the nearest match, without taking care that they have not been used in the construction of the associative memory. Our Algorithm inherited this property. This property allows our algorithm to quantify efficiently images that have not been used in the generation of the EAM-codebook, this is the reason it obtains good results in the decoding quality parameter.

Finally, in most of the data processing, the proposed algorithm makes use of the morphological operations, that is to say, its operation is based on maximums or minimums of sums, it uses only the operations of sums and comparisons. Therefore, the proposed algorithm offers a high processing speed in the search process of the encoding phase. Thus, with respect to traditional VQ algorithms, the main advantages offered by the proposed algorithm are, a high processing speed and low demand of resources (system memory).

For these reasons, we can conclude that our proposal provides a considerable improvement over the LBG algorithm.

6. References

Amerijckx, C., Verleysen, M., Thissen, P. and Legat, J.-D. (1998). Image Compression by Self-Organized Kohonen Map. *IEEE Transactions on Neural Networks*, Vol. 9, No 3, pp. 503-507, ISSN 1045-9227.

Amerijckx, C., Legat, J.-D. and Verleysen, M. (2003). Image Compression using Self-Organizing Maps. *Systems Analysis Modelling Simulation, Taylor & Francis Ltd.*, Vol. 43, No. 11, pp. 1529-1543, ISSN 0232-9298.

Barron, R. (2006). Associative Memories and Morphological Neural Networks for Patterns Recall. Ph.D. dissertation, Center for Computing Research of National Polytechnic Institute, México.

Basil, G. and Jiang, J. (1999). An Improvement on Competitive Learning Neural Network by LBG Vector Quantization. Proceedings of IEEE International Conference on Multimedia Computing and Systems, Vol 1, pp 244-249, ISBN 978-1-4244-3756-6, Florence, Italy.

Chin-Chuan, H., Ying-Nong, C., Chih-Chung, L., Cheng-Tzu, W. and Kuo-Chin, F. (2006). A Novel Approach for Vector Quantization Using a Neural Network, Mean Shift, and Principal Component Analysis. Proceedings of *IEEE Intelligent Vehicles Symposium*, pp 244-249, ISBN 4-901122-86-X, Tokyo, Japon.

Chin-Chuan, H., Ying-Nong, C., Chih-Chung, L. and Cheng-Tzu W. (2007). A Novel Approach for Vector Quantization Using a Neural Network, Mean Shift, and Principal Component Analysis-based seed re-initialization. *Signal Processing, Journal*. Elsevier. Vol. 87, Issue 5, pp 799-810, ISSN 0165-1684.

Gersho, A. and Gray, R. M. (1992). *Vector Quantization and Signal Compression*. Kluwer Academic, ISBN 0-7923-9181-0, Norwell, Massachusetts USA.

Gray, R. M. (1984). Vector Quantization. *IEEE ASSP Magazine*, Vol 1, pp. 4-9, ISSN 0740-7467.

Guzmán, E., Oleksiy, P. and Yánez, C. (2007). Design of an Evolutionary Codebook Based on Morphological Associative Memories. *Proceeding of the 6th Mexican International Conference on Artificial Intelligence,* LNAI 4827, Springer, Vol. 4827/2007, pp. 601-611, ISSN 0302-9743, Aguascalientes, México.

Guzmán, E., Oleksiy, P., Sánchez, L. and Yánez, C. (2008). A Fast Search Algorithm for Vector Quantization based on Associative Memories. *Proceeding of the 13th Iberoamerican congress on Pattern Recognition,* LNCS 5197, Springer, Vol. 5197/2008, pp. 487-495, ISSN 0302-9743, Havana, Cuba.

Kohonen, T. (1980). Automatic formation of topological maps of patterns in a self-organizing system., *Proceedings of 2nd Scandinavian Conference on Image Analysis*, pp. 214-220, Helsinki, Finland.

Kohonen T. (1982). Self-organizing formation of topologically correct feature maps. *Biological Cybernetics*, Vol. 43 No. 1, pp. 59-69, ISSN 0340-1200.

Linde, Y., Buzo, A. and Gray R. (1980). An Algorithm for Vector Quantizer Design. *IEEE Trans. on Communications*, Vol 28, No. 1, pp. 84-95, ISSN 0090-6778.

Lloyd, L. P. (1982). Least Squares Quantization in PCM. *IEEE Trans. Inform. Theory*, Vol. IT-28, No. 2, pp. 129-137, ISSN 0018-9448.

Nasrabadi, N. M. and King, R. A. (1988). Image Coding Using Vector Quantization: A Review. *IEEE Trans.on Communications*, Vol. 36, No. 8, pp. 957-971, ISSN 0090-6778.

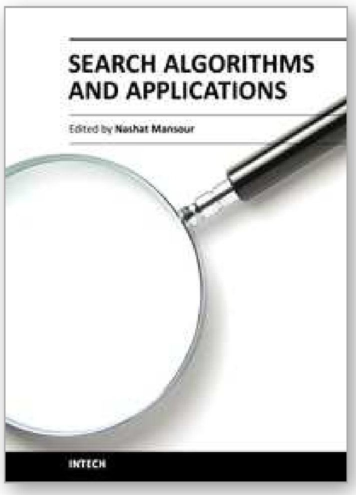
Sossa, H., Barrón, R. and Vázquez, A. (2004). Real-valued Patterns Classification based on Extended Associative Memory, *Proceedings of IEEE 15th Mexican International Conference on Computer Science*, pp. 213-219, ISBN 0-7695-2160-6, Colima, México.

Steinbuch, K. (1961). Die Lernmatrix, Kybernetik, Vol. 1 No. 1, pp. 26-45, ISSN 0340-1200.

Vázquez, R. (2005). Objects recognition in presence of traslapes by means of associative memories and invariants descriptions. Master dissertation, Center for Research in Computing of National Polytechnic Institute, México.

Yair, E., Zeger, K. and Gersho, A. (1992). Competitive Learning and Soft Competition for Vector Quantizer Design. *IEEE Transaction on Signal Processing*, Vol. 40, No. 2, pp. 294-309, ISSN 1053-587X.

Yong-Soo, K. and Sung-Ihl K. (2007) Fuzzy Neural Network Model Using a Fuzzy Learning Vector Quantization with the Relative Distance. *Proceedings of IEEE 7th International Conference on Hybrid Intelligent Systems*, pp. 90-94, ISBN 978-0-7695-2946-2, Kaiserslautern, Germany.



Search Algorithms and Applications
Edited by Prof. Nashat Mansour

ISBN 978-953-307-156-5
Hard cover, 494 pages
Publisher InTech
Published online 26, April, 2011
Published in print edition April, 2011