Multimedia Coding
Implementation of LBG Algorithm

Anıl Adar 1216126

Department of Information Engineering
University of Padova, Padua, Italy
Email: anil.adar@studenti.unipd.it

Abstract

In general, multimedia coding aims to compress source data efficiently to transmit them using less bandwidth and to save the data in a more compact way. Many algorithms have been developed to relieve the constraints on storage space and transmission capacities and are currently used in products on the shelf. In this report, the steps of implementation of the LBG algorithm for CD-Quality audio signals are explained and the performance evaluation of the technique used is shown.

1 Introduction

Depending on the requirements in the area of use, we can divide the data compression methods into two: lossless compression method, where the input data is identical to the output data, or the lossy compression method, which provides much more compression, the output data may differ from the input data.

- *Lossless Compression:* Lossless compression, as the name suggests, does not cause any loss of information. If the data is compressed using this method, the original data can be recovered exactly (it is invertible). The idea behind lossless compression is to assign short codevectors to high probability symbols, and long codevectors to low probability symbols. The goal here is to minimize the average code length. The most important area where lossless compression is used is text compression. Because the slightest change in the text may cause distortion and misunderstanding. Examples of lossless compression algorithms include Huffman Coding, Arithmetic Coding, Dictionary-based Coding (LZ77, LZ78, LZW) and Run-length coding.

- *Lossy Compression:* In the lossy compression technique, a reconstruction of the approximation of original data is performed. Some distortions occur during reconstruction and these are generally not invertible (cannot be recovered exactly). In this technique, very high compression rates can be achieved and unlike lossless compression, analog signals can be compressed. The goal of this technique is to reconstruct the highest fidelity data subject to compressed data size (bit-rate) constraints and to provide the highest compression rate subject to distortion constraints between the reconstructed data and the original data. Examples of lossy compression algorithms include JPEG for photos, MP3 for audio files, and MPEG / H.264 for video files. When we deal with a continuous source, the outcome of each random variable may be any real number, so an infinite number of bits would be needed to represent a symbol. The number of possible source output is generally larger than the number of codewords to represent them. In this case, a process is used which is called quantization. Quantization is able to represent a large set of values with a smaller set. If the set of input and output of the quantizer is scalar, it is called *scalar quantizer.* If the input and output set of the quantizer is a vector, then it is called a *vector quantizer.*

The rest of the paper is organized as follows. Chapter 2 explains the encoding and the decoding procedure of the vector quantization and then it explains the Linde-Buzo-Gray(LBG) algorithm with its Matlab® implementation.

2 Technical Approach

2.1 Vector Quantization

In Vector Quantization (VQ), the source output into blocks or vectors is grouped together. This is an immediate generalization of scalar quantization of a single random variable.
VQ takes L consecutive source symbol block as an input:

$$\text{a vector } \mathbf{x} \in \mathbb{R}^L \text{ where } \mathbf{x} = (x_1, ..., x_L)$$

outputs an element of the codebook

$$\mathbf{C} = \{\mathbf{y}_1, ..., \mathbf{y}_K\} \text{ where } \mathbf{y}_i \in \mathbb{R}^L \text{ with } i = 1, ..., K$$

are called codevectors (or reproduction vectors). A VQ also requires to define a set of decision regions (or cells):

$$I_i \in \mathbb{R}^L, i = 1, ..., K \text{ such that } I_i \cap I_j = \emptyset \;\; \forall \;\; i \neq j \text{ and } \bigcup_{i=1}^{K} I_i = \mathbb{R}^L$$

and quantization rule:

$$Q(x) = \mathbf{y_i} \text{ if } \mathbf{x} \in I_i$$

We also need to measure the average distortion as a quality requirement of the vector quantizer. Mean squared is a common distortion measure and the quality is measured by mean squared error.

$$D = \frac{1}{L}\mathbb{E}[d(x, Q(x))] = \int_{\mathbb{R}^L} ||x - Q(x)||_2^2 \, f_X(x)dx = \frac{1}{L}\sum_{i=1}^{K}\int_{I_i} ||x - y_i||_2^2 \, f_X(x)d$$

and the Signal-to-Noise Ratio(SNR):

$$\text{SNR} = \frac{{\sigma^2}_{\text{signal}}}{{\sigma^2}_{\text{noise}}} \qquad \text{SNR}_{\text{dB}} = 10.\log_{10}(\text{SNR})$$

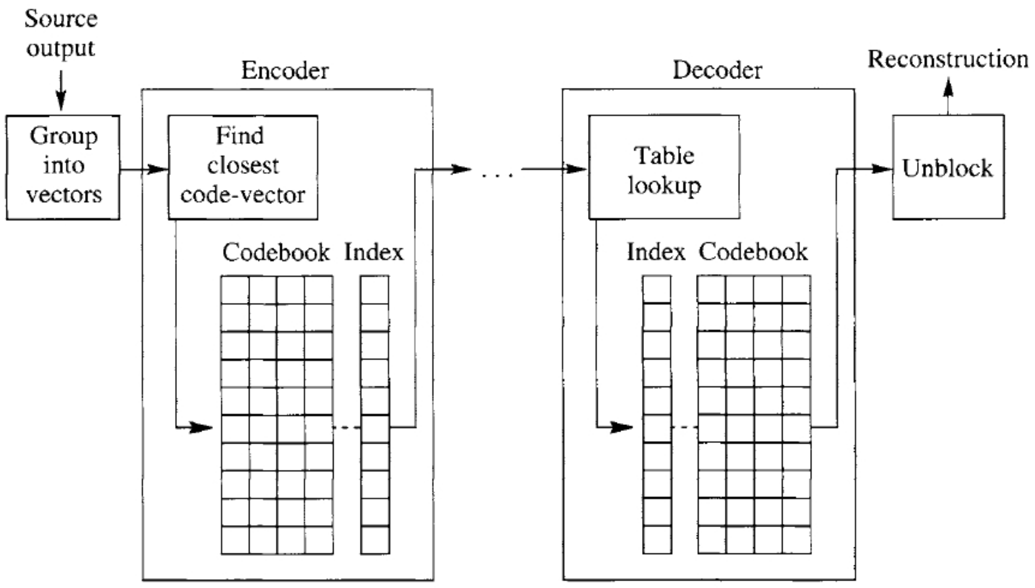

Figure 1: Vector Quantization Procedure[1]

2.1.1 Encoding Procedure

As stated in Section 2.1, in VQ, we group the K source outputs as L-dimensional blocks or L-dimensional vectors. These blocks (or vectors) are inputs of the VQ. These 3-dimensional vectors we have are called the codebook and each row of vectors in this codebook is called the codevector. Each vector is assigned a binary index. Then, the codevector is compared to the input vector to find the codevector which is closest to the input one. We transmit and store binary indices to solve at the decoder which codevector is closest to the input vector.

2.1.2 Decoding Procedure

On the decoder side, we have also the same codebook. The decoder can reconstruct the codevectors using binary indices sent from the encoder. Although the amount of calculation in the encoder increases linearly with the codebook size, the decoder can retrieve the codevectors only by looking at the binary number table. This makes the decoding process computationally efficient (limited processing power restricts the computational ability of mobile users due to the operation of the devices with battery).

[1]Figure taken from ref[1]

2.1.3 Optimality Conditions

The optimal vector quantizer design is based on finding and partitioning the codebook that minimizes distortion. The necessary conditions to ensure optimality can be listed as follows:

- *Nearest Neighbor Condition*: given the set of codevectors $y_{\mathrm{i}} \in \mathbb{R}^{\mathrm{L}}$, i=1,...,K, the optimal partition of $\mathbb{R}^{\mathrm{L}}$ is given by the minimum distortion

$$I_i = \{\mathbf{x} \in \mathbb{R}^L \text{ such that } ||\mathbf{x} - \mathbf{y}_i||_2^2 \le ||\mathbf{x} - \mathbf{y}_j||_2^2, i \neq j\}, i = 1, ..., K$$

- *Centroid Condition*: given the partition I_i the codevectors of the codebook are centroids of the decision regions that minimize the distortion:

$$\mathbf{y}_{\mathrm{i}} = \frac{\int_{I_i} \mathbf{x}\, f_X(\mathbf{x}) d\mathbf{x}}{\int_{I_i} f_X(\mathbf{x}) d\mathbf{x}} = \int_{I_i} \mathbf{x}\, f_{X|\mathbf{x}\in I_i}(\mathbf{x}|\mathbf{x} \in I_i) d\mathbf{x}, i = 1, ..., K$$

In the next topic, the LBG algorithm, one of the most popular iterative algorithms that try to satisfy these two conditions at the same time, explained.

2.2 LBG Algorithm

This algorithm is familiar with the K-means algorithm. In K-means, the distribution is known. However, there may be situations where it is not known. The algorithm steps for both cases are examined below;

- *The distribution is known*:

  1. Start with an initial codebook$\{\mathbf{y}_{\mathrm{i}}{}^{(0)},...,\mathbf{y}_{\mathrm{K}}{}^{(0)}\}$, set n=0, $D^{(0)} = \infty$. Select threshold $\epsilon > 0$.
  2. Find the optimal decision regions with respect to the Nearest Neighbour Condition

     $$I_i^n = \{\mathbf{x} \in \mathbb{R}^L \text{ such that } ||\mathbf{x} - \mathbf{y}_i^{(n-1)}||_2^2 \le ||\mathbf{x} - \mathbf{y}_j^{(n-1)}||_2^2, i \neq j\}, i = 1, ..., K$$

  3. Compute the new codebook with respect to the Centroid Condition

     $$\mathbf{y_i^{(n)}} = \int_{I_i^{(n)}} \mathbf{x}\, f_{X|\mathbf{x}\in I_i^{(n)}}(\mathbf{x}|\mathbf{x} \in I_i^{(n)}) d\mathbf{x}, i = 1, ..., K$$

  4. Compute the distortion

     $$D^{(n)} = \sum_{i=1}^{K} \int_{I_i} ||x - y_i^{(n)}||_2^2\, f_X(x) dx$$

5. If the stop condition:

$$\frac{D^{(n-1)} - D^{(n)}}{D^{(n)}} < \epsilon \text{ then stop, else set n = n + 1 and go to step number 2.}$$

In this situation, there is no guarantee that the algorithm reaches global minimum of the distortion, and it can be stuck on a local minimum. It depends on the initial codebook.

- *The distribution is unknown*: This situation is a more practical version where we have a training set.

1. Given training set τ, start with an initial codebook $\{\mathbf{y}_{\mathrm{i}}{}^{(0)},...,\mathbf{y}_{\mathrm{K}}{}^{(0)}\}$, set n=1, $D^{(0)} = \infty$. Select threshold $\epsilon > 0$.
2. Find the optimal decision regions

$$I_i^n = \{\mathbf{x} \in \tau \text{ such that } ||\mathbf{x} - \mathbf{y}_i^{(n-1)}||_2^2 \leq ||\mathbf{x} - \mathbf{y}_j^{(n-1)}||_2^2, i \neq j\}, i = 1, ..., K$$

3. Compute the new codebook

$$\mathbf{y_i^{(n)}} = \frac{1}{|I_i^n|} \sum_{\mathbf{x} \in I_i^{(n)}} \mathbf{x}$$

4. Compute the average distortion between training vectors and reconstruction values

$$D^{(n)} = \frac{1}{|\tau|} \sum_{\mathbf{x} \in \tau} ||\mathbf{x} - Q(x)||_2^2$$

5. If the stop condition:

$$\frac{D^{(n-1)} - D^{(n)}}{D^{(n)}} < \epsilon \text{ then stop, else set n = n + 1 and go to step number 2.}$$

2.3 MATLAB Implementation

There are five Matlab scripts that have been used in this project, which are briefly presented below:

- *main_lbg.m* : This is the main script to be run. It loads the audio signal with the help of other functions mentioned below. It creates an L-dimensional set in line with the LBG algorithm over the loaded audio signal and trains the Voronoi regions. It shows how many iterations were made during the operation of the LBG algorithm and the SNR value in the relevant iteration.

- *load_audio.m* : This script calls *audioread*, which is Matlab's internal function. The function takes the audio file path in the working folder as input and outputs the signal x and the sampling frequency F and the number of bits per sample for the given input.

- *TrainVoronoiRegion.m* : It is the script used for the calculation and training of Voronoi regions. It takes the training set and randomly generated codebook as input. It calculates the distance of the codevector from each of the L dimensional vectors in the training set using the L2-norm.

- *rate_snr.m* : This script takes the necessary variables in the workspace as input so that we can see the Ratio-SNR graph and draw the related graph as output.

- *visualize_audio_signals.m* : This script visually gives the time-amplitude output of the audio signals to be input to the algorithm.

3 Results

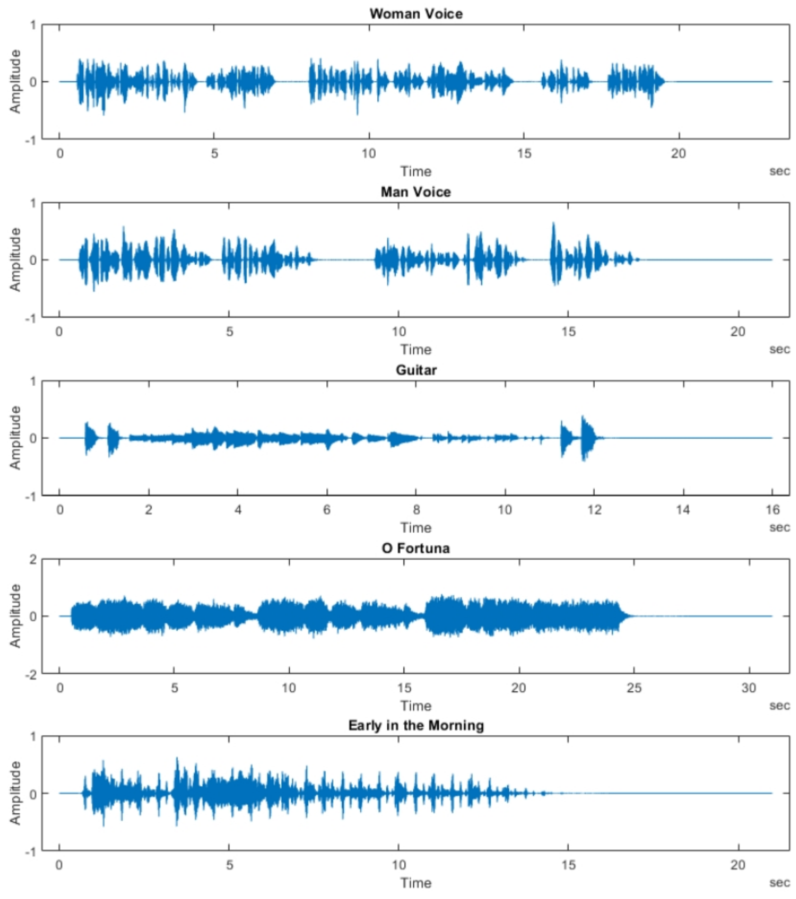

Figure 2: Input Signals

All audio files used in the project files contain different types of recordings (speech, music, etc.), these are listed below:

1. *Woman Voice:* This is a test audio file for Google Cloud Speech-to-Text API
2. *Man Voice:* A male voice reciting a word of Epictetus in German.
3. *Guitar:* A part of Spanish Guitar Zapataedo by Larry Coryell Pablo de Sarasate
4. *O Fortuna:* A song by Carl Orff
5. *Early in the Morning:* A song by Eddie Rabbitt

In order to have some idea about the audio files, each sound file is plotted in Figure 2.

Audio Name	L	K	R bit/sample	SNR_{dB}
49 - Woman Voice	2	16	2	15,1905
54 - Man Voice	2	16	2	17,0271
58 - Guitar	2	16	2	14,2228
64 - O Fortuna	2	16	2	15,2877
70 - Early in the Morning	2	16	2	16,9189

Table 1: SNR values for L=2, R=2

Audio Name	L	K	R bit/sample	SNR_{dB}
49- Woman Voice	2	256	4	26,6804
54- Man Voice	2	256	4	28,0228
58- Guitar	2	256	4	25,9915
64- O Fortuna	2	256	4	26,0334
70- Early in the Morning	2	256	4	27,3835

Table 2: SNR values for L=2, R=4

3.1 Results for L=2 and varying R

In this section, the results of compression using the LBG algorithm with vector dimension L=2 are presented fixing a rate R between 2 and 4 bits per component. Also, the epsilon value is fixed at $\epsilon = 0.1$

Each audio signal was encoded by training its codebooks on its own audio file. Each audio signal was encoded by training its codebooks on their own names. Intuitively, we would expect good results in terms of SNR from the signal encoded in this way. This is because the optimal codebook is built on the same signal.

As shown in Table 1 and Table 2, when we increase the number of bits per component R while the L-dimensional vector is constant, the number of codevectors in the codebook K and the SNR value increases in dB. This means it needs higher rates for higher SNR values. Generally, it can be noticed that the Man Voice signal has a higher SNR value. While the closest to the Man Voice is Early in the Morning, the difference with the others is about one decibel each.

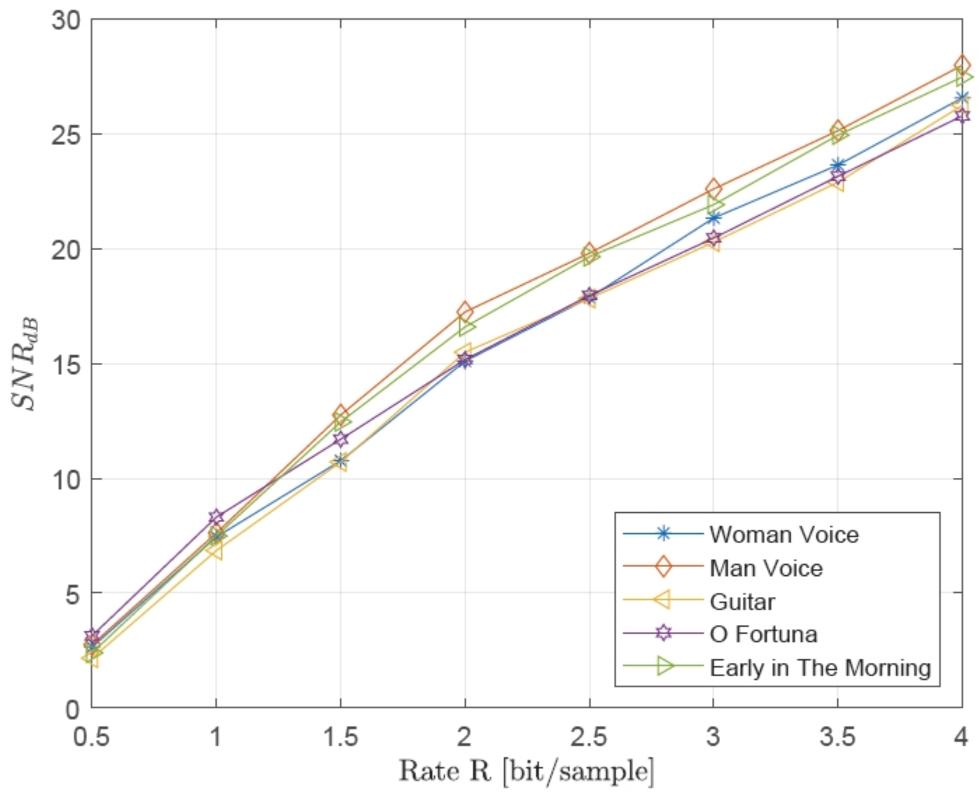

Figure 3: Rate-SNR graph for L=2

From the Figure 3 Rate-SNR graph, SNR values for L=2, corresponding to the R values of all audio files can be seen. Man Voice and Early in the Morning catch the high SNR values after R = 1.5 and take this state up to R = 4, followed by Woman Voice and then Guitar and O Fortuna.

Audio Name	L	K	R bit/sample	SNR_{dB}
49- Woman Voice	4	16	1	7,7739
54- Man Voice	4	16	1	7,6138
58- Guitar	4	16	1	7,1345
64- O Fortuna	4	16	1	8,3071
70- Early in the Morning	4	16	1	7,2948

Table 3: SNR values for L=4, R=1

Audio Name	L	K	R bit/sample	SNR_{dB}
49- Woman Voice	4	256	2	14,8176
54- Man Voice	4	256	2	17,4148
58- Guitar	4	256	2	15,7194
64- O Fortuna	4	256	2	15,2859
70- Early in the Morning	4	256	2	15,0839

Table 4: SNR values for L=4, R=2

3.2 Results for L=4 and varying R

In this section, the results of compression using the LBG algorithm with vector dimension L=4 are presented fixing a rate R between 1 and 2 bits per component. Also, the epsilon value is the same as the previous scenario and fixed at $\epsilon = 0.1$

The first thing noticed when looking at Table 3 and Table 4 is the decrease in SNR values compared to Table 1 and Table 2. For each audio signal, its own audio file was used as the training set as in the previous scenario. The results are shown in Table 3 and Table 4. Compared with Table 1 and Table 2, SNR values were always lower than in the previous scenario. This is because codewords are larger than in the previous scenario, so we increase the quantization noise. However, the rates were kept lower in this scenario as well. Different scenarios can be derived according to the desired application type. For this case (L = 4), by increasing the rate higher than 2 i.e R = 4, computation will become quite complex (2^K=65536).

Audio Name	L	ϵ	K	R bit/sample	SNR_{dB}			ϵ	SNR_{dB}
49- Woman Voice	2	0.001	16	2	14,7532			0.005	12,6969
49- Woman Voice	2	0.001	256	4	26,5479			0.005	26,3741
54- Man Voice	2	0.001	16	2	17,5149			0.005	16,7966
54- Man Voice	2	0.001	256	4	27,7766			0.005	27,5573

Table 5: SNR values for L=2 ϵ=0.001 and ϵ=0.005

Audio Name	L	ϵ	K	R bit/sample	SNR_{dB}			ϵ	SNR_{dB}
49- Woman Voice	2	0.01	16	2	13,8950			0.05	14,3906
49- Woman Voice	2	0.01	256	4	25,5995			0.05	24,0710
54- Man Voice	2	0.01	16	2	16,3758			0.05	12,6409
54- Man Voice	2	0.01	256	4	27,2588			0.05	24,6280

Table 6: SNR values for L=2 ϵ=0.01 and ϵ=0.05

3.3 Varying ϵ value

Until now, the epsilon value was always kept at $\epsilon = 0.1$. In this section, the effects of changing the epsilon value on the results for SNR value will be seen. As can be seen in Table 5, 6, 7, 8, 9 and 10, the SNR value changes depending on the epsilon value. We can see from the table that when we decrease the ϵ value, the SNR value increases, and when we increase the ϵ value, the SNR value decreases. We can explain this situation as follows, as the epsilon value decreases, we try to get closer to the codevectors by performing more iterations, and as a result, our SNR value increases.

Audio Name	L	ϵ	K	R bit/sample	SNR_{dB}			ϵ	SNR_{dB}
49- Woman Voice	2	0.1	16	2	13,6227			0.5	11,4179
49- Woman Voice	2	0.1	256	4	23,1959			0.5	20,3192
54- Man Voice	2	0.1	16	2	13,5435			0.5	7,0023
54- Man Voice	2	0.1	256	4	21,9345			0.5	21,4918

Table 7: SNR values for L=2 ϵ=0.1 and ϵ=0.5

Audio Name	L	ϵ	K	R bit/sample	SNR_{dB}			ϵ	SNR_{dB}
49- Woman Voice	4	0.001	16	1	7,3951			0.005	7,3531
49- Woman Voice	4	0.001	256	2	14,7338			0.005	13,6418
54- Man Voice	4	0.001	16	1	7,6098			0.005	7,5152
54- Man Voice	4	0.001	256	2	17,5172			0.005	15,6983

Table 8: SNR values for L=4 ϵ=0.001 and ϵ=0.005

Audio Name	L	ϵ	K	R bit/sample	SNR_{dB}			ϵ	SNR_{dB}
49- Woman Voice	4	0.01	16	1	7,1687			0.05	7,3476
49- Woman Voice	4	0.01	256	2	14,2724			0.05	12,5792
54- Man Voice	4	0.01	16	1	7,8798			0.05	7,4442
54- Man Voice	4	0.01	256	2	16,8722			0.05	13,0573

Table 9: SNR values for L=4 ϵ=0.01 and ϵ=0.05

Audio Name	L	ϵ	K	R bit/sample	SNR_{dB}			ϵ	SNR_{dB}
49- Woman Voice	4	0.1	16	1	7,3109			0.5	7,0204
49- Woman Voice	4	0.1	256	2	13,3804			0.5	10,8122
54- Man Voice	4	0.1	16	1	7,5570			0.5	6,7072
54- Man Voice	4	0.1	256	2	15,6911			0.5	10,6776

Table 10: SNR values for L=4 ϵ=0.1 and ϵ=0.5

4 Conclusion

Finally, in this report, the application of the LBG algorithm to CD-Quality audio files using different parameters and the performance analysis of this algorithm are discussed. When we listen to audio signals processed with the LBG algorithm, we realize that it does not sound bad. We can also tell this result by looking at the SNR results. Despite these SNR values, the algorithm is computationally demanding and the codebook must be sent to the receiver side each time the codebook is updated. It is obvious that this algorithm cannot be used as a substitute for well-known algorithms for audio compression (MP3, WMA or AAC).

References

[1] K. Sayood, Introduction to Data Compression. New York: Elsevier, 2009.

[2] Y. Linde, A. Buzo and R. Gray, ”An Algorithm for Vector Quantizer Design”, IEEE Transactions on Communications, vol. 28, no. 1, pp. 84-95, 1980. Available: 10.1109/tcom.1980.1094577.

[3] G.Calvagno, ”Multimedia Coding Lecture Notes”. Padova, 2020.