Multimedia Coding, Project 7

LBG quantization, analysis of three proposed implementations of the Splitting Technique

Daniele Foscarin

13 January 2021

Contents

1 Abstract **2**

2 Theoretical approach **2**
2.1 LBG Algorithm . . . 2
2.2 Splitting Technique and Empty Cell Problem . . . 3
2.3 Proposed variants . . . 3
2.3.1 Regularized Random Perturbation . . . 4
2.3.2 Std Normalized Random Perturbation . . . 4
2.3.3 Std Scaled Random Permutation . . . 5

3 Matlab implementation **5**

4 Performances Analysis **6**
4.1 Test set . . . 6
4.2 Testing the algorithms . . . 7
4.2.1 Examples of reconstructed images . . . 7
4.2.2 Distortion analysis . . . 7

5 Conclusions **20**
5.1 Possible improvements . . . 20

1 Abstract

Vector quantization algorithms are the foundations of many data compression techniques. We wrote a quick theoretical review of the Linde-Buzo-Gray (LBG) Algorithm and then we performed the analysis on three versions of the Splitting Technique that we propose.
We tested a Matlab implementation for the quantization of RGB images from 24 bits/pixel to 8 bits/pixel with three different splitting algorithms. Then we compared their distortion against the Matlab JPEG implementation.
We concluded that we can build a well performing LBG quantization if we exploit the statistical properties of the dataset during the splitting process.

2 Theoretical approach

2.1 LBG Algorithm

In a vector quantization algorithm there are two conditions that assure the non-increasing behaviour of the clustering distortion over the iterations. Those are necessary conditions for the optimality of the result.

- Nearest Neighbour condition. Given a set of codevectors $\mathbf{y_i}$, the optimal partition I is the one that minimize the distance between the observations $\mathbf{x}$ and the codevectors.

$$I_i = \{\mathbf{x} \quad s.t. \quad ||\mathbf{x} - \mathbf{y_i}||_2^2 \leq ||\mathbf{x} - \mathbf{y_j}||_2^2, \quad i \neq j\}. \tag{1}$$

- Centroid Condition. Given a partition I, the set of codevectors $\mathbf{y_i}$ that minimizes the distortion is

$$\mathbf{y_i} = \frac{\int_{I_i} \mathbf{x} f_X(\mathbf{x}) dx}{\int_{I_i} f_X(\mathbf{x}) dx}. \tag{2}$$

  We can notice that the centroid condition assumes the knowledge about the distribution of the observations $\mathbf{x}$, this is not the case in many practical situations where we would like to apply a vector quantization algorithm. So we can rewrite the centroid condition in order to adapt it for empirical data and we obtain

$$\mathbf{y_i} = \frac{1}{|I_i|} \sum_{\mathbf{x} \in I_i} \mathbf{x}. \tag{3}$$

At this point we can easily describe the Linde-Buzo-Gray Algorithm as the iterative application of those two necessary conditions for optimality.
The description of the LBG algorithm steps is the following.

1. Given a set of empirical data $T = \{\mathbf{x_1}, \mathbf{x_2} ... \mathbf{x_N}\}$, we consider an initial codebook $\{\mathbf{y_i^{(0)}}, \quad i = 1, 2...k\}$, initialize $n = 1$, $D^{(0)} = \infty$ and $\epsilon > 0$.

2. Compute the decision cells

$$I_i^{(n)} = \{\mathbf{x} \in T \quad s.t. \quad ||\mathbf{x} - \mathbf{y_i^{(n-1)}}||_2^2 \leq ||\mathbf{x} - \mathbf{y_j^{(n-1)}}||_2^2, \quad \forall i \neq j\} \tag{4}$$

3. Update the codebook

$$\mathbf{y_i^{(n)}} = \frac{1}{|I_i^{(n)}|} \sum_{\mathbf{x} \in I_i^{(n)}} \mathbf{x}. \tag{5}$$

4. Update the distortion

$$D^{(n)} = \frac{1}{|T|} \sum_{\mathbf{x} \in T} ||\mathbf{x} - Q(\mathbf{x})||_2^2. \tag{6}$$

5. If $\frac{D^{(n-1)} - D^{(n)}}{D^{(n)}} < \epsilon$ stop. Else update $n \leftarrow n + 1$ and go to step 2.

2.2 Splitting Technique and Empty Cell Problem

The choice of the initial codebook can greatly influence the final result, so in the literature there are many solutions that deal with this problem. The simplest solution is a random initialization of the codevectors in the space of the observations, or the initialization in a uniform grid. A more sophisticated idea is to perform an analysis of the data distribution and assign the initial codevectors in the more probable regions [1].
The initialization technique that was proposed in the original Linde-Buzo-Gray paper is called Splitting Technique, and we are going to focus on the variations of this.
Another phenomenon that occurs when the codebook is not optimally initialized is called Empty Cell Problem. That occurs when there are no points assigned to a cluster and it is evident how this can only worsen the performances of our vector quantizer. A solution for this problem is to replace the codevector of an empty cell to the position of a point that is member of another cluster.
The Splitting Technique proposed by Linde-Buzo-Gray consists in defining the initial codebook as only one vector $\mathbf{y}$ positioned in the centroid of the entire dataset. Then a perturbation is applied on the vector in order to obtain two codevectors $\mathbf{y_1}$, $\mathbf{y_2}$ and the LBG optimization is performed in order to obtain two clusters with the codevectosr as their centroids. These steps are repeated until we obtain the wanted number of codevectors.

2.3 Proposed variants

While we can easily define the perturbation vector as a random vector, we observed that we still need to perform some choices about it that greatly influence the final outcome. We focused on the following ideas.

- We necessarily need to define the maximum length of our random perturbation vector.
- How is this length related to the dimensionality of the data?
- How is it related to the number of clusters we have in the current iteration of the splitting routine?
- How can its direction be related to the distribution of the data?
- Is it possible to define the perturbation vector in a deterministic way?

[1] k-means++: The Advantages of Careful Seeding, David Arthur,Sergei Vassilvitskii, 2007

So in order to explore those ideas we have formulated and implemented three different way to define the perturbation vector $\mathbf{\Delta}$ and we tested their performances on a set of images and against a lossy JPEG implementation in order to have the comparison with a more refined compression technique.

- Regularized random perturbation:
$$\mathbf{\Delta} = \mathbf{r}k^{-\frac{1}{L}}, \tag{7}$$
where $\mathbf{r}$ is a random vector with $||\mathbf{r}|| = 1$, k is the number of codevectors after the split, L is the dimensionality of the data.

- Std normalized random perturbation:
$$\mathbf{\Delta} = \frac{\sigma_T}{||\sigma_T||} \odot \mathbf{r}k^{-\frac{1}{L}}, \tag{8}$$
where $\odot$ is the element wise multiplication, σ_T is the standard deviation of the dataset T, $\mathbf{r}$ is random vector with $||\mathbf{r}|| = 1$, k is the number of codevectors after the split, L is the dimensionality of the data.

- Std scaled rand perturbation:
$$\mathbf{\Delta} = \alpha\sigma_T \odot \mathbf{r}k^{-\frac{1}{L}}, \tag{9}$$
with the scale factor $\alpha \in \mathbb{R}^+$, $\odot$ is the element wise multiplication, σ_T is the standard deviation of the dataset T, $\mathbf{r}$ is random vector with $||\mathbf{r}|| = 1$, k is the number of codevectors, L is the dimensionality of the data.

In the following sections there are more precise considerations about those definitions.

2.3.1 Regularized Random Perturbation

The rationale behind the term $k^{-\frac{1}{L}}$, that is present in each version, is the idea that when we have few clusters we want to apply a bigger perturbation while when we have many cluster we want a smaller one. The term $k^{-\frac{1}{L}}$ is in fact a decreasing function with the increasing of k, and we put the dimensionality L in the denominator of the exponent because in this way the length of $\mathbf{\Delta}$ will be always the distance from two points in a orthogonal grid that ranges in every dimension in the interval $[0, 1]$ with $k^{\frac{1}{L}}$ points in every edge.
For example, when we split 32 clusters in 64 clusters in a RGB images, so $k = 64$ and $L = 3$, the length of the vector $\mathbf{\Delta}$ will be $||\mathbf{\Delta}|| = (\frac{1}{64})^3$. In this way we control the length of the perturbation but its direction is still a uniform random distribution,

$$\mathbf{\Delta} = \mathbf{r}k^{-\frac{1}{L}}. \tag{10}$$

2.3.2 Std Normalized Random Perturbation

After this we can think about the way the distribution of the data can help us in the search for the best splitting technique. So if we think about a dataset that is heavily skewed towards a direction in the three dimensional space, we can try to control the direction of the perturbation in order to increase the probability that the splitted codevectors will lie in high density regions of the space.

So we can add a term that will modify the direction and length of $\mathbf{\Delta}$ according to the standard deviation of the observations. using σ_T as the standard deviation of the dataset T we obtain

$$\mathbf{\Delta} = \frac{\sigma_T}{||\sigma_T||} \odot \mathbf{r} k^{-\frac{1}{L}}, \tag{11}$$

where $\odot$ is the element wise multiplication. We can notice that, while $\mathbf{r}$ is a random vector with length 1 and so it spans a sphere in the 3d space, multiplying it element wise for the normalized standard deviation we obtain $\mathbf{\Delta}$ as a random vector that is parallel to σ_T. It has length equal to zero if $\mathbf{r}$ is orthogonal to σ_T, and it has length $k^{-\frac{1}{L}}$ if $\mathbf{r}$ is parallel to σ_T. Now the perturbation vector is still chosen randomly with the contribution of $\mathbf{r}$ but its direction is fixed .

2.3.3 Std Scaled Random Permutation

At the end we tried to change the range of lengths that $\mathbf{\Delta}$ can assume. Also it is interesting to let the σ_T vector control the length removing its normalization. In this case, if we consider the extreme case where $||\sigma_T|| = 0$, $\mathbf{\Delta}$ will have length equal to zero. That is coherent with our ideas, since it will be the case where all the observations are identical and any perturbation will increase the clustering distortion. Otherwise, if the dataset is sparse and the standard deviation vector has a bigger length, it is rational to desire a longer $\mathbf{\Delta}$ as well. The only problem is that since we are using observations that are normalized in the [0,1] range, σ_T can reduce too much the length of $\mathbf{\Delta}$, so we can try to apply a scaling factor α that increase the length of such vectors:

$$\mathbf{\Delta} = \alpha\sigma_T \odot \mathbf{r} k^{-\frac{1}{L}}. \tag{12}$$

In our Matlab implementation we chose $\alpha = 2$, since it showed good performances in our tests.

We tried to implement also a fully deterministic version, by avoiding the use of any random vector, but the performances of those deterministic implementation were consistently worse than the already described algorithms.

3 Matlab implementation

We used Matlab to write the scripts that define the LBG algorithms with the proposed variations and to tests their performances on different RGB images [2]. The code is contained in the following files:

- *LBGSplitRegRand.mlx, LBGSplitStdNorm.mlx, LBGSplitStdScaled* are the functions that implement an iteration of the LBG algorithm with the already described variants the splitting technique. So they receive a list of codevectors, they split them in order to double the number of codevectors and perform the LBG minimization of the clustering distortion. At every iteration they also deal with the Empty Cell Problem: for every empty cluster the cluster are sorted by their size and the codevector of the empty cluster is reassigned with the coordinates of the closest point the bigger cluster codevector plus the perturbation $\mathbf{\Delta}$. We need to notice that all the clustering operations are performed with the data normalized from uint8 in the range $[0, 255]$ to double in the range $[0, 1]$. They receive in input the coordinates of the

[2] Public Github repository: https://github.com/DanieleFoscarin/MCoding_project/tree/main

existing codevectors, the image data reshaped in a array where every row is a point in the 3d space and the array that stores the membership of each point. They return an array of codevectors that are twice the number of the input codevectors.

- *imageDistortion.mlx* receive in input two images array and returns the distortion value between them and also the image shaped array that is the per pixel distortion. The distortion is computed as the mean squared error

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

  We need to precise that the distortion computation is executed using already normalized data, so using points ranging in the interval [0,1].

- *LBG_v1.mlx* is the main script that read the images and reshape them in arrays where every row is a point in the 3d space corresponding to the description of a pixel. Then it calls one of the three LBGSplit functions iteratively 8 times. At first the codebook is just one single codevector corresponding to the centroid of the entire image array, and after the eighth iteration the codebook contains 256 codevectors.
  The input image gets encoded by assigning at every pixel the closest codevector and storing its index in a array, so the encoded image has a rate of 8 bits/pixel. In this implementation we considered the compression of every image with a personalized codebook, so in the ipothetical transmission of this image we should add the size of the codebook. The codebook is composed by 256 codevectors of size 24 bits. So the final bitrate, considering the transmission of the codebook is 8.0703bits/pixel when using an image of size 512x512 pixels.
  The decoding is executed by replacing the codevector correspondent to the index stored in the encoded image.
  At the end we use the Matlab implementation of the JPEG compression algorithm to compare its distortion against the distortion obtained with the LBG algorithm. Since this implementation of JPEG can be used with a range of quality from 0 to 100 where 100 is the lossless compression, we decided to use the quality value 50 as a reference.

- *LBG_tester.mlx* is a refactored version of *LBG_v1.mlx* that execute the same operations but in three nested for cycles. That allows us to compute quantization, encoding and decoding of every image using every one of the three variants with many number of runs, and store the distortion for every run. We need to execute the quantization many time in order to reduce the effect of randomness on our evaluations about the performances of every algorithm.

- *Result_visualization.py* is a simple python script that generates and exports the figures regarding the distortion analysis on the different algorithms. The errorbar plots are generated using the *pyplot* package by *matplotlib*.

4 Performances Analysis

4.1 Test set

We chose eight not compressed png images as a test set. They are frequently used test images for image processing like Lena and the baboon image, or other images provided by Matlab as examples

images. We also used an image of a blue-green-magenta gradient with the aim to be able to easily observe the color blocks produced by the quantization process.
The images of the test set are displayed in Figures 1-8.

4.2 Testing the algorithms

We ran the *LBG_tester* script in order to test every image ten times for every one of the three LBG variants, we also performed the test using the JPEG Matlab implementation with the quality value set to 50. For the StdScaled algorithm we used the scale factor $\alpha = 2$.

4.2.1 Examples of reconstructed images

We can see an example of every image reconstructed using our proposed variants of LBG and using JPEG (Figure 9-40).

We notice that, generally speaking, we cannot find outstanding differences in the three versions, and it is not simple to judge the quality of those algorithms by eye. About this we can still consider two exceptions in the lighthouse image, where the RandReg version gives a noticeable worse quantization (Figure 33), and the reconstructions of the *gradient* image (Figure 21-23). This image in fact is the one that gives the least satisfying reconstruction, but it is expected since a clustering algorithm based on a fixed number of centroids struggles to classify data that have a uniform distribution, and this is the case for this image. Anyway we can notice that using the StdScaled algorithm we obtain smaller color blocks, that correspond also to the quantization levels, and for a very uniformly distributed set of points, like the pixels in the *gradient* image, this intuitively leads to a smaller distortion.

4.2.2 Distortion analysis

We need to compare the distortion values of the four compression algorithms we used in order to determine which approach is the best performing, and also to understand if their performances are heavily influenced by the random choice of the perturbation direction, or they produce consistent results regardless of their random component.

We used the data extracted by *LBG_tester.mlx* to find the mean and the standard deviation for each algorithm applied to every image (Figure 41-48). Notice that since the JPEG algorithms is instead deterministic, its standard deviation is equal to zero.

The most interesting result that we see in the error bar plots is that, when we consider the three version of LBG we proposed, StdScaled is the best performing algorithm in 7 images, and RandReg is the worst performing one in 7 images, too. That means that the use of the standard deviation of the points of the entire image is a useful term that allows us to increase our control over the direction and length of the perturbation during the splitting process, and doing so we obtain a significantly lower distortion over a random implementation. The only outlier image, where StdScaled is not the best performing algorithm, is *peppers1*. Also, in this image StdScaled produced very dishomogeneous results, how we can see looking at its standard deviation, while in the other images this algorithm was often the one with the most homogeneous results. It is not clear for us what is the substancial difference in this test image, that produce such results.

The distortion evaluated on the image compressed by JPEG was the highest one or at best similar to the distortion of RandReg, that is the worst from the LBG algorithms we tried. Anyway this comparison between JPEG and our implementations of LBG is not very interesting since we arbitrarily decided the quality value 50 on a scale from 0 to 100 at the beginning of our experimentation.

Figure 1: Lena

Figure 2: baboon

Figure 3: peppers1

Figure 4: gradient

Figure 5: peppers2

Figure 6: autumn

Figure 7: lighthouse

Figure 8: pears

Figure 9: Lena, LBG RandReg

Figure 10: Lena, LBG StdNorm

Figure 11: Lena, LBG StdScaled

Figure 12: Lena, JPEG quality 50

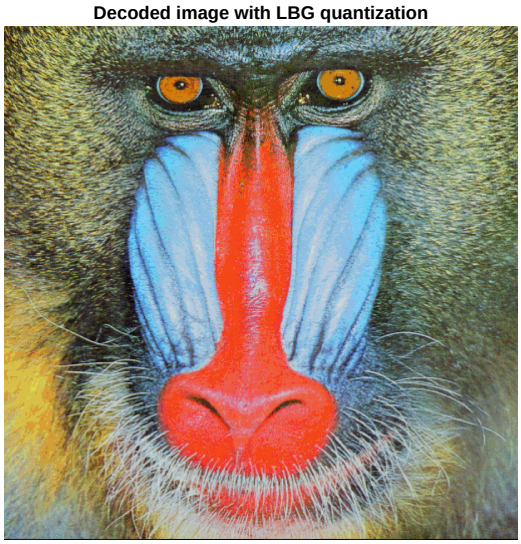

Figure 13: Baboon, LBG RandReg

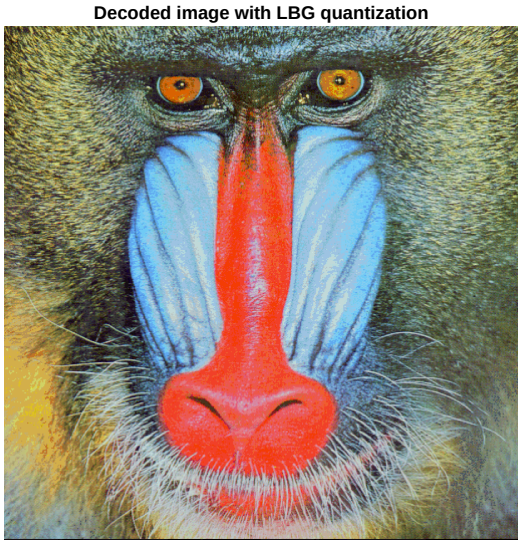

Figure 14: Baboon, LBG StdNorm

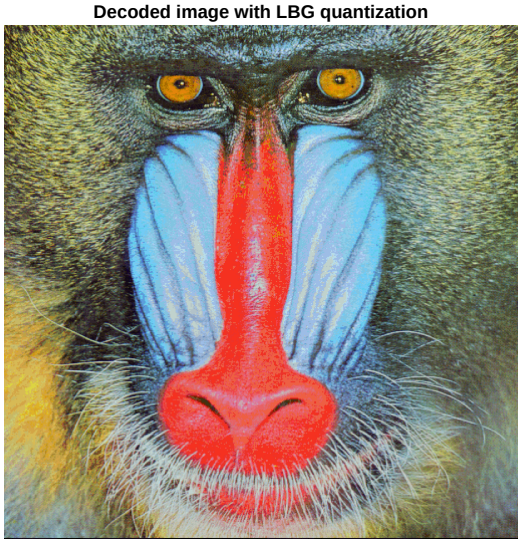

Figure 15: Baboon, LBG StdScaled

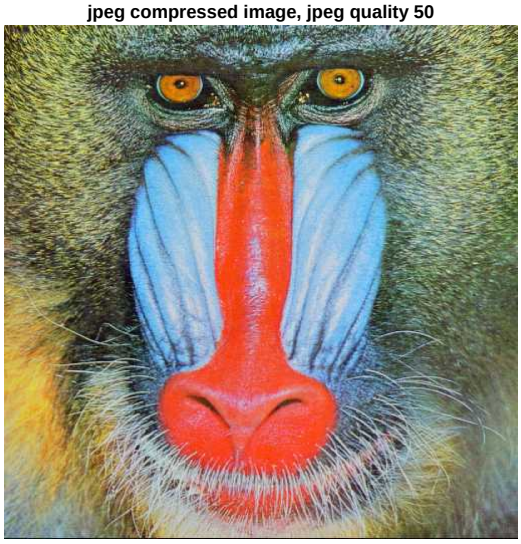

Figure 16: Baboon, JPEG quality 50

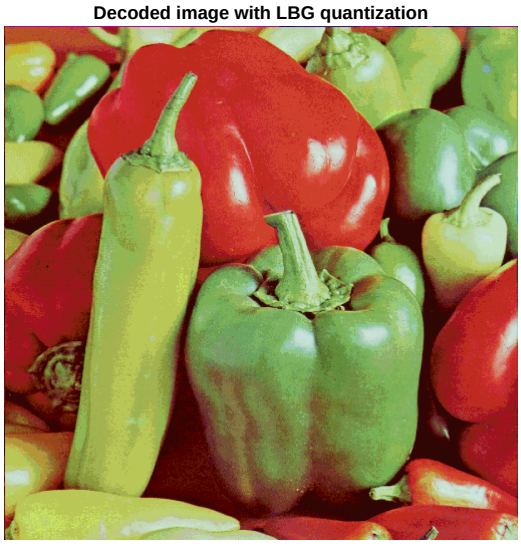

Figure 17: Peppers1, LBG RandReg

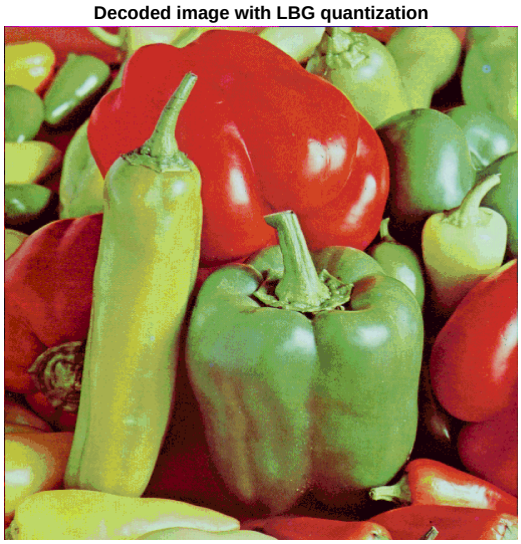

Figure 18: Peppers1, LBG StdNorm

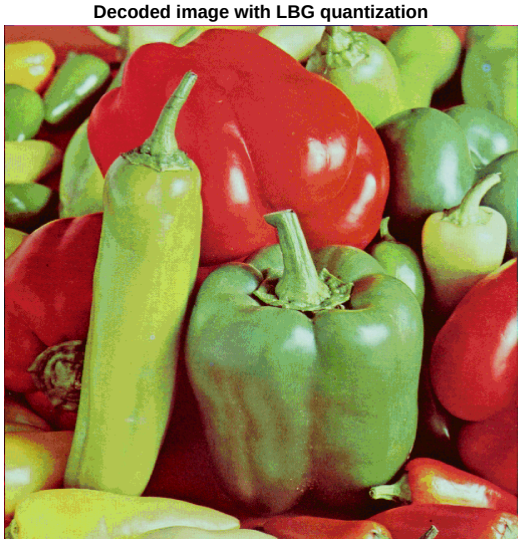

Figure 19: Peppers1, LBG StdScaled

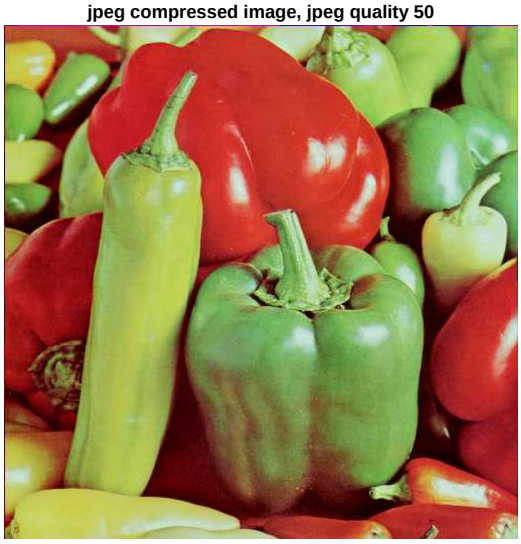

Figure 20: Peppers1, JPEG quality 50

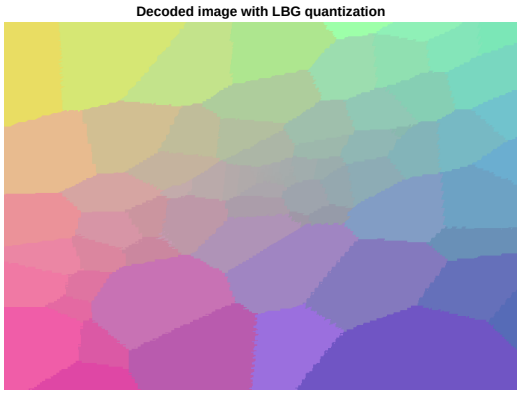

Figure 21: Gradient, LBG RandReg

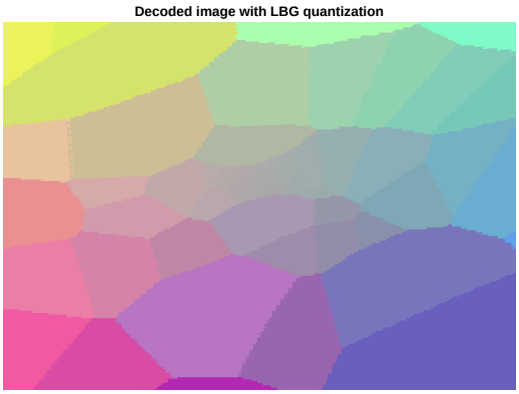

Figure 22: Gradient, LBG StdNorm

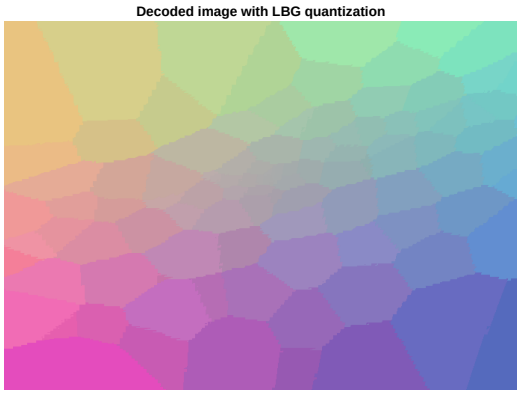

Figure 23: Gradient, LBG StdScaled

Figure 24: Gradient, JPEG quality 50

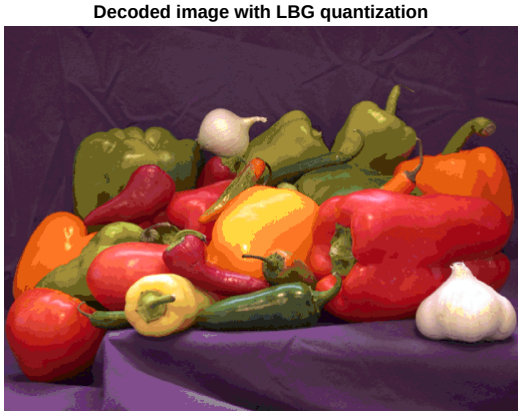

Figure 25: Peppers2, LBG RandReg

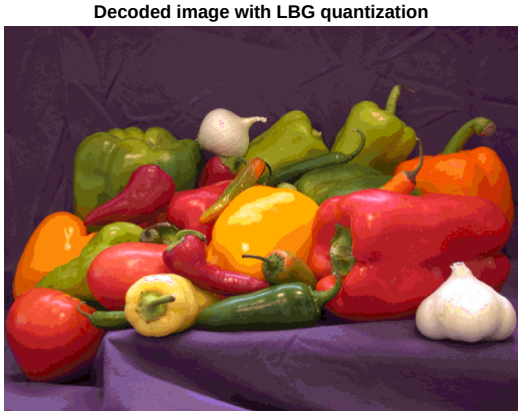

Figure 26: Peppers2, LBG StdNorm

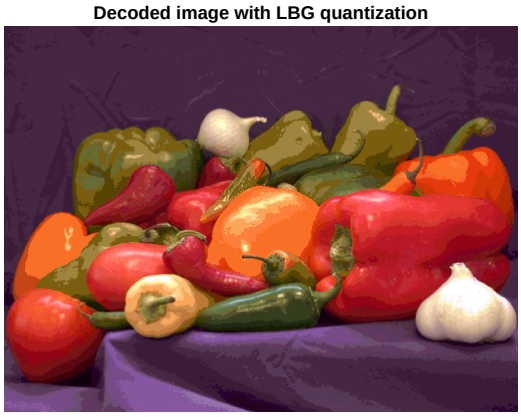

Figure 27: Peppers2, LBG StdScaled

Figure 28: Peppers2, JPEG quality 50

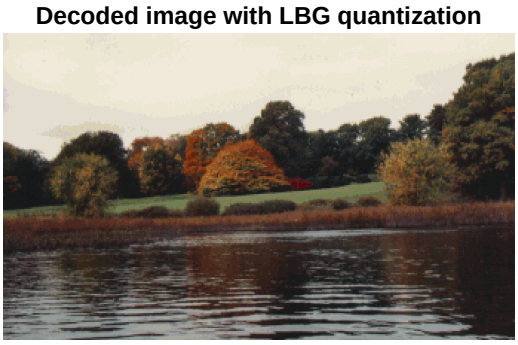

Figure 29: Autumn, LBG RandReg

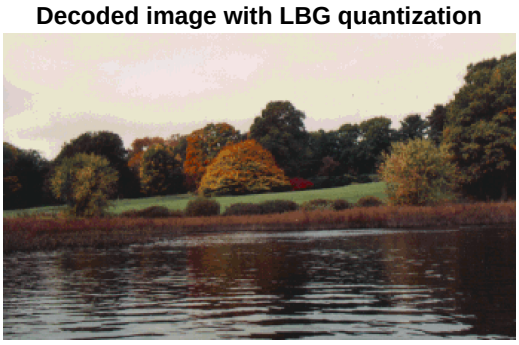

Figure 30: Autumn, LBG StdNorm

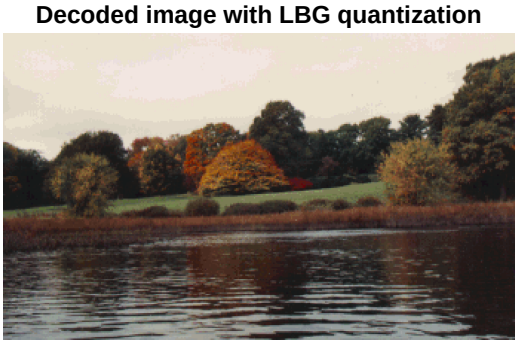

Figure 31: Autumn, LBG StdScaled

Figure 32: Autumn, JPEG quality 50

Figure 33: Lighthouse, LBG RandReg

Figure 34: Lighthouse, LBG StdNorm

Figure 35: Lighthouse, LBG StdScaled

Figure 36: Lighthouse, JPEG quality 50

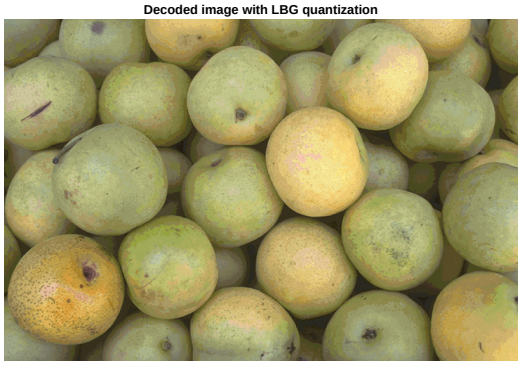

Figure 37: Pears, LBG RandReg

Figure 38: Pears, LBG StdNorm

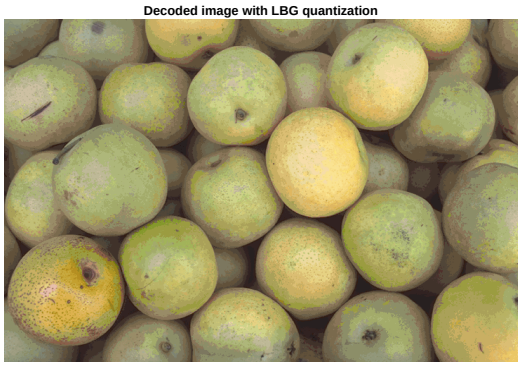

Figure 39: Pears, LBG StdScaled

Figure 40: Pears, JPEG quality 50

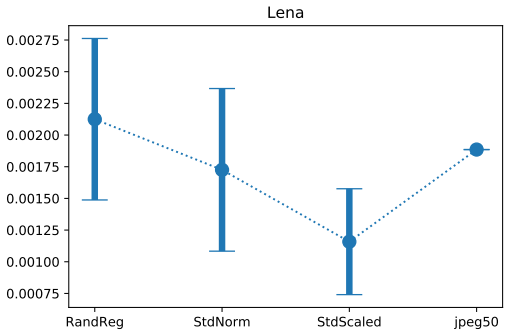

Figure 41: Distortion on Lena

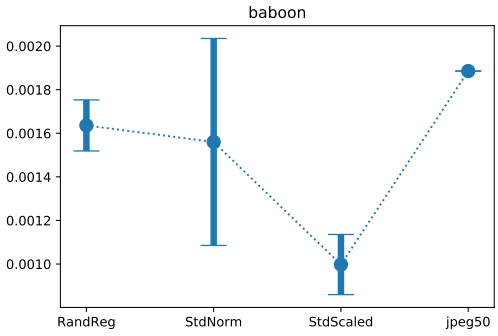

Figure 42: Distortion on baboon

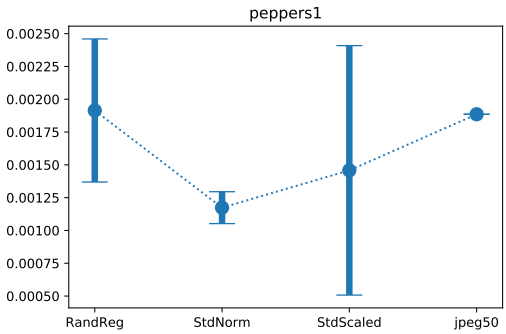

Figure 43: Distortion on peppers1

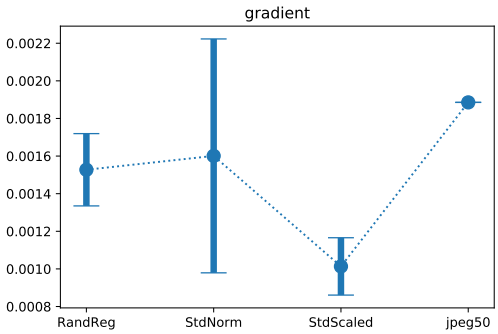

Figure 44: Distortion on gradient

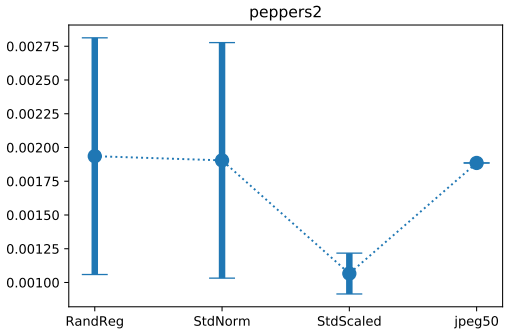

Figure 45: Distortion on peppers2

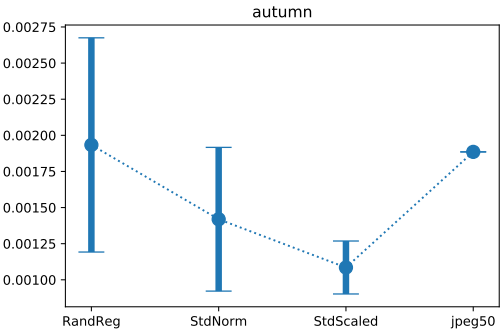

Figure 46: Distortion on autumn

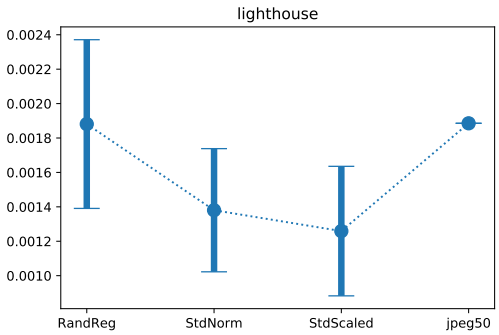

Figure 47: Distortion on lighthouse

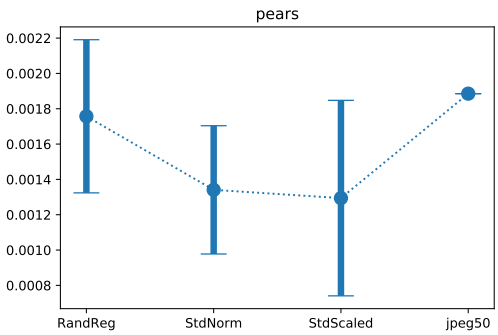

Figure 48: Distortion on pears

5 Conclusions

We proposed and implemented three version of the LBG quantization algorithm and we tested them for the compression of color images in order to obtain a codebook where every codevector has size 8 bits, that describes the color if each pixel in such images.
The three variants of the LBG focus on the definition of the perturbation vector that allows the splitting technique, a technique needed for the initialization of the codevectors during the clustering process described by the LBG algorithm.
We tried at first to regularize the length of such vector introducing a rule based on the number of clusters used and the dimensionality of the observations space. Then we tried to reduce the randomness of the perturbation vector using the standard deviation of the dataset as a way to control the direction and the length of the perturbation vector. In this way we described three way to define the perturbation vector which we called

- Regularized Random,
- Std Normalized,
- Std Scaled.

At the end we computed the distortion on the reconstructed images and we compared it against a JPEG implementation.
After the analysis of the distortions produced by the different algorithms we can confidently affirm that the Std Scaled variation of the LBG algorithm is the one producing the least amount of distortion on the reconstructed images.

5.1 Possible improvements

The use of the standard deviation of the dataset as the metrics used to control the perturbation error in the Std Normalized and Std Scaled algorithm has bee chosen because of its computational simplicity and intuitiveness. It is probable that other more complex description of the distribution or statistical properties of the dataset would improve the performances.
Another approach could be the use of such description on every cluster instead of the whole dataset. This approach has been briefly explored during the development of this project but it has not produced satisfying results.
Focusing on the winner Std Scaled algorithm, it would be interesting to fine tune the scale parameter, or to define this parameter using a statistical property of the dataset.
The final aim of our experimentation was to define a fully deterministic process, so a more detailed analysis on why the elimination of the random chosen component r leads to bad performances is also the first step for other improvements.