Parallel Clustering on a Commodity Supercomputer

Giuseppe Patanè
Institute of Computer Science and Telecommunications
Faculty of Engineering, University of Catania
V.le A. Doria 6, 95125 Catania - ITALY,
INFN-Section of Catania, Corso Italia, 57, 95129 Catania - ITALY;
e-mail:gpatane@iit.unict.it

Marco Russo
Dept. of Physics - Faculty of Engineering, University of Messina,
Contrada Papardo, Salita Sperone 31, 98166 Sant'Agata (ME) - ITALY,
INFN-Section of Catania, Corso Italia, 57, 95129 Catania - ITALY;
e-mail: mrusso@ingegneria.unime.it,mrusso@ieee.org;
Tel: +39 (0)347 1749070

Abstract

k-means based clustering algorithms have interesting performances in several application fields. The computational complexity of these techniques depends on the size of the data set and the codebook. The larger the data set and the codebook, the greater the computing time to reach the convergence. This paper illustrates the behaviour of two clustering algorithms we have realized and parallelized on a commodity supercomputer.

Keywords

Parallel, Clustering, Unsupervised Learning, GLA, LBG, ELBG

I. Introduction

Clustering algorithms attempt to organize unlabeled feature vectors into groups and represent each group with some prototypes. These algorithms, which are often used for unsupervised learning, have many application possibilities ranging from classification and pattern recognition to audio and video data compression.

Among the various possible approaches to clustering, we consider the Vector Quantization (VQ) [1]. The objective of VQ is the representation of a set of vectors in $\Re^k$ by means of a reduced set, whose elements are in $\Re^k$, too. The reduced set is called the *codebook* and its elements *codewords*. Codewords are determined trying to minimize an objective function representing the Quantization Error (QE) [2].

k-means-based techniques are very expensive in terms of computational effort, above all when we must deal with large data sets and many codewords.

This paper reports our results regarding the parallelization of these techniques with the aim of reducing the computing time. For our studies we have used a commodity supercomputer based on a heterogeneus form of 32 PCs with a unique fast-Ethernet network we have realized at the University of Messina. However, for time reasons, this work reports the performances of the algorithms by using a maximum of 16 machines.

In particular, we analyze the performance of the classical LBG algorithm and of the ELBG. The ELBG is an enhanced version of the LBG that we have developed and some preliminary results were presented in [3], [4], [5]. It belongs to the hard k-means vector quantization family, but it is almost insensitive to the initial choice of the codewords and no parameters have to be configured by hand. Besides, it has performances better than or equal to other existing algorithms paying a negligible price in terms of computational overhead.

These studies show that the reachable speedup grows when the clustering problem becomes more complex. Furthermore, in our realization of the LBG and the ELBG, we have found that the LBG is easier to parallelize.

II. Vector Quantization (VQ)

Definitions. A Vector Quantizer is a function $q : X \longrightarrow Y$ that represents 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 the *codebook* and its elements *codewords*. The vectors of X are called also *input patterns* or *input vectors*. According to

the previous definitions, we can obtain a partition $\mathcal{S}$ of X constituted by the N_C subsets S_i (called cells): $S_i = \{\mathbf{x} \in X : q(\mathbf{x}) = \mathbf{y}_i\} \;\; i = 1, \ldots, N_C$.

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 performance of a vector quantizer is given by its mean QE (MQE). Several functions can be adopted as distortion measures [6]. The most widely adopted is the Euclidean distance and we will refer to it in the rest of the paper.

LBG. LBG was proposed in 1980 [6] by Linde, Buzo and Gray and its original name was Generalized Lloyd Algorithm (GLA) because it extended the Lloyd's technique [7] from mono- to k-dimensional cases. The name LBG comes from the initials of its authors. It is an algorithm, that, at every iteration, generates a quantizer whose MQE is less than or equal to the previous one. This is obtained by a process of optimization where the best partition (given the codebook) and the best codebook (given the partition) are alternatively obtained. Several techniques for calculating an initial codebook exist [6] but, in this paper, we are not interested in this phase. Therefore, we assume that a random choice of the initial codewords is enough to start-up the algorithm. The LBG can be summarized as follows;

1. an initial codebook is given;
2. given the codebook, the best partition is calculated (Voronoi partition). It is obtained by assigning each pattern to the nearest codeword to it;
3. termination condition check: the current MQE of the quantizer is compared to the one at the previous iteration. If it decreased less than a prefixed threshold, the algorithm ends; otherwise it continues;
4. given the partition calculated at point 2, the best codebook is calculated. Its elements are the geometrical barycentres of the cells; it can be demonstrated that this is the optimal choice when the Euclidean distance is adopted as the distortion measure.
5. return to step 2.

Enhanced LBG (ELBG). In [3], [4], we tried to explain why the LBG suffers the problem of the local minima deriving from a bad initialization of the codebook. There, we illustrated the technique we employed to solve such a problem. It consists in the pursuit of the equalization of the total distortions introduced by each cell. The total distortion of a cell is the sum of the QEs of the vectors belonging to it. The operations we execute to try to obtain such an equalization are based on a stochastic process for selecting some cells to be split and joined. They are executed inside another step, called the ELBG block, which is placed between points 3 and 4 of the LBG. For this reason, we called our algorithm ELBG. The interested reader can see [3], [4].

III. Our commodity supercomputer

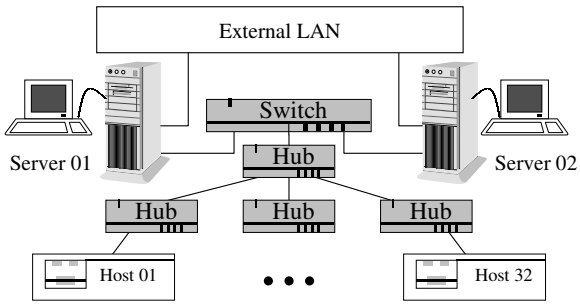

Fig. 1: The elaboration system we used

The system. The elaboration system (see Fig. 1) on which we implemented the ELBG is a cluster of PCs with the LINUX operating system. It consists of 32 Processing Elements (PEs) and a server (duplicated for redundancy) acting also as a gateway towards the external LAN. All of the 32 PEs are mono-processor machines, 7 are based on Pentium-II 233 and 25 on Celeron 366. All of them are endowed with 32 MB of RAM, one hard disk and one Fast-Ethernet network card. One server is a bi-processor computer with two Pentium-II 300 and two network cards. The PEs are interconnected by a dedicated Fast Ethernet network and all of them share the same bus. Physical connections have been realized by means of four 12-port hubs distributed between two levels as we show in Fig. 1. However the electrical topology is such that all of the 32 hosts are placed on the same bus. The highest level hub is connected to a switch, where the servers are directly connected, too.

Inter-process communication is realized with the Parallel Virtual Machine (PVM) software[1]. It makes the user see several computers as a large and single virtual machine and every process (task, according to the PVM nomenclature) has a unique identifier (task identifier, tid) inside the virtual machine. The PVM uses

[1] It is available for free at http://www.netlib.org/pvm

the transport functionality provided by the TCP/(UDP/)IP that are directly implemented inside the kernel of LINUX. Together with the PVM, we employ the graphic tool named XPVM; it allows us to visualize the diagram of the activity of the tasks versus time. It is very useful during the debugging and the testing of parallel algorithms.

The management of the system. We think that, to administrate our system, a centralized policy is suitable. However, a totally centralized solution, based on a Network File System (NFS), would generate additional network traffic and would decrease the availability of transmission bandwidth for inter-process communication. For this reason, we have chosen a hybrid solution between a centralized policy and a local one. Each host, in normal operative conditions, uses a file-system stored on the local hard disk. However, it is a copy, identical for all the hosts, of a file-system stored on the server. So, we need to establish the configuration of only two machines: the server and one host. In fact, once a generic host has been configured (upgraded), its file-system is transferred to the server and, automatically, all of the other hosts download it.

IV. Parallelization

In this section we describe our implementation of the parallel versions of the LBG and the ELBG.

In [3], we saw that the overhead introduced to the traditional LBG by the execution of the ELBG block is quite low. Besides, the profiling we executed on the software we wrote to implement the LBG allowed us to see that almost all of the calculation time is spent to determine the Voronoi partition (point 2 of the LBG algorithm). More precisely, this time is spent to calculate the distance between each input pattern and all of the codewords, in order to choose the nearest one. Our work focused on the parallelization of this operation by trying to share the load among all of the hosts. Now, we will describe our parallel implementation of the LBG and, after, the parallel version of the ELBG, that is identical to the LBG with only the addition of the ELBG block.

A. LBG

As we previously said, during the calculation of the Voronoi partition, each input pattern is assigned to the nearest codeword to it. As this is done by comparing each input vector with all of the codewords, it means that the distance (Euclidean, in this case) between two k-dimensional vectors is calculated $N_P \times N_C$ times. If we have N processors at our disposal, we can launch, on each processor, a process (task, according to the PVM nomenclature) that executes $\frac{N_P \times N_C}{N}$ of distance calculations. In practice, each task keeps in the memory a different portion of the input patterns and the whole codebook. It has to find the nearest codeword only for the portion of input patterns that were assigned to it.

Given N equal hosts, we adopted a master-slaves policy. At each iteration, the master calculates, together with the slaves, the Voronoi partition, collects the results and, finally, calculates the new codebook alone. At the beginning of the new iteration, the new codebook is broadcast to all of the slaves.

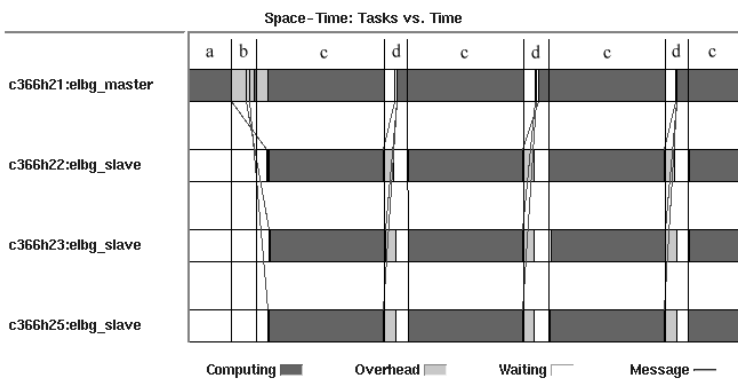

Fig. 2: Task vs. Time diagram for the LBG with $k = 16$, $N_P = 16384$, $N_C = 128$, $N = 4$. The scale of times is such that an iteration is about 0.75 s.

Fig. 2 is a screen-shot taken from the X-PVM. It reports a part of the diagram of the task activity versus the time. It refers to a run of the parallel LBG on 4 hosts, with an input data set of 16384 16-dimensional vectors and a codebook of 128 elements. The input data set was obtained from the image of Lena [8] of 512×512 pixels at 256 grey levels. It was divided into blocks of 4×4 pixels, and, in this way, 16384 16-dimensional vectors were generated. The figure illustrates the periods of activity and inactivity of the tasks, the exchange of the messages and the overhead introduced by their transmission. The XPVM does not make the graphic tracing of the broadcast messages[2], that in our algorithm are the messages related to the transmission of the codebook from the master to the slaves. The scale of times was divided into parts labeled as (a), (b), (c) and

[2]The PVM provides a function to broadcast data to all the tasks; however, this is not a real broadcast function because it is implemented by sending the same copy of the data to all the tasks, one at a time.

(d); now, we will describe the operations executed during each of them.

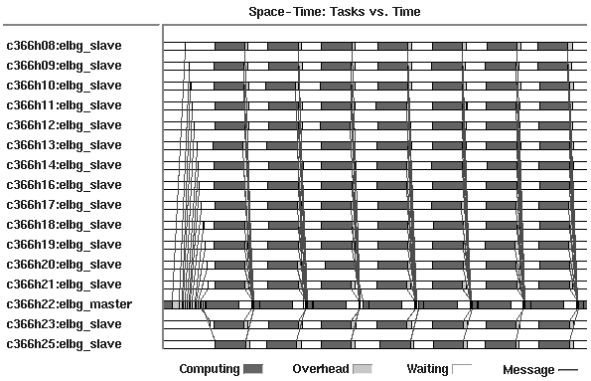

Fig. 3: Task vs. Time diagram for the LBG with $k = 16$, $N_P = 16384$, $N_C = 256$, $N = 16$. The scale of times is such that an iteration is about 0.60 s.

(a) Preliminary phase. The master spawns the slaves and waits until all of them start correctly.
(b) Distribution of the input data set. In this phase, the master distributes (just once), to each slave, the portion of input patterns for which it has to calculate the nearest codeword.
(c) Voronoi partition calculation. This is the heart of the parallel algorithm. The first operation in this phase is the broadcast of the codebook from the master to all the slaves. Even if we cannot see the graphic tracing of the broadcast message, we can see that, for the considered example, the transmission of the codebook introduces a negligible delay with respect to the whole phase. In fact, by carefully looking at all segments labeled as (c), we see that all of the slaves, practically, start working at the beginning of the phase. This is not true for the first iteration (the one after segment (b)), but, from the same picture, we can see that such a delay is the consequence of the messages transmitted during phase (b) still occupying the network. Phase (c) ends when the master completes the calculation of its quota of the Voronoi partition.
(d) Collection of the results from the slaves and calculation of the new codebook. This is the critical part for the parallelization; in fact, during its execution, only the master works, while the slaves, after they have sent their results, wait for the new codebook from the master. Therefore, it is opportune to try to minimize its duration. We effected several tests of the algorithm when k, N_P, N_C and N change and we analyzed the results, both the graphic ones from the XPVM, and the numerical ones from the profiling of the master. We concluded that, for the implementation of the parallel LBG in our system, it is worthwhile to minimize the information transmitted on the network, but, in any case, leaving the master without the load of the calculation of the Euclidean distance, except for determining its portion of the Voronoi partition. According to these considerations, the information transmitted from a slave to the master is constituted, for each pattern, by the index of the nearest codeword and by the QE related to it. Each slave sends these results when it has completed the calculation of its portion of the Voronoi partition. When all of these values have been collected by the master, it is able to calculate the total distortion and the new codebook. The master cannot calculate the new codebook until all of the data have been received. In order to minimize its periods of inactivity, it operates as follows: it checks if the results from any slave have been received and executes all of the operations it can execute with those data; after, it checks if there are other results, and so on until it has received all of them. From the profiling of the master, effected for $N = 1, 2, 4, 8, 16$, we saw that the time it employs to calculate the new codebook, after all the results have been collected, is less than 1%; so, it can be considered negligible.

Fig. 3 refers to the same input data set employed for the example of Fig. 2 and, in this case, we have $N_C = 256$ and $N = 16$. We can see that the periods of inactivity of the tasks increase when N increases, too. This is due to the greater quantity of data to be transmitted over the network. Therefore, when N reaches a certain value, the adding of other processors does not produce any benefit, i.e. the speed-up saturates.

B. ELBG

As we previously said, our implementation of the parallel ELBG is substantially identical to the implementation of the LBG with the addition of the ELBG block. It is executed, in serial mode, only by the master.

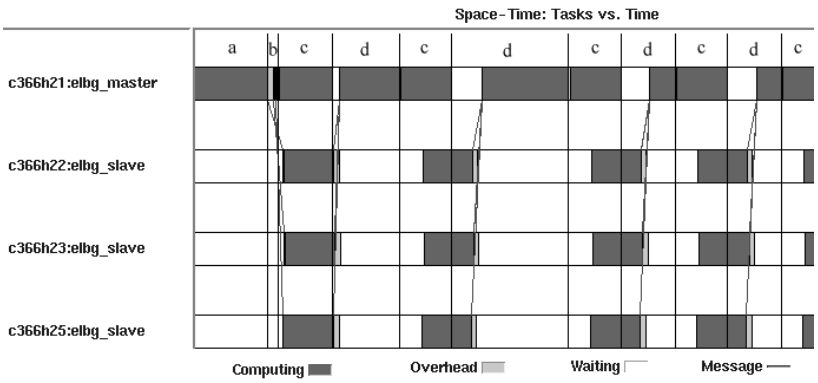

Fig. 4: Task vs. Time diagram for the ELBG with $k = 16$, $N_P = 16384$, $N_C = 128$, $N = 4$. The scale of times is such that an iteration is is about 1.27 s.

Fig. 4 reports a part of the diagram of the activity of the tasks versus the time, for the same problem we considered in Fig. 2 for the LBG. The meaning of segments (a), (b), (c) and (d) is the same as in Fig. 2 with the only difference that, in this case, inside phase (d), before the new codebook calculation, the master also executes the ELBG block. In order to visualize a greater number of iterations, in Fig. 4, we chose a different scale for the axis of the times with respect to Fig. 2. However, we can take as a reference the time of "Computing" of the slaves at each iteration; it is the same both in the LBG and in the ELBG.

The first difference we can notice between Figs. 2 and 4 is that, in the latter, the periods of inactivity of the slaves are longer than in the former. This has to be attributed to the execution of the ELBG block by the master; such an operation makes segments (d) longer than in the LBG. We verified that the periods of inactivity are reduced when the complexity of the problem increases. In the cases we analyzed, an increase in the complexity consists in the increase of N_C, being N_P fixed. Besides, as we can see in Fig. 4, the length of segments (d) is not fixed because of the ELBG block, whose computing time is not deterministic as in the calculation of the Voronoi partition.

Another difference between the diagrams, that we think could be imputable to the pair PVM-TCP(UDP), is related to segments (c). Both the LBG and the ELBG transmit the new codebook to the slaves at the beginning of (c) and, as the dimensions of the codebook are the same in both of the cases, the quantity of data to send over the network is the same. However, the time required for the transmission is negligible in Fig. 2, while it is not in Fig. 4. We repeated the runs several times, but the result did not change. We think this could derive from different behaviour of the pair PVM-TCP(UDP) in the two situations we examined. As a consequence of this, in the ELBG, the period of inactivity of the master at the beginning of segments (d) is longer than in the LBG.

V. Results

		LBG		ELBG	
N_C	N	T it. (s)	S	T it. (s)	S
32	1	0.62	1.00	0.79	1.00
	2	0.48	1.30	0.64	1.23
	4	0.23	2.73	0.40	1.97
	8	0.16	3.89	0.33	2.40
	16	0.13	4.60	0.32	2.49
64	1	1.17	1.00	1.41	1.00
	2	0.76	1.54	1.18	1.19
	4	0.45	2.60	0.87	1.61
	8	0.24	4.89	0.75	1.88
	16	0.19	6.26	0.73	1.93
128	1	2.28	1.00	2.60	1.00
	2	1.31	1.73	1.85	1.40
	4	0.75	3.05	1.27	2.04
	8	0.49	4.69	1.00	2.61
	16	0.28	8.17	0.91	2.86
256	1	4.51	1.00	5.00	1.00
	2	2.44	1.85	3.17	1.58
	4	1.31	3.44	2.02	2.48
	8	0.77	5.83	1.48	3.39
	16	0.60	7.51	1.25	4.00
512	1	9.30	1.00	9.90	1.00
	2	4.77	1.95	5.77	1.72
	4	2.49	3.73	3.50	2.83
	8	1.37	6.79	2.37	4.18
	16	1.03	9.02	1.86	5.32
1024	1	18.56	1.00	20.47	1.00
	2	9.39	1.98	11.10	1.84
	4	4.77	3.89	6.94	2.95
	8	2.59	7.17	4.41	4.65
	16	1.65	11.24	3.22	6.35

TABLE I: Speed up (S) for the LBG and the ELBG

In this section we report the performances of the parallel LBG and ELBG in terms of speed-up's (S). Performances, in terms of final MQE and number of iterations required for the convergence, are identical to the ones obtained by the corresponding serial versions. On this subject, several comparisons with other existing techniques [9], [10], [11], [12], both hard and fuzzy, both k-means and competitive learning, are reported in [3], [4], [5]. In such comparisons, the ELBG obtained results better than or equal to all of the other considered techniques, both for the final MQE and for the number of required iterations. Besides, the difference in favour of the ELBG increases when the complexity of the clustering problem increases, too and the final MQE the ELBG obtains is, practically, independent of the initial conditions.

Before reporting the numeric values related to S, we think that some considerations regarding the validity of the results obtained in respect of the times of calculation are worthwhile. The LBG is a deterministic algorithm, i.e., after the initial codebook has been fixed, it always develops in the same way. The same deterministic behaviour has to be followed both by the serial and by the parallel implementation of the LBG. However, the utilization of a Fast Ethernet-based network, whose management is not deterministic, gave us slightly variable results. Regarding the ELBG, we must also consider the non-deterministic behaviour of the ELBG block. So, in order to make

allowance for these factors, all of the results we present are the mean value of 20 runs. In Tab. I we report the performances of the LBG and the ELBG in terms of S of the time required per iteration; the input data set is the same we adopted in the previous section. Some considerations regarding these results follow.

- We can see that, for all of the examined cases, the LBG presents a higher value of S than the ELBG. This is obvious because of the serial execution of the ELBG block only by the master, while the time of activity for the slaves is the same as the LBG.
- The number of processors (N) being equal, generally, S increases when the complexity of the problem increases, too. In this case, the complexity is determined by the dimensions of the codebook (N_C) because we fixed the input data set. Such a trend is justified by the decrease in the time of inactivity of the slaves. However, there are some cases where this trend is inverted. For example, let us look at the LBG when we change $N_C = 32$ into $N_C = 64$ for $N = 4$, $N_C = 64$ into $N_C = 128$ for $N = 8$, $N_C = 128$ into $N_C = 256$ for $N = 16$. We saw that such behaviour derives from the increase of the overhead in the transmission of the results from the slaves to the master; in our opinion, this could be imputable to the behaviour of the pair PVM-TCP(UDP).
- N_C being fixed, when N increases, S saturates. This occurs because the increase of N prolongs the periods of inactivity of the slaves and, in practice, their contribution to the algorithm no longer increases. Both for the LBG and the ELBG, when N_C increases, saturation occurs for higher values of N. Also in this case, it is a good trend because both of the parallel algorithms we considered perform better with high complexity problems.

VI. Conclusions and future developments

The work we presented in this paper is a preliminary study regarding the parallelization of clustering algorithms on a commodity supercomputer. In particular, we analyzed two techniques for VQ: the LBG and the ELBG. We could see that some points, such as the calculation of the Voronoi partition, are easy to implement on the system we described, other can be improved. Regarding the ELBG, we are thinking of developing a new version of the ELBG block that could be executed in parallel by all of the hosts in order to increase the speed up of the whole algorithm. Another improvement, whose benefits would interest both the LBG and the ELBG, could be introduced in the inter-process communication by the utilization of a real broadcast function. In fact, in the current implementation of the two algorithms, the broadcast transmission of the codebook from the master to the slaves is effected by the PVM by simply sending the same copy of the data, one at a time. With a real broadcast function (like the one provided by the UDP on a Fast Ethernet network), the master could transmit the codebook on the network, just once per iteration, and all of the slaves could read it at the same time.

References

[1] A.Gersho and R.M.Gray, *Vector Quantization and Signal Compression*. Boston: Kluwer, 1992.

[2] T.Hofmann and J.M.Buhmann, "Pairwise Data Clustering by Deterministic Annealing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 1–14, Jan. 1997.

[3] M.Russo and G.Patanè, "The Enhanced-LBG Algorithm," *IEEE Transactions on Knowledge and Data Engineering*, submitted.

[4] M.Russo and G.Patanè, "Improving the LBG Algorithm," in *Proc. of IWANN'99* (J.Mira and J.V.Sánchez-Andrés, eds.), vol. 1606 of *Lecture Notes in Computer Science*, (Barcelona, Spain), pp. 621–630, Springer, June 1999.

[5] G.Patanè and M.Russo, "Comparisons between Fuzzy and Hard Clustering Techniques," in *Proceedings of WILF'99*, June 1999. in press.

[6] Y.Linde, A.Buzo, and R.M.Gray, "An Algorithm for Vector Quantizer Design," *IEEE Transaction on Communications*, vol. 28, pp. 84–94, Jan. 1980.

[7] S.P.Lloyd, "Least Squares Quantization in PCM's." Bell Telephone Laboratories Paper, Murray Hill, 1957.

[8] D.C.Munson, Jr., "A Note on Lena," *IEEE Transactions on Image Processing*, vol. 5, p. 3, Jan. 1996.

[9] C.Chinrungrueng and C.H. Séquin, "Optimal adaptive K-Means Algorithm with Dynamic Adjustament of Learning Rate," *IEEE Transaction on Neural Networks*, vol. 6, pp. 157–169, Jan. 1995.

[10] B.Fritzke, "The LBG-U Method for Vector Quantization – an Improvement Over LBG Inspired from Neural Network," *Neural Processing Letters*, vol. 5, no. 1, pp. 35–45, 1997.

[11] D.Lee, S.Baek, and K.Sung, "Modified K-means Algorithm for Vector Quantizer Design," *IEEE Signal Processing Letters*, vol. 4, pp. 2–4, Jan. 1997.

[12] N.B.Karayiannis and Pin-I Pai, "Fuzzy Algorithms for Learning Vector Quantization," *IEEE Transaction on Neural Networks*, vol. 7, pp. 1196–1211, Sept. 1996.