Fully Automatic Clustering System

Giuseppe Patanè and Marco Russo

***Abstract*—In this paper, the fully automatic clustering system (FACS) is presented. It is a technique for clustering and vector quantization whose objective is the automatic calculation of the codebook of the right dimension, the desired error (or target) being fixed. At each iteration, FACS tries to improve the setting of the existing codewords and, if necessary, some elements are removed from or added to the codebook. In order to save on the number of computations per iteration, greedy techniques are adopted. It has been demonstrated, from a heuristic point of view, that the number of the codewords determined by FACS is very low and that the algorithm quickly converges toward the final solution.**

***Index Terms*—Clustering, enhanced Linde–Buzo–Gray (ELBG), fully automatic clustering system (FACS), unsupervised learning (UL), vector quantization (VQ).**

I. Introduction

CLUSTER analysis (CA, or clustering) is an important instrument in engineering and other scientific disciplines. Its applications cover several fields ranging from texture and image segmentation [1], [2], magnetic resonance imaging [3], and computer vision [4] to information retrieval [5] and machine learning [6].

According to Jain *et al.* [7], CA is the organization of a collection of patterns (usually represented as a vector of measurements, or a point in multidimensional space) into clusters based on similarity. Intuitively, patterns within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster. Pattern proximity is usually measured by a distance function defined on pairs of patterns.

In many applications, for example, regarding telecommunications and signal compression, some techniques, based on algorithms for vector quantization (VQ) are often used [8]–[11]. Also in this case, patterns are subdivided into groups (or *cells*), based on similarity measured by a distance function [12]–[16]. Each cell is represented by a vector (called *codeword*) approximating all of its elements. The set of the codewords is called the *codebook*.

In the wide scientific community using techniques for CA and/or VQ, a widespread opinion is that they are essentially different. CA is, generally, conceived as the problem of identifying, with an unsupervised approach, the eventual clusters inside the multidimensional data set to be analyzed [17]–[22]. Differently, a VQ algorithm is not so much interested in finding the clusters, but in representing the data by a reduced number of elements that approximate the original data set as well as possible. The concept is clearer with the help of some figures. Let us consider, for example, the bidimensional patterns in Fig. 1(a). The aim of a CA algorithm is to automatically identify the two clusters present, as shown in Fig. 1(b). A good algorithm can also autonomously recognize the number of the clusters, even if it is not known *a priori* and also in the presence of noise/outlier points [2], [4], [23]–[26]. While, if we apply a VQ technique to the same data set, the number of cells into which the data have to be subdivided depends only on the desired degree of approximation. The higher the number of the codewords (and likewise of the cells), the better the degree of approximation. Four and eight cells (with the related codewords) are highlighted in Fig. 1(c) and (d), respectively. It is evident that, in the latter case, the degree of approximation is better that in the former.

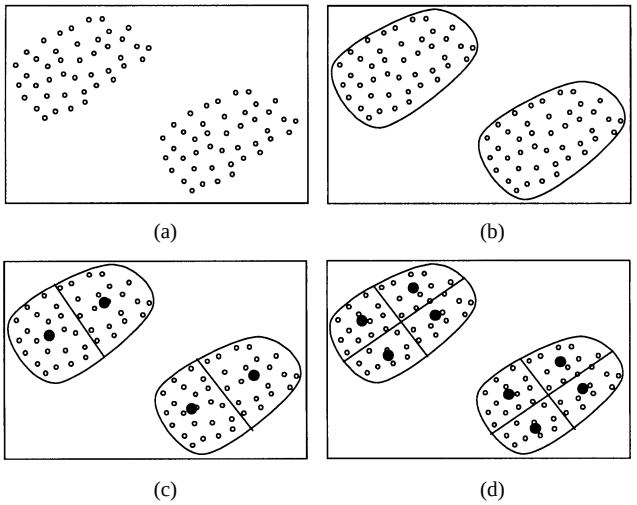

Fig. 1. Clustering and VQ. (a) Original data set. (b) Pair of clusters identified by the CA algorithm. (c) VQ with four codewords. (d) VQ with eight codewords.

In spite of the differences outlined here, it is possible to demonstrate that, in many cases, CA and VQ are, practically, equivalent [27]–[29]. Summarizing, we can say that, often, from an operative point of view, the two approaches roughly execute the same operations: grouping data into a certain number of groups so that a loss (or error) function is minimized.

In literature, several techniques for VQ/CA exist where, as the algorithm develops, not only the values of the parameters to be optimized vary, but also their number. They are used for problems of both supervised learning (SL) and unsupervised learning (UL). Among them, we cite: growing and splitting elastic nets (UL) [30], incremental radial basis functions networks (SL) [31], growing cell structures [(GCS), SL, and UL] [32], growing neural gas [(GNG), SL, and UL]) [33], [34] that take neural gas [35] (UL) as a starting point, fuzzy ARTMAP (SL) [36] that derives from the adaptive resonance theory

Manuscript received July 31, 2000; revised June 27, 2001 and February 27, 2002.

The authors are with the Department of Physics, University of Messina, 98166 Messina, Italy, and also with National Institute for Nuclear Physics (INFN), Section of Catania, 95129 Catania, Italy (e-mail: gpatane@ai.unime.it; marco.russo@ct.infn.it).

Digital Object Identifier 10.1109/TNN.2002.804226

(ART) of Grossberg [37], the fully self-organizing simplified ART (FOSART) [29], [38], and the competitive agglomeration algorithms [2], [25].

In this paper, the fully automatic clustering system (FACS) is presented. It is a VQ/CA iterative algorithm whose aim is, given a data set and a target error e_T, to find a codebook that approximates the input data set with an error less than e_T. The cardinality of the codebook, such as the codewords themselves, is a parameter that the system, in a completely automatic way, identifies during its execution. At each iteration, FACS tries to improve the setting of the existing codewords and, if necessary, some elements are removed from or added to the codebook. In order to save on the number of computations per iteration, *greedy* techniques, i.e., techniques of local updating, are adopted. In this paper it is also demonstrated, from a heuristic point of view, that the number of the codewords is very low and that the algorithm quickly converges toward the final solution.

This paper is organized as follows: in Section II, a brief introduction about VQ is given. Section III summarizes the enhanced Linde–Buzo–Gray (ELBG) algorithm, that is the starting point for FACS. In Section IV, FACS is presented, and its performances are reported in Section V. Conclusions are presented in Section VI.

II. VQ

A. Definition

The objective of VQ is the representation of a set of feature vectors $\mathbf{x} \in X \subseteq \Re^k$ by a set, $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_{N_C}\}$, of N_C reference vectors in $\Re^k$. Y is called *codebook* and its elements *codewords*. The vectors of X are also called *input patterns*, *input vectors* or *input data set*. 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)

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

B. Quantization Error (QE)

The QE is the value assumed by $d(\mathbf{x}, q(\mathbf{x}))$, where d is a generic distance operator for vectors. Several functions can be adopted as distortion measures [12]. In this paper, we will consider two of them: the square error (SE) and the weighted square error (WSE). Their formulations are

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

for the SE and

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{k} w_i (x_i - y_i)^2 \tag{3}$$

for the WSE, where w_i (≥ 0) are the weights for each of the components.

The mean QE (MQE) is used to evaluate the performance of a quantizer. In particular, if the SE is adopted for the distortion measure, the MQE is called mean square error (MSE); if the WSE is used, then the MQE is called weighted mean square error (WMSE). We will also use the square root of the MSE (RMSE).

In general terms, when X is constituted by a finite number (N_P) of elements, the MQE is given by

$$\begin{aligned} \text{MQE} &\equiv D(\{Y, \mathcal{S}\}) \\ &= \frac{1}{N_P} \sum_{p=1}^{N_P} d(\mathbf{x}_p, q(\mathbf{x}_p)) = \frac{1}{N_P} \sum_{i=1}^{N_C} D_i \end{aligned} \tag{4}$$

where we indicate with D_i the ith cell total distortion

$$D_i = \sum_{n:\, \mathbf{x}_n \in S_i} d(\mathbf{x}_n, \mathbf{y}_i). \tag{5}$$

Equations (4) and (5) show that the MQE can be calculated as a function (D) of the codebook (Y) and the partition $(\mathcal{S})$. Now, we report two important conditions we will use throughout this paper that are necessary for calculating the optimal partition (when the codebook is fixed) and the optimal codebook (when the partition is fixed) [12].

Nearest neighbor condition (NNC): Given a fixed codebook Y, the NNC consists in assigning to each input vector the nearest codeword. So, it is possible to divide the input data set in the following manner:

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

The sets $\overline{S}_i$ just defined, constitute a partition of the input data set. This is called the Voronoi partition [13] and is referred to with the symbol $\mathcal{P}(Y) = \{\overline{S}_1, \ldots, \overline{S}_{N_C}\}$. It is possible to demonstrate that the Voronoi partition is optimal [12], i.e., for every partition $\mathcal{S}$ of the input data set, it holds

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

Centroid condition (CC): Given a fixed partition $\mathcal{S}$, the CC concerns the procedure for finding the optimal codebook. This is the codebook constituted by the centroid of each cell [12].

If we consider the set $A \subset \Re^k$ constituted by N_A elements and the SE or the WSE are adopted as measures for distance, its centroid $\overline{\mathbf{x}}(A)$ is

$$\overline{\mathbf{x}}(A) = \frac{1}{N_A} \sum_{\mathbf{x} \in A} \mathbf{x}. \tag{8}$$

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

$$\overline{X}(\mathcal{S}) \equiv \{\overline{\mathbf{x}}(S_i);\ i = 1, \ldots, N_C\} \tag{9}$$

it is optimum [12], i.e., for every codebook Y, it holds

$$D\{Y, \mathcal{S}\} \geq D(\{\overline{X}(\mathcal{S}), \mathcal{S}\}). \tag{10}$$

III. Previous Works: ELBG

The starting point of the research reported in this paper was our previous work: the ELBG [39]. For this reason, here, we briefly describe it. The ELBG has been developed starting from

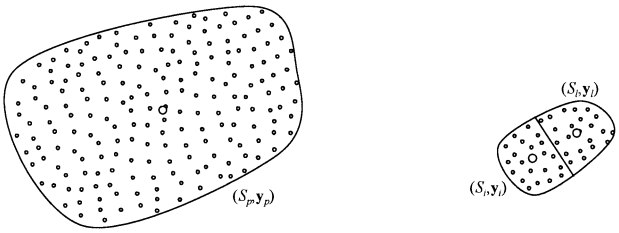

Fig. 2. Situation where the codewords are badly distributed, but locally optimally placed.

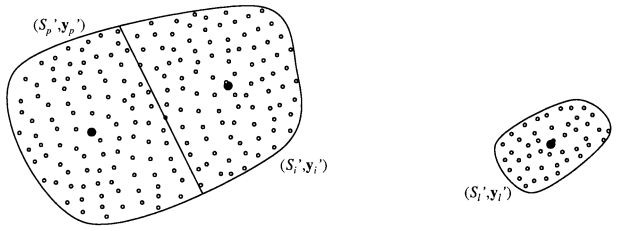

Fig. 3. Better distribution for the codewords than the one shown in Fig. 2.

the original LBG algorithm [12], known also as the generalized Lloyd algorithm [40]. The ELBG is an iterative algorithm, that, N_C being fixed, iteration by iteration, produces a quantizer better than or equal to the one at the previous iteration. The steps through which it develops can be summarized as follows:

Step 1) Initialization. In [39], it was verified with several examples that the ELBG is practically insensitive to the initial choice of the codewords. Therefore, a random initialization of the codebook is, generally, sufficient to start it up.

Step 2) Partition calculation. Given the current codebook, the related Voronoi partition is calculated according to the NNC (6).

Step 3) Termination condition check. The error at the current iteration (D_{curr}) is compared with the one at the previous iteration (D_{prev}). If the ratio $|D_{\text{prev}} - D_{\text{curr}}|/D_{\text{prev}}$ is less than a prefixed threshold (ϵ) then the algorithm ends; otherwise it continues with the next step.

Step 4) ELBG-block execution. Its aim is to escape from the local minima by equalizing the distortions introduced by the cells (D_i); its meaning and the operations executed inside will be summarized later, in Section III-A.

Step 5) New codebook calculation. Given the current partition, the new codebook is calculated according to the CC (9).

Step 6) Return to Step 2.

The steps above are the same as the LBG [12] with the addition of the ELBG-block between the termination condition check and the new codebook calculation. The aim of the ELBG-block is to identify the possible situations of local minima and to remedy them by shifting some codewords in an opportune manner.

In Fig. 2, a typical example of a local minimum is reported. In Fig. 3, a better distribution of the codewords, that arises by executing the shift of a codeword and some local adjustments, is shown. The ELBG-block just attempts to remedy the situation of local minimum by smartly performing several shifts such as the one illustrated in Figs. 2 and 3.

The mathematical justification of such operations comes from one of Gersho's theorems [41]. He explained his partial distortion theorem [42] 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 [42], a high-resolution quantizer has a number of codewords tending to infinite). But, in [43], experimental results proved that it maintains a certain validity also when the codebook has a finite number of elements. So, in [39], [44], the ELBG-block was introduced in order to pursue the equalization of the total distortions of the cells (D_i). The basic idea of the ELBG-block is to go toward the desired equalization by joining a low-distortion cell with a cell adjacent to it. At the same time, a high-distortion cell is split into two smaller ones. So, it is as if the low-distortion codeword is moved inside the high-distortion cell. This is a shift of codeword attempt (SoCA). If this produces a decrease in the MQE, then the SoCA is confirmed, i.e., a shift of codeword (SoC) is executed. Otherwise, the shift is discarded. Several SocAs are executed inside the ELBG block. According to these definitions, Figs. 2 and 3 represent a simple example of a SoCA.

A. ELBG-Block

Now we describe in more detail the operations executed inside the ELBG-block.

Previously, we said that the ELBG-block attempts to obtain the equalization of the distortions by joining low-distortion cells with cells adjacent to them and by splitting high-distortion cells into two. If we define the mean distortion per cell D_{mean} as

$$D_{\text{mean}} = \frac{1}{N_C} \sum_{i=1}^{N_C} D_i \tag{11}$$

we can define as low-distortion the cells with $D_i < D_{\text{mean}}$ and as high-distortion the cells with $D_i > D_{\text{mean}}$.

1) SoCAs: To execute a SoCA three cells are necessary (see, for example, Fig. 2). They are as follows:

- ith cell (S_i): a low-distortion cell ($D_i < D_{\text{mean}}$);
- lth cell (S_l): the cell whose codeword ($\mathbf{y}_l$) has the minimum distance from $\mathbf{y}_i$;
- pth cell (S_p): a high-distortion cell ($D_p > D_{\text{mean}}$).

S_i is searched for in a sequential manner. We mean that, for the first SoCA, we start from the beginning of the codebook and, when we find a cell whose distortion is less than D_{mean}, we choose it. Afterwards, we look for the other two cells (S_l and S_p) that are necessary. At the next SoCA, we continue the search for another cell S_i from the point where we stopped previously, and so on. When we reach the end of the codebook, the termination condition is verified and we try no more SocAs for that iteration of the ELBG.

S_p is looked for in a stochastic way. The method adopted sounds like the roulette wheel selection in genetic algorithms [45]. In practice, we choose a cell with a probability P_p proportional to its distortion value. In mathematical terms

$$P_p = \frac{D_p}{\sum_{h:\, D_h > D_{\text{mean}}} D_h}. \tag{12}$$

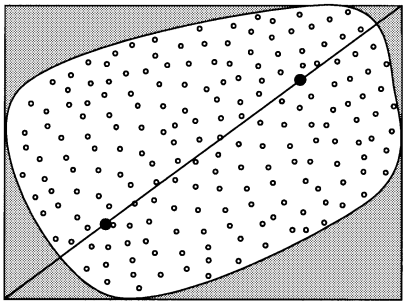

Fig. 4. Hyperbox containing S_p (I_p) and the position of the codewords on the principal diagonal of I_p.

In [39] and [46], the criterion adopted for the selection of cells are described in detail. However, the description given here is enough to explain the remainder of the algorithm.

Figs. 2 and 3 will be used as a simple bidimensional example to explain a SoCA. In Fig. 2, the three cells S_i, S_l, and S_p are represented. We must split the big cell S_p in two smaller ones and join S_i and S_l to form a bigger cell. This is realized by shifting $\mathbf{y}_i$ near $\mathbf{y}_p$ and executing some local rearrangements.

Splitting: We said that we shift $\mathbf{y}_i$ *near* $\mathbf{y}_p$. But, what does *near* mean? We know that a finite number of k-dimensional vectors form the input data set. So, we can say that S_p is contained inside the k-dimensional hyperbox I_p

$$I_p = [x_{1\mathrm{m}}, x_{1\mathrm{M}}] \times [x_{2\mathrm{m}}, x_{2\mathrm{M}}] \times \cdots \times [x_{k\mathrm{m}}, x_{k\mathrm{M}}] \quad (13)$$

where $x_{h\mathrm{m}}$ and $x_{h\mathrm{M}}$ are, respectively, the minimum and maximum values assumed by the hth dimension of all the patterns belonging to S_p. From this consideration, we place both $\mathbf{y}_i$ and $\mathbf{y}_p$ on the principal diagonal of I_p; in this sense, we can say that the two codewords are *near* each other.

Codewords are set in such a way that the diagonal is subdivided into three parts; the central part is twice the length of the ends, as illustrated in Fig. 4. The situation of Fig. 4 can be easily generalized to a k-dimensional problem. Afterwards, a rearrangement of $\mathbf{y}_i$ and $\mathbf{y}_p$ is executed by means of a local LBG where only the patterns belonging to the old S_p constitute the input data set and the two codewords on the principal diagonal are the initial codebook. By choosing a high value for ϵ (typically $0.1 \div 0.3$), in a few iterations (one or two) the local LBG ends. In Fig. 3, we can see the result of this operation where the two new codewords ($\mathbf{y}'_i$, $\mathbf{y}'_p$) and the two new cells (S'_i, S'_p) are reported.

Union: After the codeword $\mathbf{y}_i$ has been moved away, we add all of the patterns belonging to the old cell S_i to S_l and we place $\mathbf{y}_l$ in the centroid of the cell so obtained. The result of this operation is reported in Fig. 3. In symbols

$$\begin{cases} S'_l = S_l \cup S_i \\ \mathbf{y}'_l = \overline{\mathbf{x}}(S'_l). \end{cases} \quad (14)$$

2) Mean Quantization Error Estimation and Eventual SoC: After the shift, we have a new codebook (Y') and a new partition ($\mathcal{S}'$). Therefore, by applying (4), we can calculate the new MQE. If it is lower than the value we had before the SoCA, this is confirmed, i.e., it turns into a SoC. Otherwise, it is rejected. As only three cells and the related patterns are involved in the SoCA, we can execute the new MQE calculation considering only the three old cells (S_i, S_p, S_l), and the three new ones (S'_i, S'_p, S'_l).

B. Considerations Regarding the ELBG

We have seen that a SoC consists of the combined execution of two operations: the splitting of a cell and the union of two other cells. Thus, N_C remains unchanged because the codeword that is eliminated from one place is inserted into another. For this reason, we use the word *shifting* to describe the whole operation. Instead, if we execute only the part related to splitting or only that related to union, the number of codewords will increase or decrease, respectively. In this way, we can insert or delete some codewords from a codebook. Such operations can be considered, in first approximation, *fast* and *intelligent* because:

1) insertions are effected in the regions where the error is higher and deletions where the error is lower;
2) operations are executed locally, i.e., without global reorganizations of the codebook and the partition (see Section III-A1);
3) several insertions or deletions can be effected during the same iteration always working locally.

Insertions and deletions of codewords effected by FACS are realized working in this way, as will be explained in detail in Section IV.

IV. FACS

A. Introduction

In this section, we will describe how FACS works. As we have previously mentioned, it is a CA/VQ technique whose objective is to automatically find the codebook of the right dimension, when the input data set, the distortion measure and the desired error (or target, e_T) are fixed. It develops through a sequence of iterations like the ELBG, that is taken as a starting point. But, unlike the ELBG, FACS, evaluates the error at the end of each iteration and, according to whether it is above or below the target, decides to make another iteration with the same number of codewords or if it is necessary to increase or decrease N_C. Such an increase or decrease happens *smartly*, trying to insert new codewords where the quantization error is higher and to eliminate them where the error is lower. A similar strategy is proposed by Fritzke in [30] and [32] for his competitive-learning algorithms, while FACS is a K-means type algorithm [39], [47]. Besides, in FACS, insertions and deletions of codewords are regulated by a stochastic process; in [30] and [32] they occur deterministically. Particular attention is paid so that the technique employed allows the convergence of the algorithm toward a good solution in a few iterations. The results presented in Section V will show that a number of iterations comparable with the ones required by the ELBG are enough. In [39], [44], and [46], the high speed of convergence of the ELBG has already been highlighted.

B. General Description

Each of the iterations through which FACS develops (FACS-iterations) can be summarized as in Fig. 5. It can be divided into two parts: in the first one the same operations that have been described in Section IV-A for the ELBG are executed.

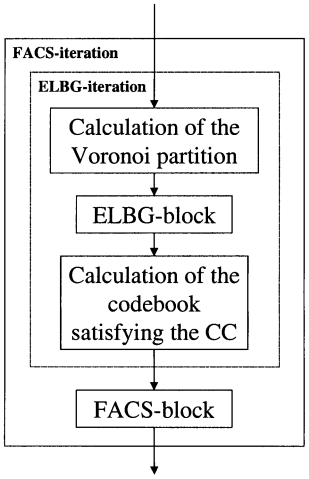

Fig. 5. FACS iteration.

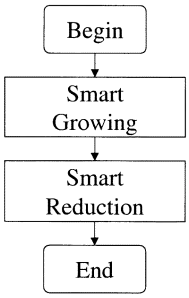

Fig. 6. Two phases through which FACS develops.

They are: the Voronoi partition calculation, the ELBG-block execution and the calculation of the codebook satisfying the CC. For this reason, such a sequence of operations was grouped into a single block that we called ELBG-iteration. An FACS iteration is completed by the execution of the last block (the FACS-block) whose functionality is to *smartly* modify, if necessary, the number of the codewords.

As illustrated in Fig. 6, we can distinguish two phases during the execution of FACS. Each of them is constituted by a sequence of FACS-iterations like the ones in Fig. 5. For each FACS-iteration, the ELBG-iteration executes the same operations for both of the phases, while the operations executed inside the FACS-block are different.

The first phase, called *smart growing*, consists of a certain number of FACS-iterations, during which the number of the codewords is gradually increased until the error barely goes below e_T. Such an increase happens by splitting, one at a time, some cells, chosen among the ones with a total distortion greater than D_{mean}. So, codewords are inserted where the distortion is higher.

During the second phase, called *smart reduction*, a certain number of FACS-iterations are executed during which the number of codewords is decreased. More precisely, if during the first phase of an iteration (the ELBG iteration) the error goes below e_T, as many codewords as are necessary to obtain $D > e_T$ are deleted. A codeword deletion is realized by choosing a cell whose total distortion is less than D_{mean} and by joining it to the nearest cell.

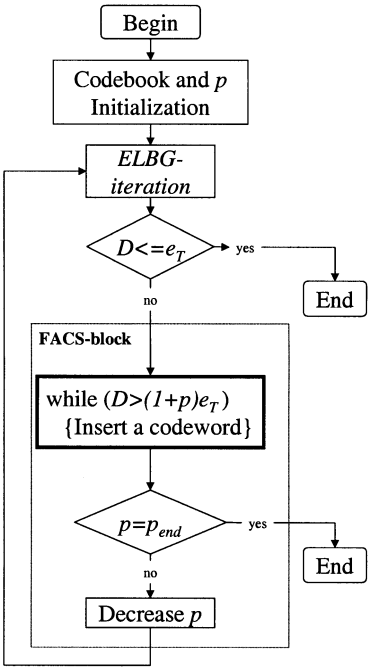

Fig. 7. *Smart growing* phase.

After an insertion or deletion occurs, we can calculate the new distortion by modifying, in (4), only the contributions of the cells (D_i) involved in the splitting or in the union. So, we are able to immediately evaluate if other insertions or deletions are necessary without recalculating the Voronoi partition, therefore very quickly. The whole procedure for the selection and the insertion or deletion of the cells will be described later.

Now, let us describe the two phases through which the algorithm develops, as we have shown in Fig. 6.

C. Smart Growing

The initialization of the codebook could be executed starting from a single codeword in the centroid of the whole input data set and inserting new ones until the error is below e_T. Afterwards, during the next iterations, codewords in excess would be removed. However, we realized, experimentally, that a better result is obtained when the insertion of the codewords is effected more gradually. By the term better result we mean that, under the same value of e_T, final codebooks with lower values of N_C are obtained. The phenomenon was more evident when the complexity of the problem increased.

In Fig. 7, the whole phase of the *smart growing* is illustrated. The growing happens by inserting, each time, as many codewords as are necessary for the error to be equal to or less than $(1+p)e_T$, where $p \geq 0$ and is monotonically not increasing from p_{ini} to p_{end} ($p_{\text{end}} = 0$). In the same picture, the FACS-block and the operations executed by it when we are in the *smart growing* phase are highlighted. Inside the FACS-block, the block dealing with the insertion of the codewords is further highlighted. A more detailed description of its function will be given later in this sction. Once the law regulating the decrease of p with the iteration number (n) has been fixed, we can summarize the *smart growing* as follows.

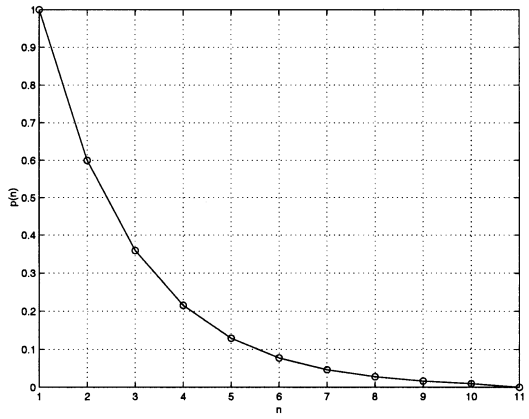

Fig. 8. p versus the number of iterations.

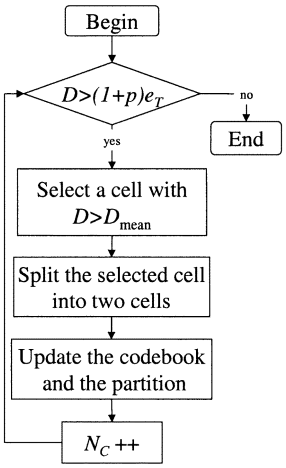

Fig. 9. Detailed description of the insertion of the codewords.

1) $N_C = 1$ and $n = 1$ being fixed, place the first codeword in the centroid of the whole input data set.
2) Insert as many codewords as are necessary to obtain $D \leq (1 + p(n))e_T$.
3) If ($p(n) = 0$) then the *smart growing* ends.
4) Execute an *ELBG-iteration*.
5) If ($D \leq e_T$) then the *smart growing* ends.
6) $n++$.
7) Return to point 2.

We verified experimentally that it is better to select a high initial value (about one) for p and make it decrease quickly iteration by iteration. For this reason, we chose to make p decrease from p_{ini} to $p_{\text{end}} = 0$ in n_I iterations and that, for the first $n_I - 1$ iterations, it follows an exponential law, i.e.,

$$p(n) = \begin{cases} \alpha e^{-\beta n}, & \text{for } n = 1, 2, \ldots, n_I - 1 \\ 0, & \text{for } n = n_I \end{cases} \tag{15}$$

where α and β are positive constant values.

Still experimentally, we saw that about ten iterations are enough to obtain good results. In Fig. 8, $p(n)$ is reported when $n_I = 11$ and α and β were fixed so that $p(1) = 1.0$ and $p(n_I - 1) = 0.01$. Such values were used for all of the tests we effected with FACS.

Fig. 9 details the operations that are executed by the block regulating the insertion of the codewords inside the FACS-block and highlighted in Fig. 7 for the *smart growing* phase. If the quantization error is above the desired threshold $[(1+p)e_T]$, the FACS-block inserts the number of codewords necessary to take it just below. This happens by executing several times the procedure of the splitting that was explained in Section III-A1 for a SoCA. For a better understanding of the operations reported in Fig. 9, we must keep in mind the following.

- The selection of the cell to split and of the related codeword $(S_i, \mathbf{y}_i)$ is made among the ones whose distortion is greater than D_{mean} with the roulette wheel method [see (12)].
- The splitting of $(S_i, \mathbf{y}_i)$ in $(S_i', \mathbf{y}_i')$ and $(S_i'', \mathbf{y}_i'')$ happens as is explained in Section III-A1.
- The update of the partition and of the codebook consists of the substitution of $(S_i, \mathbf{y}_i)$ with $(S_i', \mathbf{y}_i')$ and $(S_i'', \mathbf{y}_i'')$.

1) Discussion About the Law Regulating the Decrease of the Target Error: The employment of the greedy strategy described above for the insertion of the codewords, allows an enormous saving on the computation. However, it is a nonoptimal solution because not all of the elements of the codebook and of the data set are considered. For this reason, each iteration includes, as well as a number of local updates, also a global rearrangement. The nonoptimal effect of the local updates is more evident particularly in the early iterations when the codebook is very disorganized. In that case, it is necessary to insert a high number of codewords in order to ensure the desired target. But, if the initial target is higher than the one specified by the user and, gradually, approaches it, during the early iterations a lower number of codewords is inserted. Iteration by iteration, because of the global optimization, the codewords distribute themselves better and better; so, the new insertions can occur more exactly.

Besides, also the following phase, i.e., the *smart reduction*, benefits from such a way of operating because it will have to deal with the removal of a lower number of exceeding codewords.

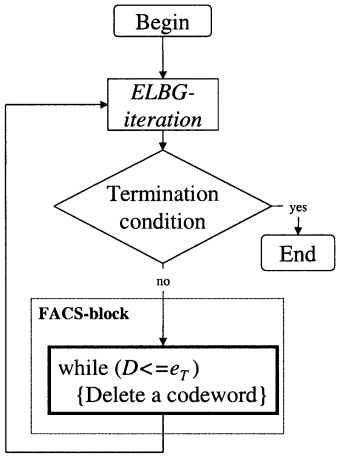

Fig. 10. *Smart reduction* phase.

D. Smart Reduction

The whole phase of the *smart reduction* can be summarized as in Fig. 10. The number of the codewords is gradually decreased as soon as, at the end of an iteration, the error is equal to or

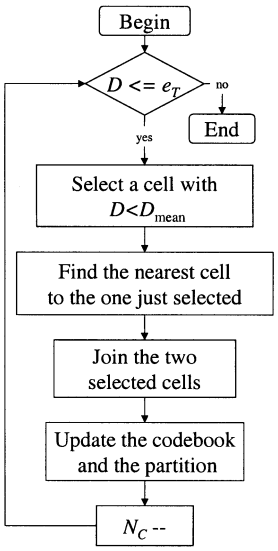

Fig. 11. Detailed description of the deletion of the codewords.

less than the target. Instead, if the error is above the target we continue with the same resolution, i.e., with the same number of codewords. In Fig. 10, the FACS-block and the operations it executes during the *smart reduction* phase are highlighted. A more detailed description of it is reported in Fig. 11. If the quantization error is equal to or less than the desired value (e_T), the FACS-block deletes the number of codewords necessary to take it just above. The elimination happens in a *smart* way because we try to remove the codewords that contribute the least to the total distortion.

To understand better the operations reported in Fig. 11, we must keep in mind the following.

- The cell to eliminate and the related codeword $(S_i, \mathbf{y}_i)$ are selected among the ones whose distortion is less than D_{mean} with a probabilistic method analogous to the one expressed by (12). Here, the cell to eliminate is chosen with a probability that is a decreasing function of its distortion. In mathematical terms

$$P_p = \frac{D_{\text{mean}} - D_p}{\sum_{h:\, D_h < D_{\text{mean}}} (D_{\text{mean}} - D_h)}. \tag{16}$$

- The union of $(S_i, \mathbf{y}_i)$ and $(S_l, \mathbf{y}_l)$ to form $(S'_l, \mathbf{y}'_l)$ is effected as explained in Section III-A1.
- The update of the partition and the codebook consist of the substitution of $(S_i, \mathbf{y}_i)$ and $(S_l, \mathbf{y}_l)$ with $(S'_l, \mathbf{y}'_l)$.

The block related to the termination condition will be explained later.

E. Behavior of FACS Versus the Number of Iterations and Termination Condition

The algorithm was developed so that, during the *smart deletion*, the error is always near e_T. This happens thanks to the continuous adjustments of N_C.

In Fig. 12, we report the typical trend of the error and the number of codewords FACS works with, versus the number of iterations. The graph refers to an image compression task, whose details, that are not, at present, important for understanding the picture, are given in the section related to the comparisons.

For a better graphic visualization, we chose to report the normalized values of the variables in question. In particular, N'_C is N_C normalized with respect to the value found by FACS for that run after 200 iterations; the RMSE (D) is normalized with respect to the target (e_T). We can immediately see that, after the insertion of the codewords ends, the algorithm always keeps the error very close to e_T. In particular, we can notice the correlation between the two curves represented. When the error is greater than e_T, N_C is kept constant. As soon as D goes below e_T, N_C is automatically decreased until the error is above e_T.

The graph just examined suggests two criteria to employ as termination conditions for FACS. Both of them are to be adopted when the *smart growing* phase has ended.

- The algorithm ends after a certain number of prefixed iterations.
- The algorithm ends when a certain number of consecutive iterations with the same value of N_C have been executed.

If we choose one of them as the termination condition, it is possible that, when FACS ends, the quantizer obtained, with N_C codewords, generates an error greater than e_T. Such an error, considering how the algorithm develops, is very close to e_T. So, according to its value, it could be considered acceptable. However, even if it cannot be considered acceptable, it is sufficient to remember that the quantizer with N_C codewords was obtained by eliminating a codeword from the one with $N_C + 1$ codewords, that produced an error less than e_T. Therefore, if we keep in memory the last codebook that was able to ensure an error less than e_T, we can stop the algorithm at any moment and have a codebook satisfying the desired specifications.

F. Discussion About Outliers

We wish to underline again that FACS has been conceived with the aim of autonomously calculating an opportune codebook having, as its only requirement, the satisfying of a target error (specified by the user) with the least number of codewords. However, the presence of outliers may degrade the results obtained by FACS. For example, let us consider Fig. 13(a). There, we can locate two clusters of points and one outlier point rather "far" from the clusters. Let us suppose FACS is launched and that, for a certain value of e_T, it finds three codewords, one in the center of each cluster and one coinciding with the outlier, as in Fig. 13(b). According to (4), the MQE is given by

$$\text{MQE} = \frac{D_1 + D_2 + D_3}{N_P}. \tag{17}$$

Given that the outlier is exactly represented by codeword number 3, we have $D_3 = 0$. Now, if we calculate the MQE only on the "clean" part of the data set, i.e., excluding the outlier, it is

$$\text{MQE}_{\text{clean}} = \frac{D_1 + D_2}{N_P - 1}. \tag{18}$$

Obviously, $\text{MQE}_{\text{clean}} \geq \text{MQE}$. This implies that the presence of outliers can also heavily affect the final result obtained by FACS. This is not a drawback of FACS, but it derives from the nature of the algorithm itself.

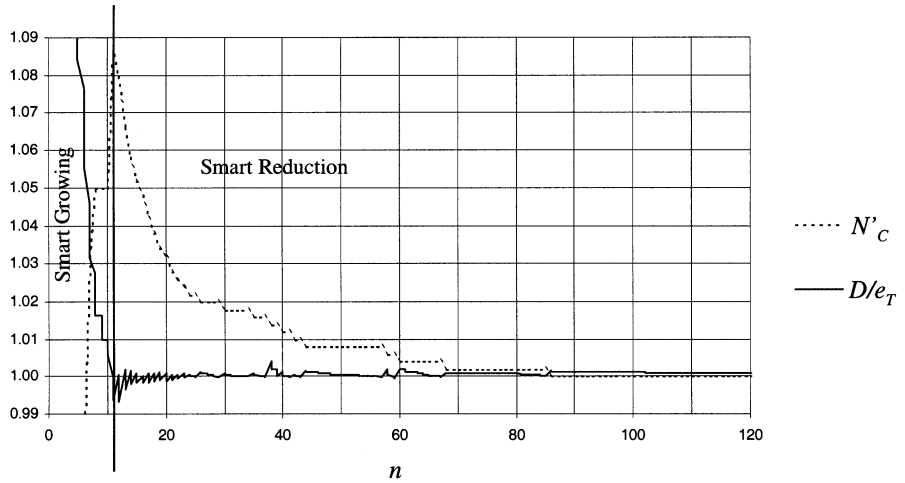

Fig. 12. Typical trend of N'_C (it is N_C normalized with respect to its value after 200 iterations) and D (normalized with respect to e_T) versus the number of iterations.

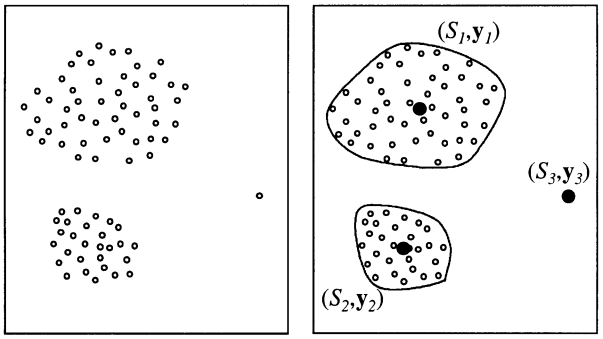

Fig. 13. Dataset with two clusters and one outlier.

However, there are applications where the identification of the outliers is essential in order to avoid their influence on the final result. In such cases, FACS can still be used, provided a noise category removal mechanism [38] is added for treating the outliers. Even if it is outside the scope of this paper, we wish to cite some methods commonly adopted in literature that, opportunely readapted, could represent the desired modifications. For example, a simple mechanism lets the algorithm run with the desired value of e_T until it ends. Afterwards, the outliers are identified (for example, the patterns constituting a cell with a single pattern), and removed (together with the related codewords). Therefore, the algorithm can go on with the "clean" data set so obtained. More advanced mechanisms could provide for the removal of patterns constituting cells whose cardinality is less than a prefixed threshold (for example, a certain quota of the total number of patterns).

V. Results

A. Introduction

In this section, we present the results obtained by FACS and compare it with both algorithms where the codebook has a fixed size, and techniques where the dimension of the codebook is not known.

In order to perform the comparisons, both in terms of final result and speed of convergence, we looked for similar techniques already documented in the literature. We found several algorithms that, as underlined by their authors, can generate a codebook of opportune dimension that is able to ensure the desired level of performance [30]–[33]. However, we noticed they have been used either as traditional algorithms for VQ/CA [30], where N_C is fixed, or as algorithms for supervised learning and classification [32]. For some of them, it is explicitly said that they can be used to generate a codebook of the opportune size once the desired error has been fixed [31], [33]. Unfortunately, we did not find any example where they were employed in this way. Besides, also when they were used as traditional VQ/CA algorithms, we found no numerical results directly comparable with FACS. For example, in [30], Fritzke reports the results related to an image compression task for its growing and splitting elastic net. In particular, he uses the image of Lena [48] of 480 $\times$ 480 pixels at 256 levels of gray. But, his results also consider an intermediate processing of the data by means of a multilayer perceptron; so, they are not directly comparable to the ones obtained by FACS.

However, comparisons with some of these algorithms ([33] and GNG-U [49]) have been performed following the suggestion of the author, i.e., making the number of codewords "grow" until the desired target is reached. Besides, the performances of FACS can be compared with the ones of a traditional algorithm for VQ/CA where N_C is a datum. In fact, it is possible to fix a value of N_C and see what error the VQ algorithm obtains. Afterwards, we can choose that error as the target for FACS and see how many codewords are necessary to obtain that performance. The comparison has been executed with the ELBG, that has obtained performances better than or equal to the ones of the previous VQ algorithms in [15], [16], [43], [50].

Several more algorithms exist in the literature, where the number of the codewords is not a datum but one of the results. Among them, we have chosen to make comparisons with FOSART, an algorithm in the family of ART [29], [38], and the competitive agglomeration algorithm [25].

Afterwards, we have tried using FACS for one of the classification problems reported in [32].

All of the tests have been effected on machines equipped with Intel Celeron 366-MHz processors and running the LINUX operating system. The related details will be given later.

TABLE I
ELBG PERFORMANCE

N_{it}	$N_C = 128$		$N_C = 512$		$N_C = 4096$	
	RMSE	$t_{it}(s)$	RMSE	$t_{it}(s)$	RMSE	$t_{it}(s)$
15	29.26 ± 0.06	2.38 ± 0.04	22.39 ± 0.03	8.76 ± 0.19	10.81 ± 0.02	106.57 ± 0.56
∞	29.20 ± 0.07	2.25 ± 0.03	22.33 ± 0.03	8.67 ± 0.04	10.74 ± 0.02	106.75 ± 0.56

As we have said in Section IV-F, FACS has been conceived with the aim of autonomously calculating an opportune codebook having, as its only requirement, the satisfying of e_T with the least number of codewords. We are aware that some of the algorithms we used for our comparisons were designed for solving different problems. Thus, a direct comparison with FACS may appear rather forced. However, the techniques considered are the works in literature that, to the best of our knowledge, can be considered more similar to ours.

B. Comparison With ELBG

In this section, we evaluate the performances of FACS for a typical task of VQ: image compression. For our tests, we chose the image of Lena of 512×512 of 256 gray levels. It was subdivided into 16 384 blocks of 4×4 pixels and the related 16-dimensional vectors were used as learning patterns.

The procedure employed to effect the comparisons develops through the following points.

1) A value for N_C being fixed, the ELBG is executed and its results, in terms of RMSE, are collected.
2) FACS is executed using the value of the RMSE obtained by the ELBG as target.
3) The number of codewords found by FACS is compared with N_C.

The comparison was repeated for several values of N_C. As we will see, FACS obtains, practically, the same number of codewords the ELBG had been launched with. Besides, we will see that it is possible to fix a relatively low number of iterations (we have chosen 15) and obtain results nearly equal to the ones we would obtain by making FACS run for a much longer period. In the remainder of this paper, we describe in detail the procedure we have followed.

The ELBG was launched for $N_C = 128, 512, 4096$ and we chose a very low value for ϵ (0.000 000 01). In this way, the algorithm ends when, practically, it reaches the minimum for that run.

In Table I, the results obtained are reported. All of the values are the mean calculated on ten runs. For each value of N_C, we report the results obtained after 15 iterations and the ones obtained after the algorithm ends (∞ iterations). Each result is reported together with the variance calculated on the runs effected.

Before we proceed to the comparisons with the ELBG, some considerations regarding the calculation times are necessary. As we saw in [39], the time required per iteration, once the input set has been fixed, increases almost linearly when N_C increases, too. As we can see from Table I, this is, more or less, valid when passing from $N_C = 128$ to $N_C = 512$, while this is not true when passing from $N_C = 512$ to $N_C = 4096$. Such behavior occurs because, in such a complicated task (the ratio N_P/N_C is four), about half of the patterns are involved in SoCAs during the execution of the ELBG-block. Instead, in the other cases considered, the contribution of the ELBG-block is lower and is in agreement with the rate (about 5%) that we indicated in [39] and the time per iteration increases almost linearly when N_C increases, the other parameters being fixed.

TABLE II
FACS PERFORMANCES

e_T		$N_{it} = 15$	$N_{it} = 200$
$\mathrm{RMSE}_{15}(128)$	N_C	128.4 ± 0.8	124.9 ± 0.7
$=$	$\frac{N_C}{128}$	1.003 ± 0.006	0.976 ± 0.005
29.26	N'_C	1.028	1
	$t_{it}(s)$	2.16 ± 0.03	2.14 ± 0.03
$\mathrm{RMSE}_{\infty}(128)$	N_C	130.8 ± 1.6	126.4 ± 1.2
$=$	$\frac{N_C}{128}$	1.022 ± 0.013	0.988 ± 0.009
29.20	N'_C	1.035	1
	$t_{it}(s)$	2.21 ± 0.04	2.17 ± 0.03
$\mathrm{RMSE}_{15}(512)$	N_C	517.4 ± 3.0	496.7 ± 2.2
$=$	$\frac{N_C}{512}$	1.011 ± 0.023	0.970 ± 0.017
22.39	N'_C	1.042	1
	$t_{it}(s)$	7.83 ± 0.05	8.29 ± 0.03
$\mathrm{RMSE}_{\infty}(512)$	N_C	527.3 ± 4.3	504.2 ± 4.9
$=$	$\frac{N_C}{512}$	1.030 ± 0.008	0.985 ± 0.010
22.33	N'_C	1.046	1
	$t_{it}(s)$	8.00 ± 0.16	8.41 ± 0.08
$\mathrm{RMSE}_{15}(4096)$	N_C	4143.5 ± 16.7	4074.8 ± 16.1
$=$	$\frac{N_C}{4096}$	1.012 ± 0.004	0.995 ± 0.004
10.81	N'_C	1.017	1
	$t_{it}(s)$	100.23 ± 1.38	103.95 ± 1.82
$\mathrm{RMSE}_{\infty}(4096)$	N_C	4188.6 ± 14.9	4118.9 ± 17.3
$=$	$\frac{N_C}{4096}$	1.023 ± 0.004	1.006 ± 0.004
10.74	N'_C	1.017	1
	$t_{it}(s)$	102.39 ± 1.25	105.67 ± 1.29

Afterwards, we launched FACS with e_T set to the RMSE found by the ELBG after 15 iterations ($\mathrm{RMSE}_{15}(N_C)$) and after infinite iterations ($\mathrm{RMSE}_{\infty}(N_C)$) (for the values of N_C considered) as targets. In Table II, the values obtained by FACS after

15 and after 200 iterations are reported. As we can see, FACS obtains codebooks with, practically, the same number of codewords used by the ELBG to achieve that error; the difference is inside 3% in all of the considered cases. Besides, the values of N'_C (the number of codewords obtained by FACS normalized with respect to the value found after 200 iterations) show that, after 15 iterations, FACS obtains a number of codewords that is very close to the one it will obtain after 200 iterations; the difference is below 5% in all of the considered cases. Also the mean time per iteration for ELBG and FACS are comparable.

C. Comparison With GNG and GNG-U

Here, a comparison with GNG and its variant (GNG-U), both from Fritzke [33], [49] is reported. They, gradually, insert codewords until the prefixed number or until a certain "performance measure" is fulfilled. In our case, the performance measure to be considered is the achievement of e_T. The error is calculated at the end of each epoch[1] (or iteration) and, like in FACS, according to whether it is above or below e_T, a decision about the insertion of more codewords is taken.

All of the tests described here have been executed using the image of Lena previously mentioned. Also the simbology is the same as the one used in the previous comparison.

For the execution of GNG, as well as the desired e_T, it is necessary to specify several configuration parameters. In order to choose such values, we have referred to the works from Fritzke where he presented his algorithms [33], [49], [51]. The values that we extracted from such papers follows:

- $\epsilon_b = 0.05, 0.2$;
- $\epsilon_n = 0.006, 0.0006$;
- $\alpha = 0.5$;
- $\beta = 0.005, 0.0005$;
- $\lambda = 100, 300, 500$;
- $a_{\max} = 50, 88, 120$.

We can see that, for α, a single value has been used in all of the examples reported. On the contrary, different values in different examples have been used for the other parameters. So, we have realized several tests for choosing their best combination; we fixed $e_T = \mathrm{RMSE}_{15}(128)$ and tried all of the possible configurations. In the end, we have verified that, for this problem, the best results were obtained by choosing $\epsilon_b = 0.2$, $\epsilon_n = 0.0006$, $\beta = 0.0005$, $\lambda = 500$, $a_{\max} = 50$. We have also executed some tests with GNG-U, but, if using, for this problem, the same values of the parameters used by Fritzke, GNG-U performs worse than GNG. According to such considerations, we have completed our tests with GNG, changing the final RMSE and using the values determined above. The results obtained are summarized in Table III. In Table III, two series of values are reported: the former (labeled $Ep_{\min}$) refers to the values obtained after the least number of epochs necessary for the achievement of e_T; the latter refers to the values obtained when the algorithm goes on until a maximum of 50 epochs, in order to better locate the codewords. N_{ep} is the number of epochs. As regards the test whose target is $\mathrm{RMSE}_{15}(4096)$, the algorithm is stopped as soon as e_T is reached, because of the high number (136) of epochs required. By comparing Table III with Table II, we can clearly see that GNG ensures the desired target with a higher number of codewords. Further, when the e_T decreases, it also needs more epochs than FACS for ensuring the target.

[1]In [32], Fritzke says that, in case of a finite training set, a common measure is the number of cycles through all training patterns, also called *epochs*. Practically, an epoch is equivalent to a FACS-iteration.

TABLE III
GNG PERFORMANCES

e_T		Ep_{min}	Ep_{max}
$\mathrm{RMSE}_{15}(128)$	N_{ep}	5	50
=	N_C	165	165
29.26	$\frac{N_C}{128}$	1.29	1.29
	RMSE	29.20	28.96
$\mathrm{RMSE}_{15}(512)$	N_{ep}	19	50
=	N_C	624	624
22.39	$\frac{N_C}{512}$	1.22	1.22
	RMSE	22.14	21.88
$\mathrm{RMSE}_{15}(4096)$	N_{ep}	136	-
=	N_C	4456	-
10.81	$\frac{N_C}{4096}$	1.09	-
	RMSE	10.78	-

TABLE IV
FOSART COMPARISON

FOSART				FACS
ρ	N_C	$N_{it(\mathrm{FOSART})}$	MSE	N_C (15 it.)
0.0002	92	122	1109.7	65
0.0008	205	111	880.18	124
0.0010	323	84	735.42	194
0.0020	951	96	437.16	706
0.0050	2604	74	199.40	2423
0.0080	3816	59	131.16	3701

D. Comparison With FOSART

Now, we present a comparison with a technique belonging to the family of the ART algorithms proposed by Baraldi and Alpaydin in [29] and [38] and called FOSART. The authors themselves use it also for tasks of VQ. However, FOSART is not an algorithm designed exclusively for VQ, but it can also be used for problems of hidden data structure detection (perceptual grouping) and probability density functions estimation.

The parameters we have used for our tests are the ones used by the authors as default values, while the only quantity we have made change is the vigilance threshold ρ. The termination condition used for FOSART is the same as the one employed for the ELBG, with $\epsilon = 0.0001$ and the MSE as measure for the MQE. The range of values chosen for ρ is such that the number of codewords obtained is inside the range of values of N_C considered for the previous comparisons with ELBG and GNG. The results of the comparison are reported in Table IV. The first column

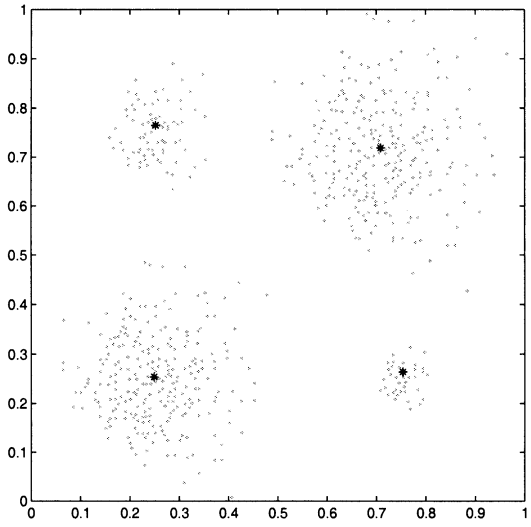

Fig. 14. Prototypes of the competitive agglomertion algorithm after ten iterations.

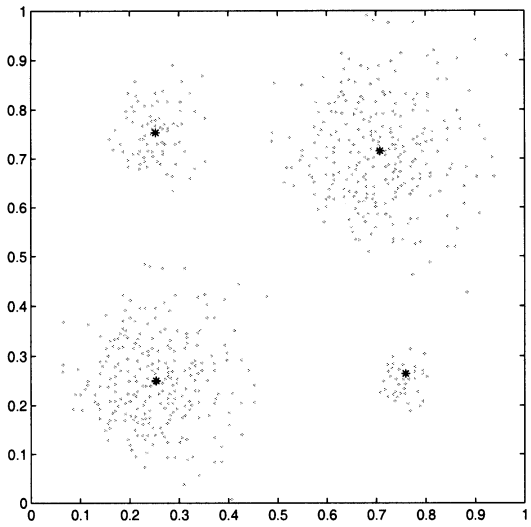

Fig. 15. Codewords of FACS after six iterations.

represents the value of ρ FOSART was launched with. The final MSE obtained by FOSART was given to FACS as target; in the table, the number of codewords determined by FACS after 15 iterations is reported. As we can see, under the same MSE, FACS needs a number of codewords that is clearly less than the number of codewords used by FOSART.

E. Comparison With the Competitive Agglomeration Algorithm

Here, we report a comparison with the technique proposed by Frigui and Krishnapuram in [25]: the competitive agglomeration algorithm. FACS and the competitive agglomeration algorithm share the property that the number of the codewords (or *prototypes*, according to the nomenclature in [25]) has not to be specified.

Let us consider, for example, one of the data set used in [25]. It is reported in Fig. 14, together with the four codewords obtained by the competitive agglomeration algorithm after ten iterations. Once the RMSE previously obtained has been fixed as the target, FACS, in only six iterations, finds four codewords located as in Fig. 15 and the RMSE obtained is practically the same (about 99.8%) as the one of the competitive agglomeration algorithm.

F. Classification

The subject of this section is a comparison between FACS and the GCS algorithm, another technique proposed by Fritzke in [32] and used, in the same paper, also for a problem of supervised classification. Also in this case, the task of finding the right number of codewords is left to the algorithm.

The test mentioned above concerns the problem of the two spirals: it consists of 194 two-dimensional (2-D) vectors lying on two interlocked spirals, which are the classes to be distinguished in this case.

The whole procedure of the classification is constituted by several phases, one of which (the clustering phase) consists in the execution of FACS. We have executed our tests operating in two distinct modalities. In the former, the labels (class 0 or class 1) of the points constituting the spirals are used during the clustering phase; in the latter, the clustering phase occurs without using the labels that are employed in the following phase, i.e., the labeling of the codewords.

1) Mode 1: In this case, the input values are given, together with the related outputs, to the clustering system. The input is constituted by 194 2-D vectors representing the two spirals, while the output is the related membership class (0 or 1).

The procedure we have followed to perform the classification can be summarized in the following four points.

1) Creation of the learning patterns to train FACS: They are constituted by the three-dimensional (3-D) vectors obtained by joining each of the 194 2-D vectors with the related output as the third component.
2) Clustering phase: FACS is launched giving it the 3-D vectors constructed at the previous point as the input data set. Later, we will see the details of this point.
3) Codewords labeling: In order for the classification to happen correctly, the execution of the opposite operation of the one at point 1) is necessary. Therefore, in each 3-D codeword, the components related to the input have to be separated from the ones related to the output. This, as we have done for the learning patterns, is used to label the related 2-D codeword. The codeword will be identified as representing class 0 or 1 whether more 2-D patterns belonging to class 0 or 1 are present inside the cell in question, respectively (majority voting, [52]).
4) Classification: Each 2-D input pattern is compared with all of the labeled 2-D codewords. Once the codeword with the least distance from the pattern in question has been found, its label is read and the pattern is considered as belonging to that class.

The following specifications are necessary so that the execution of FACS [point 2)] occurs correctly.

- Preprocessing of the data: When the components of the vectors constituting the input patterns all have very different statistical distribution (as regards both the order of magnitude and the shape of the distribution) some problems can arise.

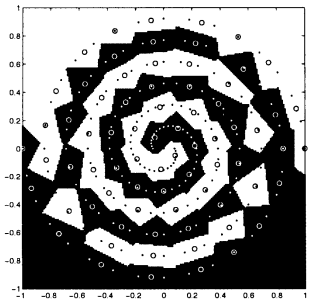

Fig. 16. Two spirals: mode 1. $e_T = 0.01$; $N_C = 74$.

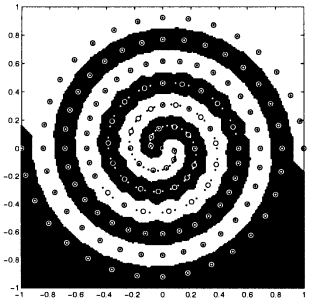

Fig. 17. Two spirals: mode 1. $e_T = 0.001$; $N_C = 144$.

In fact, it is possible that the components with higher dynamics are approximated with high precision, while the ones with lower dynamics are approximated with poor precision. In order to avoid such a situation, the 3-D vectors constituting the input data set for FACS, have been preprocessed by normalizing each component with respect to standard deviations and, for all of the subsequent operations, the vectors so obtained have been employed.

- Distortion measure adopted: We tried to force FACS to generate a codebook with cells of 3-D vectors as *homogeneous* as possible. By the term *homogeneous*, we mean that a cell is constituted by patterns belonging to the same class, i.e., the third component of its vectors is the same for all of them. To realize this condition, we employed the WSE (3) as the distortion measure and we assigned a very high weight to the output component with respect to the one assigned to the input components. In particular, we chose $w_1 = w_2 = 1$ and $w_3 = 100$ (the same result was obtained also for $w_3 = 10$ and, obviously, with $w_3 = 1000$). So, the clustering algorithm is forced to divide the patterns by favoring first their membership class and then also the neighborhood in the 2-D space. We have verified that, for the problem in question and with the value of the weights reported above, already with a value of e_T that generates only two codewords, the related cells are entirely homogeneous. Of course, this does not imply that two codewords are enough for a correct classification of the data. The evaluation of the performances occurs in terms of both classification error and number of iterations executed.
- Classification error: Figs. 16 and 17 graphically report the results found when the values $e_T = 0.01$ and $e_T = 0.001$, respectively, have been used. In both cases, FACS runs for 15 iterations. In the figures mentioned, points represent learning patterns and circles represent codewords. The decision regions (the black and white ones in the figures) determined by the codewords are reported, too. For $e_T = 0.01$, FACS found a codebook with $N_C = 74$, while, for $e_T = 0.001$, a codebook with $N_C = 144$ was found. In the first case, there is only one error in the classification of the learning patterns; in the second case no errors are committed and, as we can see from the figure, the decision-regions are well defined also when we are not in proximity to learning patterns. Even if FACS runs for 200 iterations, it obtains the same number of codewords found after 15 iterations. Afterwards, we have executed the testing of the results obtained by using the same test set employed by Fritzke, constituted by 576 points. He compares this result with the one obtained by Baum and Lang [53] that, for their best model, report an average of 29 errors on the test set. Fritzke, with 145 codewords, obtains zero errors on the same test set; FACS too achieves this result with the 144 codewords of Fig. 17.
- Number of iterations: In [32], Fritzke underlines that, for every learning method, an important practical aspect is the number of pattern presentations necessary to achieve a satisfying performance. Fritzke says that GCS needs 180 epochs to achieve the above mentioned result and reports comparisons with some earlier methods: backpropagation (BP) [54] (20 000 epochs), cross entropy BP [54] (10 000 epochs), cascade-correlation [55] (1700 epochs) and he says that the number of epochs required by GCS is about one or two orders of magnitude less than the other techniques reported.

We can see that FACS needs about 15 iterations to obtain the same result as the GCS, i.e., a twelfth of the epochs required by the GCS. However, after a careful analysis of both FACS and GCS, we have decided that a comparison performed in these terms is not very precise. In fact, the complexity of the epochs (iterations) can be different for each of them. From the analysis effected with several profiling tools we have seen that, in similar algorithms, almost the whole computation time is spent comparing the vectors of the input data set with the vectors of the codebook (distance calculation). So, we can identify the complexity of an epoch (iteration) as the number of comparisons executed inside it. But, due to the incremental nature of such algorithms, the number of codewords is not the same for all the iterations, therefore the complexity of each iteration is different, too.

As regards FACS, for the example reported in the previous section, we counted automatically the comparisons effected and, for $e_T = 0.001$ (that produced the codebook with 144 elements), we found a value of about 0.6 million for the 15 iterations executed.

For GCS, we effected the calculation starting from the values given by Fritzke. It begins with three neurons (that are equivalent to our codewords) and, every 240 pattern-presentations, inserts a new neuron until the final number of 145 cells is reached and for a total number of 180 epochs. We estimated that this is equivalent to the execution of about 2.6 million of comparisons, that is about 4.3 times the number of comparisons effected by FACS.

2) Mode 2: The test on the two spirals problem has been treated also working in another mode. From the practical point

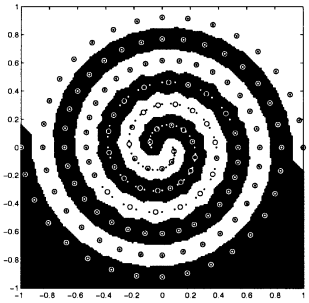

Fig. 18. Two spirals: mode 2. $e_T = 0.001$; $N_C = 143$.

of view, the steps to be executed are the same as the previous ones but, now, the clustering phase occurs using only the part of the patterns related to the input (i.e., the original 2-D patterns) without adding the membership class as the third component. Besides, we use the SE (2) as the measure for the distortion.

Giving to FACS $e_T = 0.01$, it has obtained $N_C = 69$ codewords, producing 44 classification errors for the learning patterns. Using, $e_T = 0.001$, it has found $N_C = 143$ codewords and zero errors have been obtained both on the learning and in the testing set. The results related to the last example are reported in Fig. 18. We can see that this is very similar to Fig. 17. Also in this test, FACS was stopped after 15 iterations and the same considerations about the computational complexity of the iterations made for the previous mode can be repeated here.

VI. Conclusion

This paper has introduced FACS, a new algorithm for clustering and vector quantization that is able to autonomously find the number of codewords once the desired quantization error is specified. The technique uses some concepts previously developed for an algorithm called ELBG which works with a prefixed number of codewords and rearranges them smartly to escape from the local minima of the error function. Comparative studies regarding FACS have shown that it is able to find good results in very few iterations. For complex clustering applications only 15 iterations are sufficient. In comparison to previous similar works a significative improvement in the running time has been obtained.

Further studies will be made regarding the use of different distortion measures to obtain detection of elipsoidal and linear clusters, and planar range segmentation [2]. Other studies will regard the possibility of dealing with input patterns with variable cardinality.

Acknowledgment

The authors wish to thank A. Baraldi of the Joint Research Center, Ispra (VA), Italy, for his patience and useful suggestions during the revision of the manuscript. In addition, he gave the authors his original code of FOSART and detailed instructions for making it work correctly. The authors wish also to thank the reviewers for their work. In particular, the fourth reviewer, thanks to a fine analysis of the manuscript, has given several precise and detailed suggestions for improving the quality of the paper.

References

[1] A. Jain and F. Farrokhnia, "Unsupervised texture segmentation using gabor filters," *Pattern Recognition*, vol. 24, no. 12, pp. 1167–1186, 1991.
[2] H. Frigui and R. Krishnapuram, "A robust competitive clustering algorithm with applications in computer vision," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 21, pp. 450–465, May 1999.
[3] M. C. Clark, L. O. Hall, D. Goldgof, L. P. Clarke, R. Velthuizen, and M. S. Silbiger, "MRI segmentation using fuzzy clustering techniques," *IEEE Eng. Med. Biol.*, vol. 13, pp. 730–742, May 1994.
[4] J. M. Jolion, P. Meer, and S. Bataouche, "Robust clustering with applications in computer vision," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, pp. 791–802, Aug. 1991.
[5] S. K. Bhatia and J. S. Deogun, "Conceptual clustering in information retrieval," *IEEE Trans. Syst., Man, Cybern.*, vol. 28, pp. 427–436, May 1998.
[6] C. Carpineto and G. Romano, "A lattice conceptual clustering system and its application to browsing retrieval," *Machine Learning*, vol. 24, no. 2, pp. 95–122, 1996.
[7] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surveys*, vol. 31, no. 3, pp. 264–323, 1999.
[8] K. K. Paliwal and B. S. Atal, "Efficient vector quantization of LPC parameters at 24 bits/frame," *IEEE Trans. Speech Audio Processing*, vol. 1, pp. 3–14, Jan. 1993.
[9] T. Lookbaugh, E. A. Riskin, P. A. Chou, and R. M. Gray, "Variable rate VQ for speech, image and video compression," *IEEE Trans. Commun.*, vol. 41, pp. 186–199, Feb. 1993.
[10] E. A. da Silva, D. G. Sampson, and M. Ghanbari, "A successive approximation vector quantizer for wavelet transform image coding," *IEEE Trans. Image Processing*, vol. 5, pp. 299–310, Mar. 1996.
[11] P. C. Cosman, R. M. Gray, and M. Vetterli, "Vector quantization of image subbands: A survey," *IEEE Trans. Image Processing*, vol. 5, pp. 202–225, Mar. 1996.
[12] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COM-28, pp. 84–94, Jan. 1980.
[13] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Boston, MA: Kluwer, 1992.
[14] T. Hofmann and J. M. Buhmann, "Competitive learning algorithms for robust vector quantization," *IEEE Trans. Signal Processing*, vol. 46, pp. 1665–1675, Nov. 1998.
[15] B. Fritzke, "The LBG-U method for vector quantization—An improvement over LBG inspired from neural network," *Neural Processing Lett.*, vol. 5, no. 1, pp. 35–45, 1997.
[16] D. Lee, S. Baek, and K. Sung, "Modified K-means algorithm for vector quantizer design," *IEEE Signal Processing Lett.*, vol. 4, pp. 2–4, Jan. 1997.
[17] M. R. Anderberg, *Cluster Analysis for Applications*. New York: Academic, 1973.
[18] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
[19] K. S. Al-Sultan, "A tabu search approach to the clustering problem," *Pattern Recognition*, pp. 1443–1451, 1995.
[20] G. C. Osbourn and R. F. Martinez, "Empirically defined regions of influence for clustering analysis," *Pattern Recognition*, vol. 28, no. 11, pp. 1793–1806, 1995.
[21] F. Kowalewski, "A gradient procedure for determining clusters of relatively high point density," *Pattern Recognition*, vol. 28, no. 12, 1995.
[22] A. B. Geva, Y. Steinberg, S. Bruckmair, and G. Nahum, "A comparison of cluster validity criteria for a mixture of normal distributed data," *Pattern Recognition Lett.*, vol. 21, pp. 511–529, 2000.
[23] X. Zhuang, T. Wang, and P. Zhang, "A highly robust estimator through partially likelihood function modeling and its application in computer vision," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 14, pp. 19–34, Jan. 1992.
[24] X. Zhuang, Y. Huang, K. Palaniappan, and J. S. Lee, "Gaussian mixture modeling decomposition and applications," *IEEE Trans. Image Processing*, vol. 5, pp. 1293–1302, Sept. 1996.
[25] H. Frigui and R. Krishnapuram, "Clustering by competitive agglomeration," *Pattern Recognition*, vol. 30, no. 7, pp. 1109–1119, 1997.
[26] O. Nasraoui and R. Krishnapuram, "A novel approach to unsupervised robust clustering using genetic niching," in *Proc. 9th Int. Conf. Fuzzy Syst.*, San Antonio, TX, May 2000.
[27] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 4–37, Jan. 2000.
[28] V. S. Cherkassky and F. M. Mulier, *Learning From Data: Concepts, Theory and Methods*. New York: Wiley, 1998.

[29] A. Baraldi and E. Alpaydin, "Constructive feedforward ART clustering networks—Part II," *IEEE Trans. Neural Networks*, vol. 13, pp. 662–677, May 2002.
[30] B. Fritzke, "Vector quantization with a growing and splitting elastic net," in *Proc. ICANN 93*, 1993, pp. 580–585.
[31] ——, "Fast learning with incremental RBF networks," *Neural Processing Lett.*, vol. 1, no. 1, pp. 2–5, 1994.
[32] ——, "Growing cell structures—A self-organizing network for unsupervised and supervised learning," *Neural Networks*, vol. 7, no. 9, pp. 1441–1460, 1994.
[33] ——, "A growing neural gas network learns topologies," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds. Cambridge, MA: MIT Press, 1995, pp. 625–632.
[34] F. Hamker and D. Heinke, "Implementation and comparison of growing neural gas, growing cell structures and fuzzy artmap," Schriftenreihe des FG Neuroinformatik der Tech. Univ. Ilmenau, Ilmenau, Germany, Tech. Rep. 1/97, 1997.
[35] T. Martinetz and K. J. Schulten, "A neural-gas network learns topologies," in *Artificial Neural Networks*, T. Kohonen, K. Makisara, and O. Simula, Eds. Amsterdam, The Netherlands: Elsevier, 1991, pp. 397–402.
[36] G. A. Carpenter, S. Grossberg, M. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE Trans. Neural Networks*, vol. 3, pp. 698–713, May 1992.
[37] S. Grossberg, "Adaptive pattern classification and universal recording: I. Parallel development and coding of neural feature detectors," *Biol. Cybern.*, vol. 23, pp. 121–134, 1976.
[38] A. Baraldi and E. Alpaydin, "Constructive feedforward ART clustering networks—Part I," *IEEE Trans. Neural Networks*, vol. 13, pp. 645–661, May 2002.
[39] G. Patanè and M. Russo, "The enhanced LBG algorithm," *Neural Networks*, vol. 14, pp. 1219–1237, Nov. 2001.
[40] S. P. Lloyd, "Least squares quantization in PCM's," Bell Telephone , Murray Hill, NJ, Lab. Paper, 1957.
[41] A. Gersho, *Digital Communications*. Amsterdam, The Netherlands: Elsevier, 1986, ch. Vector Quantization: A New Direction in Source Coding.
[42] A. Gersho, "Asymptotically optimal block quantization," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 373–380, July 1979.
[43] C. Chinrungrueng and C. H. Séquin, "Optimal adaptive K-means algorithm with dynamic adjustament of learning rate," *IEEE Trans. Neural Networks*, vol. 6, pp. 157–169, Jan. 1995.
[44] M. Russo and G. Patanè, "Improving the LBG algorithm," in *Proc. IWANN'99*, J. Mira and J. V. Sánchez-Andrés, Eds. Barcelona, Spain: Springer-Verlag, June 1999, vol. 1606, Lecture Notes in Computer Science, pp. 621–630.
[45] M. Russo, "FuGeNeSys: A genetic neural system for fuzzy modeling," *IEEE Trans. Fuzzy Syst.*, vol. 6, pp. 373–388, Aug. 1998.
[46] G. Patanè and M. Russo, "ELBG implementation," *Int. J. Knowledge-Based Intell. Eng. Syst.*, vol. 4, pp. 94–109, Apr. 2000.
[47] S. Z. Selim and M. A. Ismail, "K-means-type algorithms: A generalized convergence theorem and characterization of local optimality," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 81–87, Apr. 1984.
[48] D. C. Munson, Jr., "A note on Lena," *IEEE Trans. Image Processing*, vol. 5, p. 3, Jan. 1996.
[49] B. Fritzke, "A self-organizing network that can follow nonstationary distributions," in *Proc. ICANN 97*. New York: Springer-Verlag, 1997, pp. 613–618.
[50] N. B. Karayiannis and P.-I. Pai, "Fuzzy algorithms for learning vector quantization," *IEEE Trans. Neural Networks*, vol. 7, pp. 1196–1211, Sept. 1996.
[51] B. Fritzke, "Some competitive learning methods," Inst. Neural Comput., Ruhr-Univ. Bochum, Germany, Tech. Rep., 1997.
[52] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Clarendon, 1996.
[53] E. B. Baum and K. E. Lang, "Constructing hidden units using examples and queries," in *Advances in Neural Information Processing Systems 3*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 904–910.
[54] K. J. Lang and M. J. Witbrock, "Learning to tell two spirals apart," in *Proc. 1988 Connectionist Models Summer School*, D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, pp. 52–59.
[55] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.

Giuseppe Patanè was born in Catania, Italy, in 1972. He received the Laurea degree in electronic engineering from the University of Catania, Catania, Italy, in 1997 and the Ph.D. degree from the University of Palermo, Palermo, Italy, in 2001, respectively.

In 2001, he joined the Department of Physics at the University of Messina, Italy, where he is currently a Researcher of Computer Science. Currently, he is also with the National Institute for Nuclear Physics (INFN), Catania, Italy. His primary interests are soft computing, very large-scale integration (VLSI) design, optimization techniques, and distributed computing.

Marco Russo was born in Brindisi, Italy, in 1967. He received the M.A. and Ph.D. degrees in electronical engineering from the University of Catania, Catania, Italy, in 1992 and 1996, respectively.

Since 1998, he has been with the Department of Physics, University of Messina, Messina, Italy, as an Associate Professor of Computer Science. Currently, he is in charge of Research at the National Institute for Nuclear Physics (INFN). His primary interests are soft computing, VLSI design, optimization techniques, and distributed computing. He has more than 90 technical publications appearing in international journals, books, and conferences. He is Co-Editor of the book *Fuzzy Learning and Applications* (Boca Raton, FL: CRC).