The Enhanced LBG Algorithm

Giuseppe Patané [a] and Marco Russo [b,1]

[a] *Institute of Computer Science and Telecommunications, Faculty of Engineering, University of Catania, Viale A. Doria 6, 95125 Catania, Italy and INFN, Section of Catania, Corso Italia 57, 95129 Catania, Italy*

[b] *Department of Physics, University of Messina, Contrada Papardo, Salita Sperone 31, 98166 Messina, Italy and INFN, Section of Catania, Corso Italia 57, 95129 Catania, Italy; e-mail: marco.russo@ct.infn.it*

---

[1] Marco Russo, Department of Physics, University of Messina, Contrada Papardo, Salita Sperone 31, 98166 Messina, Italy; e-mail: marco.russo@ct.infn.it

The Enhanced LBG Algorithm

Abstract

Clustering applications cover several fields such as audio and video data compression, pattern recognition, computer vision, medical image recognition, etc. In this paper we present a new clustering algorithm called Enhanced LBG (ELBG). It belongs to the hard and K-means vector quantization groups and derives directly from the simpler LBG. The basic idea we have developed is the concept of utility of a codeword, a powerful instrument to overcome one of the main drawbacks of clustering algorithms: generally, the results achieved are not good in the case of a bad choice of the initial codebook. We will present our experimental results showing that ELBG is able to find better codebooks than previous clustering techniques and the computational complexity is virtually the same as the simpler LBG.

Key words: Clustering, Unsupervised Learning, LBG, GLA, LVQ, K-means, Hard c-means, Fuzzy c-means.

1 Introduction

Clustering is an important instrument in engineering and other scientific disciplines. Its applications cover several fields such as audio (Paliwal & Atal, 1993) and video (Lookbaugh, Riskin, Chou, & Gray, 1993; Silva, Sampson, & Ghanbari, 1996; Cosman, Gray, & Vetterli, 1996) data compression, pattern recognition (Fukunaga, 1990), computer vision (Jolion, Meer, & Bataouche, 1991), medical image recognition (Perlmutter, Perlmutter, Gray, Olshen, & Oehler, 1996), and so on.

The partitioning approach known as Vector Quantization (VQ) (Gersho & Gray, 1992) derives a set (codebook) of reference or prototype vectors (codewords) from a data set. In this manner each element of the data set is represented by only one codeword. Codewords are determined trying to minimize an objective function (distortion) representing the Quantization Error (QE) (Hofmann & Buhmann, 1997).

A widespread accepted classification scheme subdivides these techniques into two main groups: hard (crisp) (Linde, Buzo, & Gray, 1980) or soft (fuzzy) (Bezdek & Pal, 1995). The difference between these is mainly the degree of membership of each vector to the clusters. During the construction of the codebook, in the case of the hard group, each vector belongs to only one cluster with a unitary degree of membership, whereas, for the fuzzy group, each vector can belong to several clusters with different degrees of membership.

Another classification scheme distinguishes two other main groups: K–means and competitive learning. The clustering techniques belonging to the first scheme (Linde et al., 1980) try to minimize the average distortion through a suitable choice of codewords. In the second case (Kohonen, 1989) the codewords are obtained as a consequence of a process of competition between them.

However, the performance of many VQ algorithms is strongly dependent on the choice of the initial conditions and the configuration parameters; many works have been developed in literature in order to solve this problem (Kohonen, 1989; Pal, Bezdek, & Tsao, 1993; Gonzalez, Graña, & D'Anjou, 1995; Bezdek & Pal, 1995; Chinrungrueng & Séquin, 1995; Karayiannis & Pai, 1996; Karayiannis, Bezdek, Pal, Hathaway, & Pai, 1996; Karayiannis, 1997; Hofmann & Buhmann, 1997).

In this paper we present the new algorithm we called Enhanced LBG (ELBG). It belongs to the hard and K–means vector quantization groups and derives directly from the simpler LBG (Linde et al., 1980). The basic idea we developed is the concept of utility of a codeword. Even if some authors already introduced the utility (Fritzke, 1997), our definition, meaning and computa-

tional complexity are totally different. The utility index we use is a powerful instrument to overcome some of the greatest drawbacks of the LBG and other VQ algorithms. As we have already stated, one of the main problems is that, in the case of a bad choice of the initial codebook, generally, the results are not good. The utility allows a good identification of these situations. Further, it permits the recognition of the badly positioned codewords and gives us useful indications about regions where they should be placed. Our paper, like work from Chinrungrueng and Séquin (1995), has been inspired by a theorem from Gersho (1979). This theorem states that, if some hypothesis are verified, the distortion associated to each codeword is the same as the others in an optimal codebook. In the same way, ELBG looks for a codebook to which each codeword contributes in the same manner, i.e. the utility of all the codewords is the same.

The experimental results we have reached show that ELBG is able to find better codebooks than previous works, that it is practically independent of the initial conditions and that the computational complexity is virtually the same as the simpler LBG algorithm.

The paper is organized as follows: in Section 2 we introduce the basic VQ concepts and the symbols we adopted; in Section 3 we examine the two main necessary conditions so that a quantizer can be said to be optimum; in Section 4 we describe the classical LBG algorithm; in Section 5 we make some considerations about the traditional LBG; in Section 6 we illustrate our algorithm; in Section 7 we discuss the utility concept; in Section 8 we show that the overhead introduced by our algorithm is negligible with respect to standard LBG; in Section 9 we show our results and comparisons; lastly, Section 10 contains the authors' conclusions.

2 Vector Quantization

2.1 Definition

The objective of VQ is the representation of a set of feature vectors $\mathbf{x} \in X \subseteq \Re^k$ by a set, $Y = \{\mathbf{y}_1, ..., \mathbf{y}_{N_C}\}$, of N_C reference vectors in $\Re^k$. Y is called *codebook* and its elements *codewords*. The vectors of X are called also *input patterns* or *input vectors*. So, a VQ can be represented as a function: $q : X \longrightarrow Y$. The knowledge of q permits us to obtain a partition $\mathcal{S}$ of X constituted by the N_C subsets S_i (called cells):

$$S_i = \{\mathbf{x} \in X : q(\mathbf{x}) = \mathbf{y}_i\} \quad i = 1, \ldots, N_C \tag{1}$$

In the following, we will suppose that we are dealing with a finite input data set constituted by N_P elements, i.e. $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N_P}\}$.

2.2 Quantization Error

The Quantization Error (QE) is the value assumed by $d(\mathbf{x}, q(\mathbf{x}))$, where d is a generic distance operator for vectors. The mean QE (MQE) is:

$$\text{MQE} \equiv D(\{Y, \mathcal{S}\}) = \frac{1}{N_P} \sum_{p=1}^{N_P} d(\mathbf{x}_p, q(\mathbf{x}_p)) = \frac{1}{N_P} \sum_{i=1}^{N_C} D_i \tag{2}$$

where we indicate with D_i the ith cell total distortion

$$D_i = \sum_{n:\mathbf{x}_n \in S_i} d(\mathbf{x}_n, \mathbf{y}_i) \tag{3}$$

Several functions can be adopted as distortion measures (Linde et al., 1980). The most widely adopted is the squared Euclidean distance (Squared Error, SE)

$$d(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{k} (x_i - x_i')^2 \tag{4}$$

and we will use it in this paper. In this case, the MQE is called Mean Squared Error (MSE). The square root of the MSE (RMSE) is also used. Other times, Normalized Mean Squared Error (NMSE) is adopted. It corresponds to the MSE divided by the MSE obtained with a codebook of only one codeword, $\mathbf{c}$, placed at the centroid [2] of the whole data set:

$$\text{NMSE} = \frac{\text{MSE}}{\frac{1}{N_P} \sum_{p=1}^{N_P} d(\mathbf{x}_p - \mathbf{c})} \tag{5}$$

The interested reader can see (Linde et al., 1980) for a more detailed description of the MQE and for other quadratic distorsion measures.

[2] see subsection 3.2 for the definition of centroid

3 Optimal quantizer

A quantizer is optimum when, for each other quantizer with the same number of codewords, a higher MQE is found. In mathematical terms, q^* is optimum if, for each other q, we have $D(q^*) \leq D(q)$.

In the following, the authors will describe the two main conditions which, from a mathematical point of view, are necessary so that a quantizer can be said to be optimum. The two conditions are usually called the Nearest Neighbour Condition (NNC) and the Centroid Condition (CC).

3.1 Nearest Neighbor Condition

Given a fixed codebook Y, the NNC consists in assigning to each input vector the nearest codeword. So, we divide the input data set in the following manner:

$$\begin{aligned}\bar{S}_i = \{&\mathbf{x} \in X :\ d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j), \\ &j = 1, ..., N_C,\ j \neq i\} \quad i = 1, ..., N_C\end{aligned} \tag{6}$$

The sets $\bar{S}_i$ just defined, constitute a partition of the input data set. This is the "Voronoi Partition" (Gersho & Gray, 1992) and is refered to with the symbol $\mathcal{P}(Y) = \{\bar{S}_1, \cdots, \bar{S}_{N_C}\}$. As $\mathcal{P}(Y)$ must be a partition, when an input vector has the same distance from two or more codewords, it needs to choose a unique manner to assign this vector to only one $\bar{S}_i$.

NNC permits us to obtain an optimal partition (Linde et al., 1980), i.e. for every partition $\mathcal{S}$ of the input data set, it holds:

$$D(\{Y, \mathcal{S}\}) \geq D(\{Y, \mathcal{P}(Y)\}) \tag{7}$$

3.2 Centroid Condition

Given a fixed partition $\mathcal{S}$, the CC concerns the procedure to find the optimal codebook.

Let us define centroid or center of gravity of a given set $A \subseteq \Re^k$ the vector $\bar{\mathbf{x}}(A)$ for which:

$$E\{d(\mathbf{x}, \bar{\mathbf{x}}(A)) \mid \mathbf{x} \in A\} = \min_{\mathbf{u} \in \Re^k} E\{d(\mathbf{x}, \mathbf{u}) \mid \mathbf{x} \in A\} \tag{8}$$

If the number of elements of A is N_A and the squared Euclidean distance is adopted (4), we have:

$$\bar{\mathbf{x}}(A) = \frac{1}{N_A} \sum_{\mathbf{x} \in A} \mathbf{x} \tag{9}$$

If we take the codebook $\bar{X}(\mathcal{S})$ constituted by the centroid of all the cells of $\mathcal{S}$:

$$\bar{X}(\mathcal{S}) \equiv \{\bar{\mathbf{x}}(S_i); i = i, ..., N_C\} \tag{10}$$

it is optimum (Linde et al., 1980), i.e. for every codebook Y, it holds:

$$D\{Y, \mathcal{S}\} \geq D(\{\bar{X}(\mathcal{S}), \mathcal{S}\}) \tag{11}$$

4 Generalized Lloyd Algorithm (GLA) or LBG

In 1980 Linde, Buzo and Gray (Linde et al., 1980) proposed an improvement of the Lloyd's technique (Lloyd, 1957). They extended Lloyd's results from mono- to k-dimensional cases. For this reason their algorithm is known as the Generalized Lloyd Algorithm (GLA) or LBG from the initials of its authors.

In a few words, the LBG algorithm is a finite sequence of steps in which, at every step, a new quantizer, with a total distortion less or equal to the previous one, is produced.

Now, we will describe the LBG steps. We can distinguish two phases, as shown in Fig. 1: the initialization of the codebook and its optimization. In the initialization phase two methods are mainly used: random and by splitting.

Firstly, we will describe the optimization step. It will simplify the LBG explanation. In fact, several concepts necessary to describe this step are useful for the initialization phase, too. In the following we will use these symbols:

- m: iteration number;
- Y_m: mth codebook;
- D_m: MQE calculated at the mth iteration.

4.1 Codebook optimization

Fig. 2 shows the high-level flow-chart. The codebook optimization starts from an initial codebook and, after some iterations, generates a final codebook with

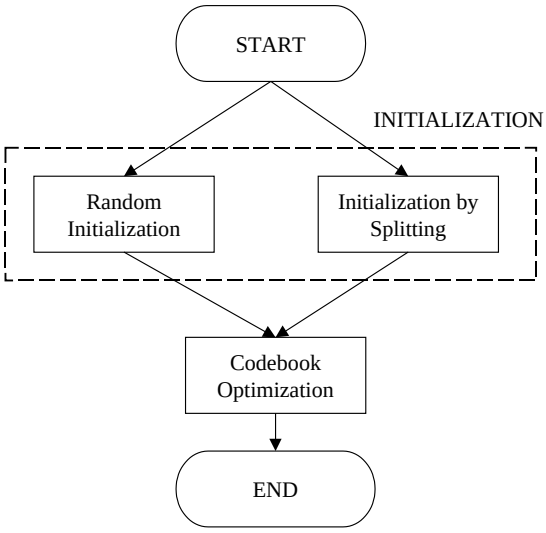

Fig. 1. The LBG procedure

a distortion corresponding to a local minimum.

(1) **Initialization.** The following values are fixed:
- N_C: number of codewords;
- $\epsilon \geq 0$: precision of the optimization process;
- Y_0: initial codebook;
- $X = \{\mathbf{x}_j;\ j = 1, ..., N_P\}$: input patterns;

Further, the following assignments are made:
- $m = 0$;
- $D_{-1} = +\infty$;

(2) **Partition calculation.** Given the codebook Y_m, the partition $\mathcal{P}(Y_m)$ is calculated according to the NNC (6).

(3) **Termination condition check.** The quantizer distortion ($D_m = D(\{Y_m, \mathcal{P}(Y_m)\})$ is calculated according to eq. (2). If $\mid (D_{m-1} - D_m) \mid / D_m \leq \epsilon$ then the optimization ends and Y_m is the final returned codebook [3].

(4) **New codebook calculation.** Given the partition $\mathcal{P}(Y_m)$, the new codebook is calculated according to the CC(10). In symbols:

$$Y_{m+1} = \bar{X}(\mathcal{P}(Y_m)) \tag{12}$$

After, the counter m is increased by one and the procedure follows from step 2.

[3] The termination condition depends both on the ϵ value and the adopted distortion measure. It is meaningless to specify only the ϵ value because, with two different distortion measures (as, for example, the MSE and the RMSE), the expected value of the number of iterations can drastically change and, of course, the final mean distortion value. In this paper, each time we specify an ϵ value, it refers to the RMSE. A typical range of values for ϵ is $[0.001, 0.1]$.

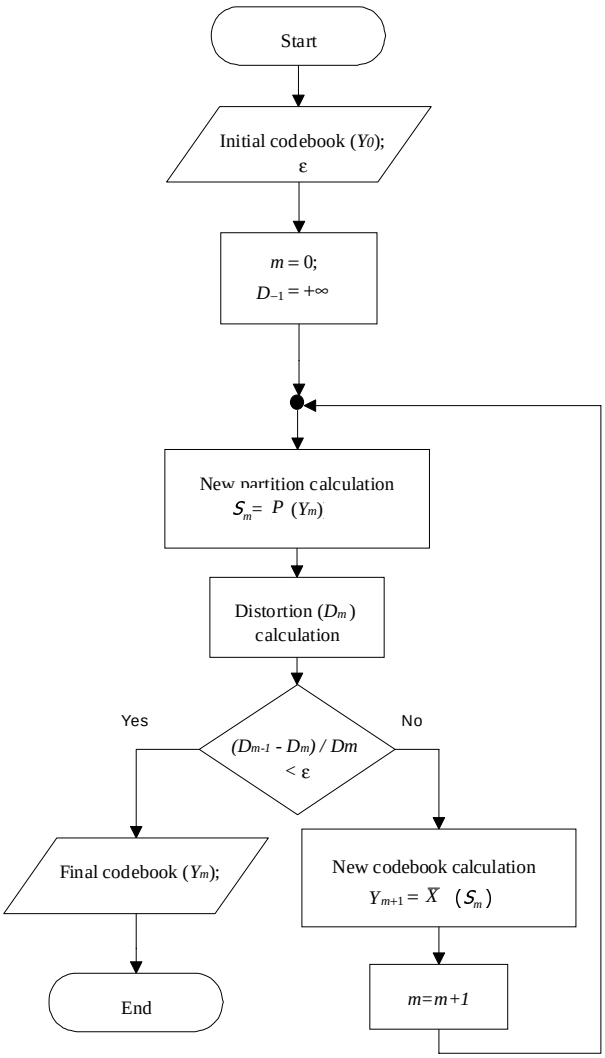

Fig. 2. LBG codebook optimization

In (Linde et al., 1980) it is demonstrated that these steps assure that the series D_m is not increasing and convergent.

4.2 Initialization of the codebook

The codebook initializiation is a very important task. In fact, a bad choice of the initial codewords generally leads to a final quantizer with a high MQE. Here, we describe the random initialization and the initialization by splitting.

- **Random initialization**. The initial codewords are randomly chosen (Pal et al., 1993). Generally they are chosen inside the convex hull of the input data set.

- **Initialization by splitting**. This initialization requires that the number of codewords is a power of 2. The procedure starts from only one codeword that, recursively, splits it in two distinct codewords (Linde et al., 1980). More precisely, the generic mth step consists in the splitting of all vectors obtained at the end of the previous step. After the splitting, an optimization step is executed according to the method described in sub-section 4.1.

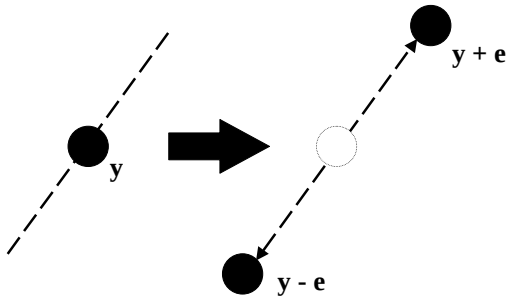

Fig. 3. Splitting of a codeword

The splitting criterion is shown in Fig. 3. It starts from one codeword $\mathbf{y}$. It splits this vector into two close vectors $\mathbf{y} + \mathbf{e}$ and $\mathbf{y} - \mathbf{e}$ where $\mathbf{e}$ is a fixed perturbation vector.

These techniques are not the only ones present in literature. From the others, we cite the maximum distance initialization (Katsavounidis, Kuo, & Zhang, 1994).

5 Considerations about the LBG algorithm

The algorithm just presented usually finds a locally optimum quantizer. The main problem is that, often, this optimum is very far from an acceptable solution.

If we qualitatively comment the analytical expressions regarding the codeword adjustment we could say that at each iteration codewords "move" through contiguous regions. This implies that a bad initialization could lead to the impossibility of finding a good quantizer.

For example, let us examine Fig. 4. On the left side, part (a), we see the codeword number 4. According to the (6), it will always generate an empty cell because all the elements of the data set are nearer to the other codewords. So, following the steps of the traditional LBG, it cannot move and will never represent any element. For this reason we can say it is useless. The same authors of the LBG (Linde et al., 1980) proposed some solutions to this problem such as the assigning of the codeword to a non-empty cell.

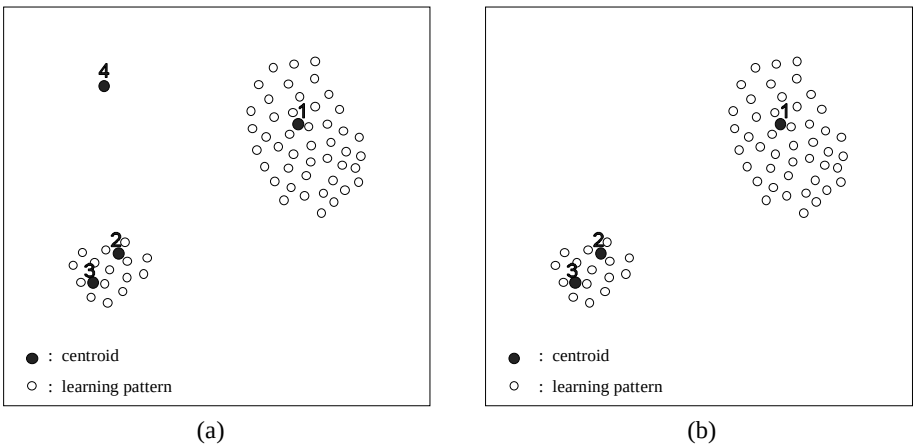

Fig. 4. Badly positioned codewords

But we think there is another problem that strongly limits the classical LBG and its solution appears a difficult task. Let us look at the right side, part (b), of Fig. 4. This configuration shows two clusters and three codewords. In the little cluster there are two codewords whereas, in the other, only one. The elements in the data set in the smaller cluster are all well approximated by the two related codewords. Instead, a lot of elements in the larger one are badly approximated by the related codeword. For this geometrical distribuition, it would be preferable that two codewords were inside the big cluster and only one in the other, but the LBG optimization algorithm, in this situation, does not permit the migration of a codeword from the little cluster to the big one. This is a great limitation.

To improve the performance of the LBG algorithm, we think that it is crucial to develop a criterion that identifies these situations. Further, it must be able to find which codewords it is better to move and where they have to be placed, without any contiguity limitation.

Some authors already introduced some interesting criterions (Fritzke, 1997).

6 Enhanced LBG (ELBG)

The aim of this section is to explain the ELBG algorithm in detail.

6.1 General considerations

The algorithm we propose is an attempt to find a solution for the two drawbacks of the classical LBG we discussed in section 5. We will formally introduce

a new quantity that we call the utility of a codeword. It allows us to deal with both drawbacks described in the previous section from a unique point of view.

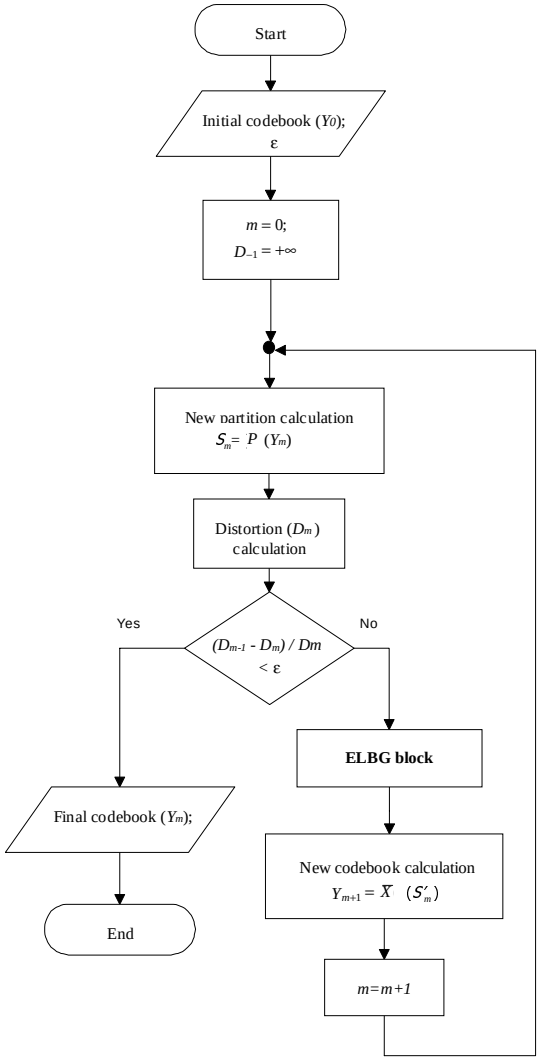

Fig. 5. ELBG codebook optimization

Fig. 5 shows the high-level flow-chart of the ELBG algorithm. The only difference between LBG and ELBG is the ELBG block. The functionalities of the ELBG block are summarized in Fig 6. First of all, there is the utility evaluation. After the evaluation of the utilities of the codewords, we identify the ones with a low utility. This information is very useful for the next step: the smart shifting of codewords. We try to shift all the low-utility codewords near to the ones with high utility. Each attempt leading to a lower MQE is confirmed. The aim of these operations is to obtain the equalization of the total distorsions related to cells (D_i, see eq. (3)), as one of Gersho's theorems suggests (Gersho, 1979, 1986). As we will see in detail in the next sub-sections, this allows us to overcome the drawbacks we exposed in section 5.

Here, we only say that the heart of the ELBG block is the execution of several

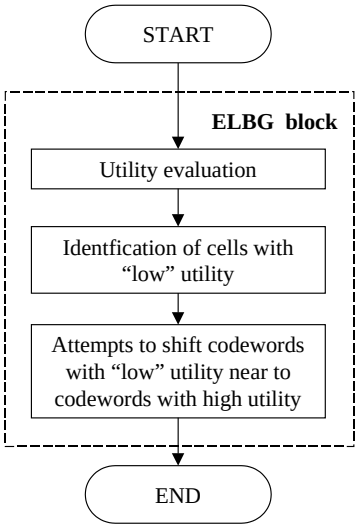

Fig. 6. High-level flow-chart of the ELBG block

Shifting of Codewords Attempts (SoCA's). When a SoCA produces a decrease in the MQE, then the SoCA is confirmed. In this case we say that a Shift of Codeword (SoC) is executed. If we do not have any MQE decrease, the shift is discarded.

Besides, we wish to underline that all the additional steps we introduced in the LBG algorithm to obtain the ELBG are very efficient. The term "efficient" refers to a low computational complexity operation. So, the overhead we introduced in the original LBG is negligible, as will be shown in section 8.

6.2 Distortion equalization and Utility

The idea of the utility was suggested to us by one of Gersho's theorems (Gersho, 1986) where he explained his partial distortion theorem (Gersho, 1979) saying: "*Each cell makes an equal contribution to the total distortion in optimal vector quantization with high resolution*". Gersho's theorem is true when certain conditions are verified (according to Gersho (1979), a high resolution quantizer has a number of codewords tending to infinite). But Chinrungrueng and Séquin (1995), experimentally, proved that it maintains a certain validity also when the codebook has a finite number of elements. So, we introduce a new step inside the LBG to pursue the equalization of the total distortions of the cells (D_i). In this context, we define the "utility index" (U_i) of the ith cell as the value of D_i normalized with respect to its mean value (D_{mean}). In formal terms, we have:

$$D_{\text{mean}} = \frac{1}{N_C} \sum_{i=1}^{N_C} D_i \tag{13}$$

$$U_i = \frac{D_i}{D_{\text{mean}}} \quad i = 1, ..., N_C \tag{14}$$

In the following, we will use both the term utility index of a cell and utility index of a codeword. Substantially, there is no difference between the two terms. In fact, we can use equations (13) and (14) only if a cell is considered together with the related codeword and vice versa. We will often use only the shorter term "utility".

According to the definition just given, the equalization of the distorsions is equivalent to the equalization of the utilities.

Our idea is to obtain the desired equalization by joining, for each SoCA, a low-utility (lower than 1) cell with a cell adjacent to it, hoping to obtain a bigger cell whose utility is closer to 1 than before. At the same time, we split a high-utility (higher than 1) cell into two smaller ones whose utilities, are, if possible, closer to 1 than the big cell. We can say that this operation is equivalent to move the low-utility codeword inside the high-utility cell. If we refer to Fig. 4. (a) we see that the utility of cell 4 (U_4) is 0, that U_2 and U_3 are lower than 1 and that U_1 is greater then 1. These values, according to the possibility illustrated above, suggest moving the 4th codeword near the 1st codeword. Instead, if we see Fig. 4. (b) we should move codeword 2 or 3 near the 1st one. This is a "smart" manner of shifting codewords that allows their migration through non-contiguous regions.

Fig. 12 shows a typical distribution of the utility indexes when an initial random codebook is given. It is a very widespread "bell". The shifting of the codewords on the left side of the figure near to the ones on the right side produces an adjustment of the original bell into a narrow one, as is shown in Fig. 14. However, we must remember that our primary objective is the MQE minimization. So, we execute a SoC only when we are sure that it produces a mean distortion decrease. The way we execute a SoC and the evaluation of its effect on the QE are the argument of the next subsections.

The considerations exposed in this sub-section allowed us to develop an objective criterion to select the codewords to be shifted and the cells where they have to be placed, as will be explained in the next sub-sections.

6.3 Detailed description of the ELBG block

Fig. 7 details the previous Fig. 6 in which the main steps of the ELBG block are illustrated.

The ELBG consists in the execution of a certain number of SoCA's. In what

follows, we will describe in detail a SoCA. We will use Figs. 8, 9, 10, 11. They represent a SoCA for a simple bi-dimensional problem.

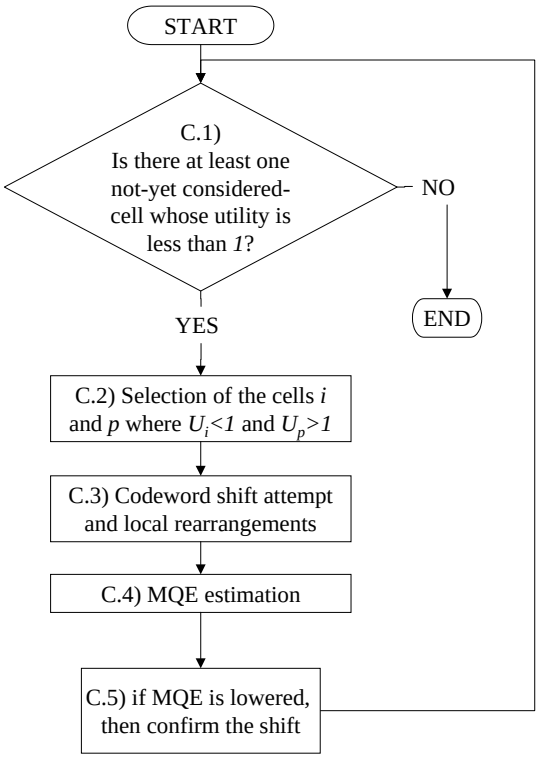

Fig. 7. Detailed description of the ELBG block

6.3.1 Termination condition

The first condition in the upper part of Fig. 7 regards the termination of the whole ELBG block. We check to see if at least one cell has a utility index lower than 1 and it has not been involved in previous shifts. If no cell exists, then the algorithm ends. Otherwise the next steps regarding a new SoCA follow.

6.3.2 Selection of cells

This step is necessary to recognize all the cells involved in the current SoCA. We look for two different cells.

- One cell must have a utility index less than 1. We will refer to it as the ith cell.
- One cell must have a utility index greater than 1. We will indicate it as the pth cell.

The ith cell, S_i, is searched for in a sequential manner. Instead, the pth cell is looked for in a stochastic way. The method adopted sounds like the roulette

wheel selection in genetic algorithms (Russo, 1998). Practically, we choose a cell with a probability P_p proportional to its utility value. In mathematical terms:

$$P_p = \frac{U_p}{\sum_{h:U_h>1} U_h} \tag{15}$$

6.3.3 Codeword shift and local rearrangements

This step consists in a SoCA. We try to shift the codeword $\mathbf{y}_i$ near $\mathbf{y}_p$, i.e. the codewords related to the cells S_i and S_p respectively. This situation is illustrated in Fig. 8 for our bi-dimensional problem.

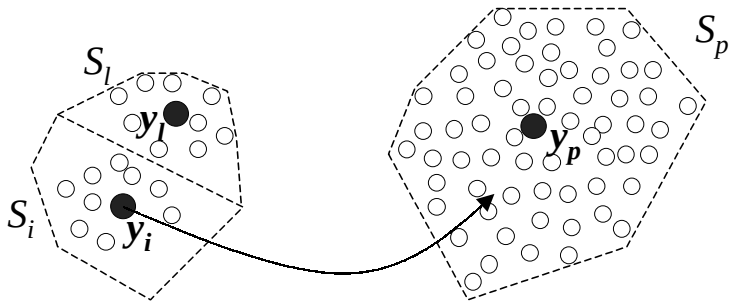

Fig. 8. Initial situation before the SoCA

A similar shift produces a new codebook. In the traditional LBG, after a new codebook generation, the calculation of the partition satisfying the NNC, i.e. the Voronoi partition, follows. As we have several SoCA's and each of them must be evaluated, we would have to introduce a very heavy overhead into the classical LBG [4]. Our aim is to improve the LBG with a low overhead, so we avoid recalculating the Voronoi partition.

To simplify the overall procedure and, of course, to drastically reduce the overhead, we suppose that, after the shift, in the new partition only the patterns related to S_i and S_p will be subject to change. In our algorithm, these patterns will be the only ones with related codewords different from before. We shift $\mathbf{y}_i$ near $\mathbf{y}_p$. As $\mathbf{y}_p$ is, generally, localized at the center of S_p, we think that it is better to move $\mathbf{y}_p$, too. So, it is possible to distribute the two codewords inside the S_p cell in a better way. The solution we have found is very simple. It does not assure a better distribution, but we made a lot of experimental trials that have shown its validity.

As a finite number of k-dimensional vectors forms the input data set, the

[4] We must remember that the most onerous step in the LBG algorithm is precisely the Voronoi partition calculation.

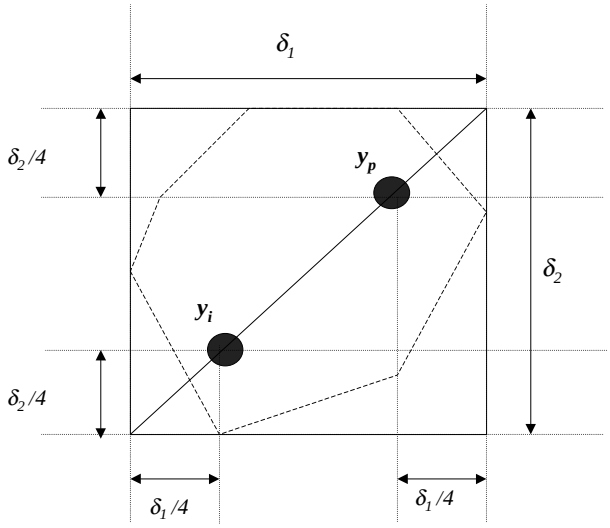

Fig. 9. The cell S_p and the hyperbox containing it

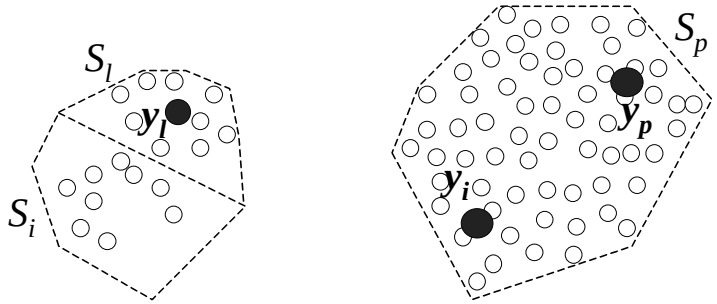

Fig. 10. Codewords position immediately after the shift

generic pth cell is contained in the k-dimensional hyperbox I_p:

$$I_p = [x_{1\mathrm{m}}, x_{1\mathrm{M}}] \times [x_{2\mathrm{m}}, x_{2\mathrm{M}}] \times ... \times [x_{k\mathrm{m}}, x_{k\mathrm{M}}] \tag{16}$$

where $x_{h\mathrm{m}}$ and $x_{h\mathrm{M}}$ are respectively the minimum and maximum value assumed by the hth dimension of all patterns belonging to S_p. We place $\mathbf{y}_i$ and $\mathbf{y}_p$ on the principal diagonal of I_p. We divide the diagonal in three parts. Two are equal to the half of the central one. We place the two codewords at the ends of the central part as is illustrated in Figs. 9 and 10.

Afterwards, $\mathbf{y}_i$ and $\mathbf{y}_p$ are adjusted with a local traditional LBG with a high value for ϵ (typically $0.1 \div 0.3$), so only a very few iterations (one or two) are generally executed. The codebook to be optimized contains only $\mathbf{y}_i$ and $\mathbf{y}_p$ and the input data set is S_p. The result of this optimization step is two new codewords ($\mathbf{y}'_i$ and $\mathbf{y}'_p$) and two new cells (S'_i and S'_p). The new codewords and the new cells substitute the corresponding ones in the old codebook and partition respectively. We show this in the right part of Fig. 11.

After the patterns belonging to S_p have been rearranged, we still have to

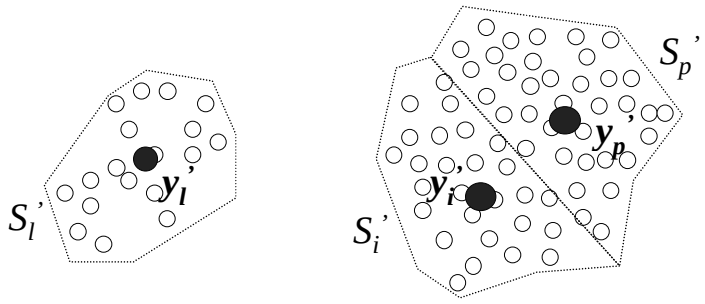

Fig. 11. Codewords position and patterns distribution after the local rearrangements

rearrange the patterns of S_i because the related codeword has been moved away. We must again remember that the optimum way to assign these vectors is to calculate, for each of them, the distance from all of the codewords. As we want to avoid this operation, we adopt another sub-optimal low-complexity operation. As shown in Figs 10 and 11, we assign all vectors in S_i to S_l, $\mathbf{y}_l$ being the nearest codeword to $\mathbf{y}_i$. Afterwards, $\mathbf{y}_l$ is substituted by the centroid $\mathbf{y}'_l$ of the new set. In symbols:

$$\begin{cases} S'_l = S_l \cup S_i; \\ \mathbf{y}'_l = \bar{\mathbf{x}}(S'_l) \end{cases} \tag{17}$$

Generally, this solution is sub-optimal because, in the Voronoi partition, the vectors of S_i could distribute themselves among more cells.

6.3.4 Mean Quantization Error estimation

Now we have to understand if the SoCA produces a lowering of the MQE. If it does, then it is confirmed, i.e. it turns into a SoC. Otherwise the SoCA is rejected.

For an exact evaluation of the MQE, the calculation of the Voronoi partition is necessary. But, as we have already stated, this calculation will introduce a very high overhead. For this reason we employ a sub-optimal, but efficient solution again. It consists in the overestimation of the MQE we would obtain by finding the Voronoi partition. If the overestimated MQE is lower than the previous MQE (i.e the MQE we had before the SoCA) then we are sure that the shift produces an actual decrease in the final MQE. None of these operations require the calculation of the Voronoi partition. Thanks to this trick, the overhead introduced by the ELBG block is negligible in comparison to the time required by the standard LBG.

Focusing our attention only on the old three cells S_i, S_p, S_l, and the three new ones S'_i, S'_p, S'_l, we can understand if the SoCA must be confirmed.

Let us remember that Y and $\mathcal{S}$ are the codebook and the partition before the shift. Y' and $\mathcal{S}'$ are the codebook and the partition we have after the shift. Y' is obtained by replacing in Y the three codewords $\mathbf{y}_i$, $\mathbf{y}_p$, and $\mathbf{y}_l$ with $\mathbf{y}'_i$, $\mathbf{y}'_p$, and $\mathbf{y}'_l$ respectively. In the same way, by substituting the related cells in $\mathcal{S}$, we obtain $\mathcal{S}'$.

The following symbols will be used:

- D_{old} is the MQE before the shift:

$$D_{old} = D(\{Y, \mathcal{S}\}) \tag{18}$$

- D_{new} is the MQE we would have by considering Y' and the Voronoi partition deriving from it :

$$D_{new} = D(\{Y', \mathcal{P}(Y')\}) \tag{19}$$

- d_{old} is the total distortion of the three considered cells before the shift:

$$d_{old} = D_i + D_l + D_p \tag{20}$$

- d_{new} is the total distortion of the three considered cells after the shift:

$$d_{new} = D'_i + D'_l + D'_p \tag{21}$$

If we calculated the Voronoi partition deriving from the new codebook Y', we would understand if the SoCA is useful or not, i.e. if $D_{new} \leq D_{old}$ or not.

We have proved in Appendix that, if we calculate only d_{new} and d_{old} and $d_{new} \leq d_{old}$, then we are sure that $D_{new} \leq D_{old}$. Our condition is only sufficient. For this reason we say that our algorithm is sub-optimal.

6.3.5 Confirmation or discarding of the SoCA

If $d_{new} \leq d_{old}$ the SoCA is confirmed. Therefore, we have a SoC. Otherwise the attempt is discarded. Afterwards, we try to effect another SoCA, i.e. we go back to point 6.3.1.

We can execute any number of consecutive SoC's when there is a decrease in the mean distortion. The final codebook and the related Voronoi partition introduce a mean distortion that is less than that we have obtained before any shift.

This approximation is, in the authors opinion, a good compromise between computational effort and precision. The value of the idea is confirmed by the experiments that we will illustrate in the next sections.

7 Considerations on the utility concept

In this Section we will discuss the utility concept. Starting from a case study we will examine the utility behaviour in the LBG case and in the ELBG.

We took the well-known Lena's image (Munson Jr., 1996) of 512×512 pixels of 256 grey levels. The image was divided into 4×4 blocks and the resulting 16384 16-dimensional vectors were used as a data set. We fixed $N_C = 256$.

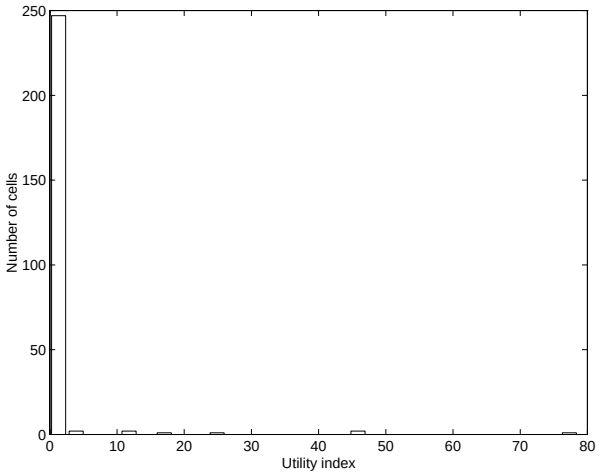

Fig. 12. The initial distribution of the utility indexes

We generated a random initial codebook. In Fig. 12 the very wide and strongly non-symmetrical initial distribution of the utility indexes is shown. The related RMSE is 201.

Successively, we used the standard LBG algorithm. It required 18 iterations. The final RMSE was 33.4. The ELBG with the same initial codebook required only 11 iterations and the final RMSE was 25.8. In both algorithms we fixed $\epsilon = 0.001$.

In Figs. 13 and 14 the final distributions of the utility indexes are shown. In the LBG case we find a lot of codewords with a utility of almost 0 and some with very high utility values, up to $16 \div 18$. In the other case we have a more compact distribution. All utility values are comprised in the real interval $[0, 3]$.

In the following, we will not use the concept of standard deviation because we are dealing with several non symmetrical distributions. We prefer to use the concepts of left and right standard deviations (σ_l, σ_r) (Teseo & Regazzoni, 1996). They are obtained calculating the standard deviation only of the values below and above the mean value respectively.

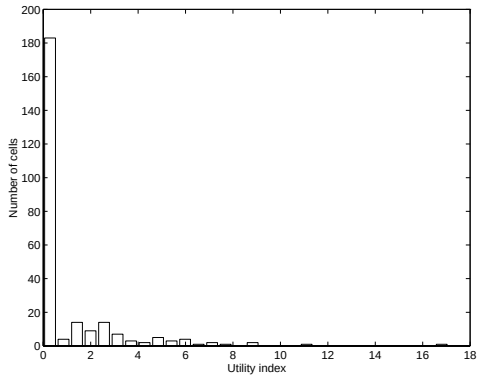

Fig. 13. The final distribution of the utility indexes when the LBG algorithm is used

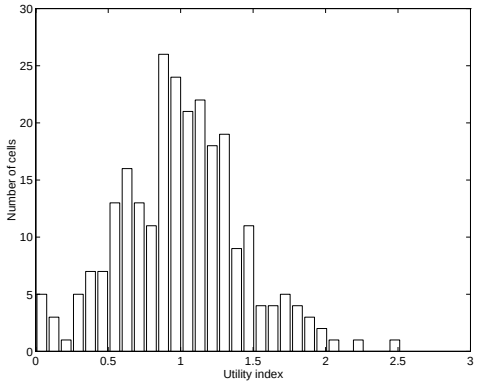

Fig. 14. The final distribution of the utility indexes when the ELBG algorithm is used

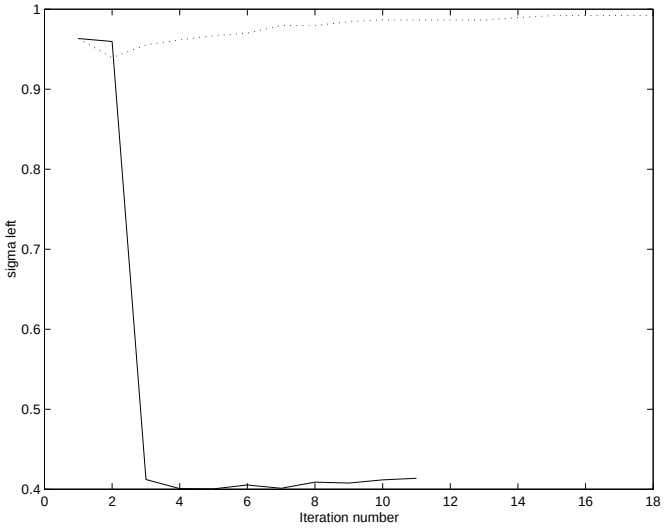

Fig. 15. σ_l versus the number of iterations in the LBG (dotted line) and in the ELBG (solid line) case

Figs. 15 and 16 show σ_l and σ_r versus the number of iterations both in the LBG and in the ELBG case. For the LBG, σ_l is almost constant to 1 for all the iterations whereas σ_r decreases up to 3.72. It reaches about 90% of its total decrease in 4 iterations and slowly continue to decreases up to the final iteration. For the ELBG, σ_l decreases in only 3 iterations to almost its final value of about 0.41. σ_r decreases up to 0.43 in only 3 iterations. So, ELBG effectively succeeds in shifting codewords with very low utility indexes near to codewords with very high utility indexes. Further, it "equalizes" the utility distribution. In fact, we have the total deviation equal to 0.42, i.e. $\sigma \simeq \sigma_l \simeq \sigma_r$.

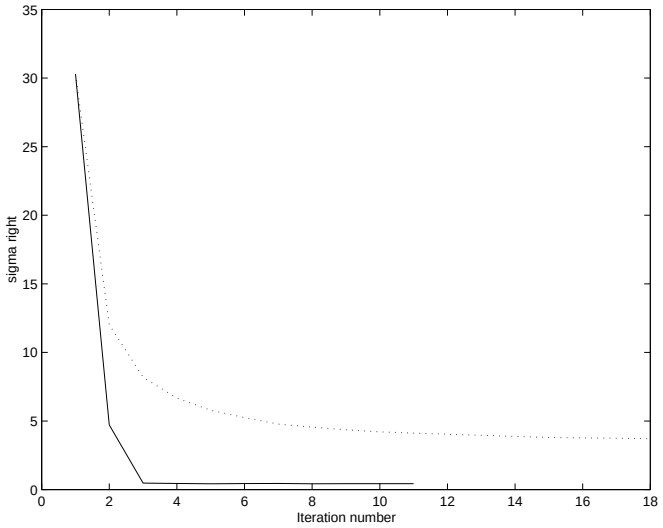

Fig. 16. σ_r versus the number of iterations in the LBG (dotted line) and in the ELBG (solid line) case

8 ELBG overhead estimation

The aim of this Section is to evaluate the ELBG overhead with respect to classical LBG.

We used the same data set of the previous section and, in all tests performed, we fixed $\epsilon = 0.001$. As the performance of our method depends on the number of codewords, we analyzed several cases ranging from $N_C = 128$ to 1024.

For each dimension of the codebook, we randomly generated 15 initial codebooks. Then, for each codebook a LBG and an ELBG quantization were performed. So, all the reported results are the average of the 15 runs. All runs were executed on an Intel Pentium 100MHz based machine and are expressed in seconds.

	LBG	ELBG
128	11.3	12.1
256	22.8	23.7
512	45.3	46.8
1024	91.1	94.3

Table 1
Execution times in seconds per iteration

Table 1 reports the mean time required for LBG and ELBG. This mean does not comprise the initialization phases.

In Fig. 17 we report the confidence intervals of the percentage time increase per

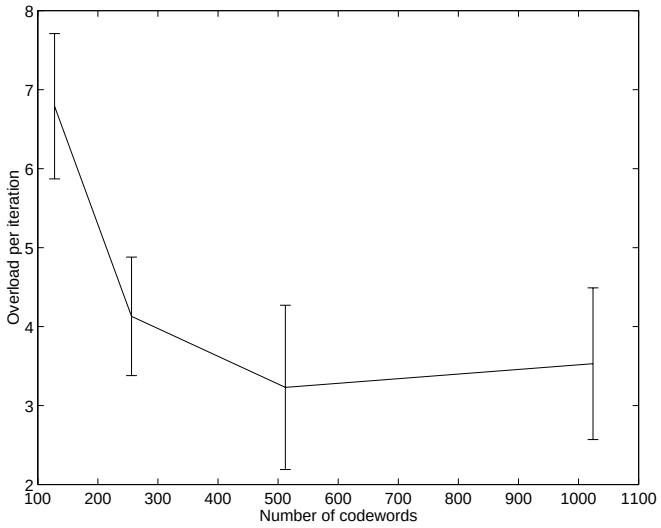

Fig. 17. Confidence intervals of the percentage time increase per iteration with a confidence level of 99%

iteration with a confidence level of 99%. This figure shows that the overhead we have introduced in the LBG algorithm is very low. Further, when the number of codewords increases, the overhead decreases below 5%.

N_C	LBG_{rnd}	LBG_{spl}	ELBG_{rnd}	ELBG_{spl}
128	23.6	13.4	11.4	11.2
256	19.4	12.0	11.0	10.4
512	19.2	10.8	10.8	10.6
1024	19.8	10.0	15.4	11.8

Table 2
Number of required iterations

Table 2 shows the average number (5 runs) of iterations required respectively for the LBG with random initialization (LBG_{rnd}), the LBG with initialization by splitting (LBG_{spl}), the ELBG with random initialization (ELBG_{rnd}) and the ELBG with initialization by splitting (LBG_{spl}). The results reported show that the LBG_{rnd} is the worst of all. Up to a codebook of 512 codewords, both the ELBG_{rnd} and the ELBG_{spl} require less iterations than the LBG_{spl}. The results we obtain for a codebook with a size of 1024 seems to show that the ELBG works worse than the LBG. In effect, more iterations are required. But the reason is that the LBG stops in local minimums. Viceversa, the ELBG succeeds in escaping from these minimums better, and consequently, it requires more iterations.

Fig. 18 shows the result we obtain in the case of a codebook with 1024 codewords. The dashed line refers to the LBG_{spl} whereas the other to the ELBG_{rnd}. After only three iterations the ELBG reaches a RMSE equal to about 20.0.

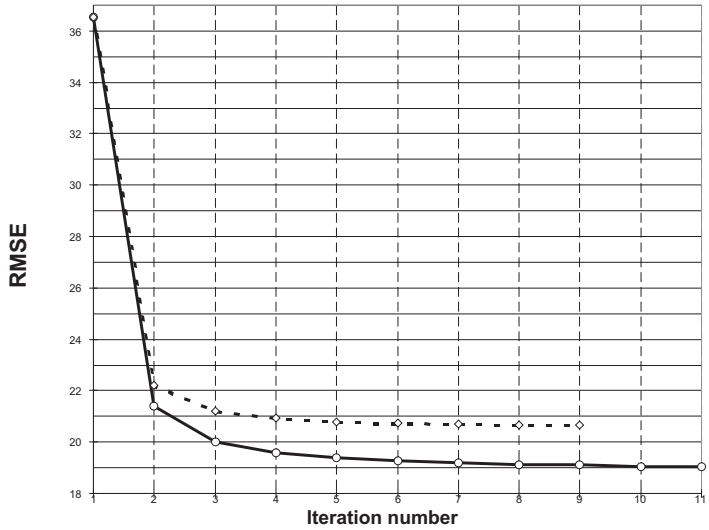

Fig. 18. RMSE versus number of iterations. ELBG_{rnd} (solid line) and LBG_{spl} (dashed line)

When the LBG stops it reaches a RMSE equal to about 20.7. We are using the worst initialization (among the types reported in the paper) for the ELBG and the best one for the LBG. Nevertheless, the ELBG performs better than the LBG. Of course, the LBG_{rnd} performs worse.

9 Results and comparisons

In this Section we will examine the ELBG performance with several application examples ranging from simple bidimensional quantization approaches to complex image compression tasks. We will compare our results with the most recent results we have found in literature. Also in these examples we fixed $\epsilon = 0.001$. Besides, if it is not specified, all of the reported results are the mean values of 5 runs.

9.1 Bidimensional cases

Chinrungrueng and Séquin (1995) propose their optimal adaptive technique and examine several bidimensional cases. We examined two of these.

Polynomial case: as first case study we have taken 2000 patterns as follows:

$$\begin{cases} x_1 \in [-0.5, 0.5] \\ x_2 = 8x_1^3 - 3x_1 \end{cases} \tag{22}$$

where the x_1 values are uniformly spaced in their interval. We fixed $N_C = 16$.

	ELBG_{rnd}	ELBG_{spl}	LBG_{rnd}	LBG_{spl}	C. & S.
Iter.	10.4	10.6	14.8	7.8	
NMSE	1.1E-2	1.1E-2	2.9E-2	1.1E-2	$\sim 1.1E-2$

Table 3
$x_2 = 8x_1^3 - 3x_1$. C. & S. is the abbreviation we used for the technique from Chinrugrueng and Séquin (1995)

We have obtained the results reported in Table 3. In this case the differences between the various methods are marginal because all methods (except the LBG with random initialization), probably, find the global minimum.

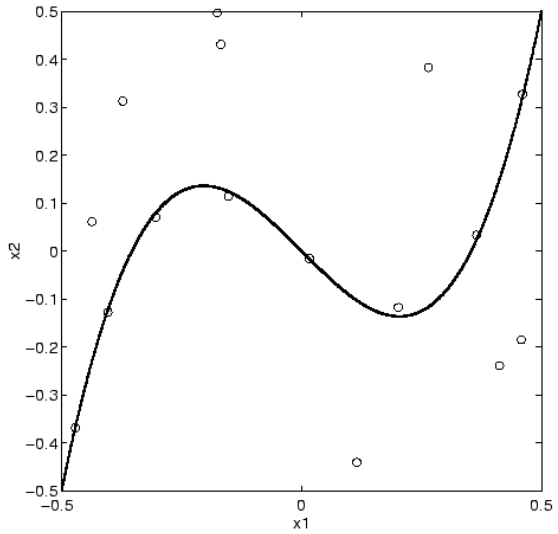

Fig. 19. LBG final distribution

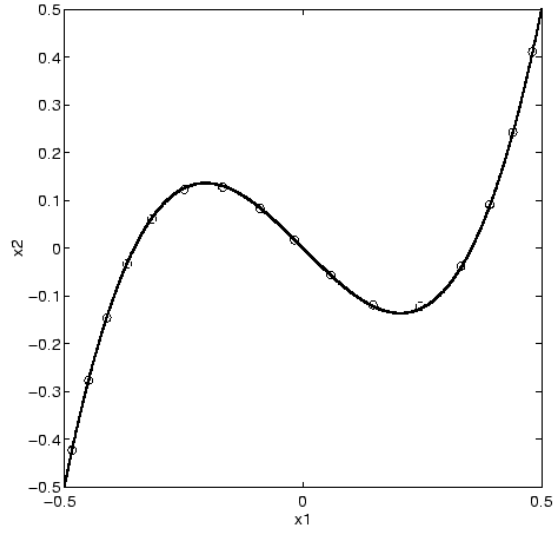

Fig. 20. ELBG final distribution

Figs. 19 and 20 show the results obtained respectively with the LBG and ELBG algorithms starting from the same random initial distribution.

Cantor distribution: as second bidimensional case, we examined the three-level Cantor distribution (Chinrungrueng & Séquin, 1995). Even in this case we considered 2000 patterns and 16 codewords. The results are reported in Table 4. Also in this case the differences between the various methods are marginal because, probably, all methods (except the LBG with random initialization) find the global minimum or a value very near to it.

	ELBG_{rnd}	ELBG_{spl}	LBG_{rnd}	LBG_{spl}	C. & S.
Iter.	5.2	4.6	5.8	4.6	
NMSE	1.2E-2	1.2E-2	3.2E-2	1.2E-2	$\sim 1.2E-2$

Table 4
Cantor's distribution. C. & S. is the abbreviation we used for the technique from Chinrugrueng and Séquin (1995)

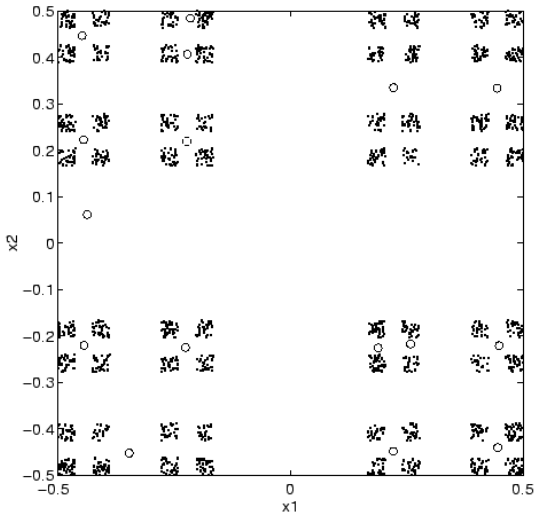

Fig. 21. LBG final distribution

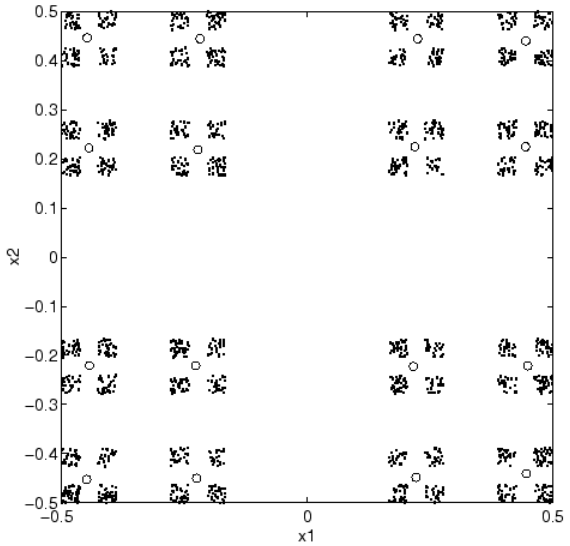

Fig. 22. ELBG final distribution

The graphic results related to the final distributions for a run of the LBG and the ELBG (both initialized with the same randomly generated codebook) are reported in Figs 21 and 22 respectively. In these figures the points represent the data set whereas the circles are the codewords.

Fritzke comparison: Fritzke (1997) used another bidimensional data-set to compare his method, called LBG-U, with the standard LBG. The experiments were done on a set of 500 input patterns [5] generated by a Gaussian mixture distribution. The author performed several runs with N_C ranging from 10 to 100. For all codebook sizes the mean improvement of Fritzke's algorithm was higher than 10% with respect to the LBG. But the LBG-U requires a lot of iterations. This number goes from three to seven times that of the LBG method.

N_C	LBG-U		ELBG	
	RMSE $\pm\sigma$	Iter.	RMSE $\pm\sigma$	Iter.
10	0.0453 ± 12.5%	31.5 ± 42.7%	0.0433 ± 0.20%	7.4 ± 28.0%
100	0.0125 ± 2.1%	57.7 ± 22.1%	0.0123 ± 0.41%	10.6 ± 10.8%

Table 5
ELBG and LBG-U comparison

We have performed the same tests and averaged the results of 10 runs. In Table 5 we have reported our results and the previous ones. The ELBG outperforms LBG-U both as regards the final error and the number of required iterations. We need about 20% of the iterations required with the LBG-U method.

[5] available from ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/data/LBG-U.dat

In image applications, often, the Peak Signal to Noise Ratio (PSNR) is used to evaluate the resulting images after the quantization process. The PSNR is defined as follows:

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{IJ} \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} (f(i,j) - \hat{f}(i,j))^2} \tag{23}$$

where $f(i,j)$ and $\hat{f}(i,j)$ are respectively the grey level of the original image and the reconstructed one. All grey levels are represented with an integer value comprised in $[0, 255]$.

Fig. 23. LBG reconstructed image of Lena

Comparison with previous works: For the first comparison, we use the image of Lena (512×512 pixels) that we introduced in Sec. 7. Figs. 23 and 24 show the encoded images using 32 codewords respectively with LBG and ELBG with random initialization. Comparing these two figures, we find an improvement that we can quantify with a PSNR of 27.7 dB for the LBG and 28.8 dB for the ELBG.

Lee, Baek, and Sung (1997) present an enhanced performance K-means algorithm. They improved both the classical K-means algorithm and Jancey's method (Anderberg, 1973). The modified K-means algorithm was stopped when the MSE was within 0.05% of the previous one. This is equivalent to our $\epsilon = 0.00025$. Table 6 shows previous results and our present results when

Fig. 24. ELBG reconstructed image of Lena

N_C	Modified K-means		ELBG	
	PSNR (dB)	Iter.	PSNR (dB)	Iter.
256	31.92	20	31.94	10.4
512	33.09	17	33.14	10.6
1024	34.42	19	34.59	11.8

Table 6
Lee et al. and ELBG comparison

initialization by splitting is used. These results refer to the same Lena's image as the previous comparison. In this case we improved both the error and the total number of iterations, that is about half.

Karayiannis and Pai (1996) improve Fuzzy Algorithms for Learning Vector Quantization. They used the Lena image of size 256×256 pixels of 8-bit gray values and 512 codewords. As their method depends on several parameters, they executed several runs with different parameter values. Among their results the best one was a PSNR of 32.62 dB. With the same initial hypothesis we obtained 33.04 dB with the random initialization and 33.09 dB with the initialization by splitting.

A study when N_C increases. When N_C increases, the quantization problem becomes more difficult. In fact, more codewords and, consequently, more parameters must be found. We think that a good VQ should work well even in this cases. In Fig. 25 we report the results we obtained. We used Lena's image with 512×512 pixels. The worst performance is obtained by the LBG algorithm both with random initialization and initialization by splitting. The

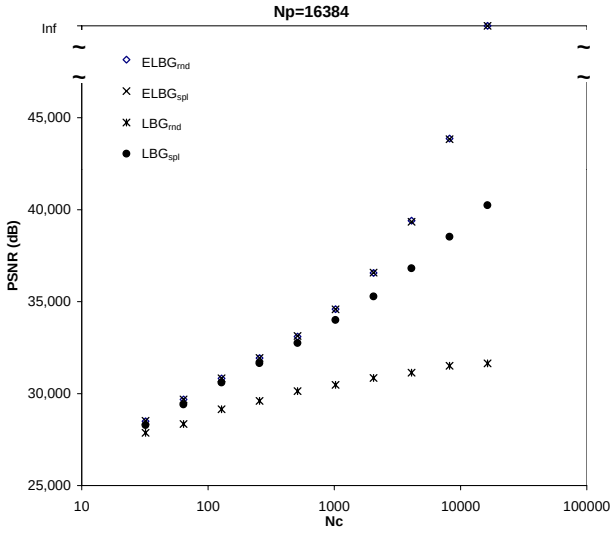

Fig. 25. PSNR in dB versus the size of the codebook

figure clearly shows that LBG with random initialization finds very bad local minimums. And when N_C increases there is very little improvement in the PSNR. With initialization by splitting things go better, but a comparison with the ELBG shows that, above all for N_C greater than 4096, there is a large difference. ELBG succeeds in escaping from bad local minimums. Further, when $N_C = N_P = 16384$, it finds the global optimum. In a few iterations it puts every codeword equal to a different input pattern!

10 Conclusions

In this paper the authors have introduced a new clustering technique they called ELBG. It is based on the concept of utility of a codeword. This new quantity shows very interesting properties. It allows us to understand which codewords are badly positioned and where they should be moved to escape from the proximity of a local minimum in the error function. The analysis of the main properties of the utility index has permitted us to develop an algorithm whose computational complexity is negligible in comparison to the simpler LBG algorithm. This algorithm improves decidedly the performance of the works regarding the most recent advances in clustering tasks. Further, the ELBG shows all its potentiality when the number of codewords increases. As the number of parameters to be found (the components of all codewords) increases, the error function becomes more and more complex and there are plenty of local minimums. So, it becomes very difficult to reach good results. With a significant example we have shown that ELBG works well also in these cases. Our results have highlighted that when the number of codewords increases the ELBG improvement increases, too.

11 Appendix

Let us suppose we have performed a SoCA as explained in sections 6.3.3 and 6.3.4. Let us suppose that all the hypothesis made in these sections are true. We want to demonstrate that:

$$\text{if } d_{new} \leq d_{old}, \text{ then } D_{new} \leq D_{old}$$

Let us indicate $d_{const} = (N_P D_{old} - d_{old})$. The quantity d_{const} is the total distortion derived from the whole codebook and codewords eliminating the codewords and the patterns related to the ith, pth and lth cells. It remains constant from the old codebook to the new one. So:

$D(\{Y', \mathcal{S}'\}) = \frac{1}{N_P}(d_{new} + d_{const}) \leq \frac{1}{N_P}(d_{old} + d_{const}) = D_{old}$.

But, from the NNC, we know that

$D(\{Y', \mathcal{P}(Y')\}) <= D(\{Y', \mathcal{S}'\}) \ \forall \mathcal{S}'$.

So we obtain:

$D_{new} = D(\{Y', \mathcal{P}(Y')\}) \leq D_{old}$

as we wished to demonstrate.

References

Anderberg, M. (1973). *Cluster Analysis for Applications.* New York: Academic.

Bezdek, J., & Pal, N. (1995). Two Soft Relatives of Learning Vector Quantization. *Neural Networks*, *8*(5), 729-743.

Chinrungrueng, C., & Séquin, C. (1995). Optimal adaptive K-Means Algorithm with Dynamic Adjustament of Learning Rate. *IEEE Transaction on Neural Networks*, *6*(1), 157-169.

Cosman, P., Gray, R., & Vetterli, M. (1996). Vector Quantization of Image Subbands: A Survey. *IEEE Transactions on Image Processing*, *5*(2), 202–225.

Fritzke, B. (1997). The LBG-U Method for Vector Quantization – an Improvement Over LBG Inspired from Neural Network. *Neural Processing Letters*, *5*(1), 35–45.

Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition* (Second ed.). 24–28 Oval Road, London NW1 7DX: Academic Press Limited.

Gersho, A. (1979). Asymptotically Optimal Block Quantization. *IEEE Transaction Information Theory*, *IT-25*(4), 373–380.

Gersho, A. (1986). In E.Biglieri & G.Prati (Eds.), *Digital communications.* North-Holland: Elsevier Science Publisher.

Gersho, A., & Gray, R. (1992). *Vector Quantization and Signal Compression.* Boston: Kluwer.

Gonzalez, A., Graña, M., & D'Anjou, A. (1995). An Analysys of the GLVQ Algorithm. *IEEE Transaction on Neural Networks*, *6*(4), 1012–1016.

Hofmann, T., & Buhmann, J. (1997). Pairwise Data Clustering by Deterministic Annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(1), 1–14.

Jolion, J., Meer, P., & Bataouche, S. (1991). Robust Clustering with Applications in Computer Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *13*, 791–802.

Karayiannis, N. (1997). A Methodology for Constructing Fuzzy Algorithms for Learning Vector Quantization. *IEEE Transaction on Neural Networks*, *8*(3), 505–518.

Karayiannis, N., Bezdek, J., Pal, N., Hathaway, R., & Pai, P. (1996). Repairs to GLVQ: A New Family of Competitive Learning Schemes. *IEEE Transaction on Neural Networks*, *7*(5), 1062–1071.

Karayiannis, N., & Pai, P.-I. (1996). Fuzzy Algorithms for Learning Vector Quantization. *IEEE Transaction on Neural Networks*, *7*(5), 1196–1211.

Katsavounidis, I., Kuo, C.-C., & Zhang, Z. (1994). A New Initialization Tecqnique for Generalized Lloyd iteration. *IEEE Signal Processing Letters*, *1*, 144–146.

Kohonen, T. (1989). *Self organization and associative memory* (3rd ed.). Berlin: Springer Verlag.

Lee, D., Baek, S., & Sung, K. (1997). Modified K-means Algorithm for Vector Quantizer Design. *IEEE Signal Processing Letters*, *4*(1), 2–4.

Linde, Y., Buzo, A., & Gray, R. (1980). An Algorithm for Vector Quantizer Design. *IEEE Transaction on Communications*, *28*(1), 84–94.

Lloyd, S. (1957). *Least Squares Quantization in PCM's.* (Bell Telephone Laboratories Paper, Murray Hill)

Lookbaugh, T., Riskin, E., Chou, P., & Gray, R. (1993). Variable Rate VQ for Speech, Image and Video Compression. *IEEE Transaction on Communications*, *41*, 186-199.

Munson Jr., D. (1996). A Note on Lena. *IEEE Transactions on Image Processing*, *5*(1), 3.

Pal, N., Bezdek, J., & Tsao, E. (1993). Generalized Clustering Networks and Kohonen's Self Organizing Scheme. *IEEE Transaction on Neural Networks*, *4*, 549–557.

Paliwal, K., & Atal, B. (1993). Efficient Vector Quantization of LPC Parameters at 24 Bits/Frame. *IEEE Transactions Speech And Audio Processing*, *1*(1), 3–14.

Perlmutter, K., Perlmutter, S., Gray, R., Olshen, R., & Oehler, K. (1996).

Bayes Risk Weighted Vector Quantization with Posterior Estimation for Image Compression and Classification. *IEEE Transactions on Image Processing*, *5*(2), 347–360.

Russo, M. (1998). FuGeNeSys: A Genetic Neural System for Fuzzy Modeling. *IEEE Transactions on Fuzzy Systems*(3), 1–16.

Silva, E. da, Sampson, D., & Ghanbari, M. (1996). A Successive Approximation Vector Quantizer for Wavelet Transform Image Coding. *IEEE Transactions on Image Processing*, *5*(2), 299-310.

Teseo, A., & Regazzoni, C. (1996). Application to Locally Optimum Detection of a New Noise Model. In *Icassp'96* (Vol. 5, pp. 2467–2470). Atlanta, Georgia.