Engineering Applications of Artificial Intelligence 79 (2019) 142–158

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Color image quantization using the shuffled-frog leaping algorithm☆

María-Luisa Pérez-Delgado

University of Salamanca, Escuela Politécnica Superior de Zamora, Av. Requejo, 33, C.P. 49022, Zamora, Spain

ARTICLE INFO

Keywords:
Swarm-algorithms
Color quantization

ABSTRACT

Swarm-based algorithms define a family of methods that consider a population of very simple individuals that cooperate to solve a difficult problem. The Shuffled frog-leaping algorithm is a method of this type that has been applied to solve different types of problems. This article describes the application of this algorithm to the color quantization problem. Although the selected method was developed to solve optimization problems, this work shows how it can be adapted to solve the problem proposed. The proposed method uses the mean squared error as the objective function of the optimization problem to be solved. To reduce the execution time of the algorithm, it is applied to a subset of pixels of the original image. As a result, a quantized palette is obtained that is used to define the quantized image. Computational results indicate that the proposed method can generate a quantized image with low computational cost. Moreover, the quality of the image generated is better than that of the images obtained by several well-known color quantization methods.

1. Introduction

In recent years several artificial intelligence methods have been proposed to tackle difficult problems, of which swarm intelligence is included. Swarm-based algorithms are nature-inspired techniques based on the collective behavior of self-organized and decentralized systems (Kar, 2016). They consider a population of individuals (particles, birds, bees, ants, pollen grains among others) and imitate the behaviors observed in natural systems and apply them to solve problems. These methods have been applied to solve a wide variety of problems (Fister et al., 2014; Karaboga et al., 2014; Neshat et al., 2014; Mavrovouniotis et al., 2017; Shehab et al., 2017; Cheng et al., 2018).

The Shuffled frog-leaping algorithm (SFLA) is a swarm-based method designed to solve optimization problems (Eusuff and Lansey, 2003; Eusuff et al., 2006). This method is inspired by the behavior of frogs when looking for food, and combines the benefits of memetic algorithms and swarm-based algorithms. To solve a problem, a population of frogs is considered. Each frog represents a feasible solution to the problem, and the quality or fitness of the said solution is calculated based on the objective function of the problem. The initial population includes frogs randomly placed on the search space. According to the fitness of the frogs, the population is divided into several subsets, called memeplexes. The frogs in a memeplex evolve through a process of memetic evolution that allows them to perform a local search without taking into account the frogs in other memeplexes. During this process, the worst frog tries to leap to a position with more food (that is, to a better position in the search space). This movement takes into account the experience of this frog and the feedback coming from the best frog in the memeplex or the best frog in the population. If the new position is worse than the previous position, the frog moves to a random position. In this way, each memeplex determines a local solution to the problem during the memetic evolution process. After this, the frogs in all the memeplexes are brought together, allowing them to exchange information and to perform global evolution. The local search and shuffling processes are applied iteratively until a convergence criterion is satisfied.

Interest in the SFLA method is shown by the fact that many authors have analyzed their characteristics and applications (Sarkheyli et al., 2015). This algorithm has been applied to solve a variety of problems, such as water distribution networks design (Mora-Melia et al., 2016), neural network training (Tripathy et al., 2015), assembly line sequencing (Guo et al., 2015), tuning of PID controllers (Xiao et al., 2016), job-shop scheduling (Lei et al., 2017), clustering (Xunli and Feiefi, 2015), optimal power flow (Azizipanah-Abarghooee et al., 2014) and routing problems (Luo and Chen, 2014; Luo et al., 2015). In relation to image processing, this method has been applied to image segmentation (Ladgham et al., 2015a; Ma and Zhu, 2017; Wang et al., 2018), image threshold selection (Ladgham et al., 2015b), face detection (Torkhani et al., 2017) and image classification (Ladgham et al., 2014).

This article proposes a new application for SFLA in the field of image processing to solve the color quantization problem, which is an operation that attempts to reduce the number of colors in an image without the loss of quality or important global information. It is evident that currently existing devices can display images with many

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.engappai.2019.01.002.

E-mail address: mlperez@usal.es.

https://doi.org/10.1016/j.engappai.2019.01.002
Received 20 July 2018; Received in revised form 16 November 2018; Accepted 4 January 2019
Available online 22 January 2019
0952-1976/

colors, but other aspects of working with images, such as their storage or transmission, must be taken into account (both features are very important, for example, for reducing the loading time of web pages that include images). Certainly, the number of colors of an image also influences the storage space it occupies. Therefore, if the number of distinct colors of an image is reduced, this can also reduce the size of the file that contains the image, which is advantageous for the storage (less space) and transmission (more velocity) of that image (Zain Eldin et al., 2015; Alqudami and Kim, 2016; Kekre et al., 2016; Rabie, 2017). On the other hand, color quantization is also related to other image processing operations such as texture analysis (Ponti et al., 2016), segmentation (An and Pun, 2014; Fu et al.) and content-based image retrieval (Liu et al., 2015; Prasad et al., 2016).

The rest of the paper is organized as follows: first, the color quantization problem is defined and several interesting solution methods are described; then, the operations of the SFLA algorithm are presented. Section 4 describes the method proposed in this article for performing color quantization using the SFLA algorithm and Section 5 includes the computational results. Finally, the conclusions are presented.

2. The color quantization problem

2.1. Problem definition

Let us consider an image with n pixels, organized in a rows and b columns. In the RGB color space, each pixel p_i, with $i \in [1, n]$, is characterized by three values in the range of 0 to 255, corresponding to the intensity of red, green and blue: $p_i = (R_i, G_i, B_i)$. The color palette used to represent the image has more than 16 million colors.

The quantization of images is a process that includes two operations. The initial operation consists of defining a quantized palette that includes a reduced number of colors, q. The next operation uses the colors of that palette to represent the original image thus obtaining the quantized image.

The quantized palette includes q elements, $\{c_1, \ldots, c_q\}$, where each element is an RGB color, $c_j = (R_j, G_j, B_j)$. To define the quantized image, each pixel p_i of the original image is replaced with a pixel p'_i whose color is one of the colors of the quantized palette.

2.2. Color quantization methods

Garey et al. (1982) demonstrated that finding the optimal quantized palette is an NP-complete problem. As a consequence, several heuristic methods have been proposed to solve the color quantization problem. Those methods can be classified into splitting methods and clustering-based methods. The methods of the second type can obtain better images, but are more time consuming.

Splitting methods apply a recursive division of the color space until q regions are obtained. When the process ends, the quantized palette includes a representative color of each region. Some methods which apply this technique are Median-cut (Heckbert, 1982), Variance-based method (Wan et al., 1990), Octree (Gervautz and Purgathofer, 1990), Binary splitting (Orchard and Bouman, 1991), and Wu's methods (Wu, 1991, 1992).

The Median-cut method divides the color space into rectangular boxes. Each iteration selects the box that includes the largest number of pixels and divides it along the longest axis at the median point. When the process ends, the centroid of each box defines a color of the quantized palette. Each color of this palette represents approximately the same number of pixels in the original image. The colors of the original image associated with fewer pixels are not well represented in the final image.

The Variance-based method applies the same idea as Median-cut, but the selected box is the one with the largest weighted variance. The splitting axis is the one with the least weighted sum of projected variances and the splitting point is the one that minimizes the marginal squared error. Since this method takes into account the actual distribution of colors in the original image, the quality of the quantized image improves. Nevertheless, the operations performed are more time-consuming.

The Octree method uses a tree structure to define the quantized palette. Each node of the tree can have 8 children and only needs 8 levels to store the colors of an RGB image. During the first step of this algorithm, the pixels of the original image are used to build the tree. Then, an iterative process merges the leaves that represent fewer pixels; this process concludes when the number of leaves reduces to q. The quantized image is worse when the original image includes similar colors in general, but has many different low-frequency colors or noise.

The Binary splitting method builds a binary tree whose leaves represent subsets of the pixels of the original image. Each subset is represented by the average color of all the pixels in that subset. At the beginning of the algorithm, the root node of the tree represents the set of all the pixels of the original image. Each iteration of the algorithm selects the leaf node with the largest distortion and creates two children of this node, in such a way that the pixels associated with the parent are divided into two subsets and each one is associated with a child. The algorithm ends when the tree includes q leaves, each one defining a color of the quantized palette. A disadvantage of this method is that it requires to compute the principal axis each time a leaf is split.

The Greedy orthogonal bipartitioning method proposed by Wu is similar to the Variance-based method, but in this case the selected box is divided along the axis that minimizes the sum of the variances of both sides (Wu, 1991). This is a very fast color quantization method because the values used to perform the splitting process are computed once before this process starts. The same author proposed another method that applies dynamic programming (Wu, 1992). This method sorts the colors along their principal axis and divides the color space with respect to this ordering; the resulting constrained optimization problem is solved by dynamic programming. This method generates better quantized images than Median-cut and Variance-based methods, but it consumes more time.

Clustering-based methods divide the pixels of the image into clusters or groups according to the similarity of the pixels; each cluster is represented by a single color in the quantized palette. The K-means method, the artificial neural networks or the swarm-based algorithms are some of the clustering methods that have been applied to perform color quantization.

The K-means algorithm is a very popular clustering method that distributes a set of points among k independent groups or clusters, k being a predefined value. The method selects k initial centroids and applies an iterative process to improve them. Each iteration performs two operations: the first one associates each point with the closest centroid, so that all the points associated with the same centroid define a cluster; and the second operation computes the centroids of the new clusters defined in the previous operation. The operations can be applied to a predefined number of iterations or can end when a predefined convergence is reached. Although this method is easy to implement, it is influenced by the initial centroids and is very time consuming. Applications of K-means to color quantization appear in Verevka and Buchanan (1995), Kasuga et al. (2000), Hu and Su (2008) and Celebi (2009, 2011).

The Fuzzy c-means algorithm applies the same idea as K-means, but in this case each point can belong to more than one cluster. There is a set of membership levels associated with each point which indicates the strength of the association between that point and a particular cluster. It should be noted that Fuzzy c-means is slower than K-means. This method has been applied to color quantization in Özdemir and Akarun (2002), Schaefer and Zhou (2009) and Wen and Celebi (2011). As described for K-means, the results of this method also depend on the initial values.

The Neuquant method is the most representative application of neural networks for color quantization (Dekker, 1994). This method uses a Kohonen neural network with q neurons, which is trained with the

pixels of the original image; the final weights of the neurons define the quantized palette. When this method is applied to images with few colors, the general colors can become similar and the colors corresponding to fewer pixels are not represented in the final image. Other applications of neural networks for color quantization are described in Papamarkos et al. (2002), Chang et al. (2005), Wang et al. (2007), Rasti et al. (2011) and Palomo and Domínguez (2014).

Related to the swarm-based algorithms applied to color quantization, some interesting proposals appear in Omran et al. (2005), Ozturk et al. (2014), Ghanbarian et al. (2007) and Pérez-Delgado (2015). A literature review shows articles whose titles indicate they apply swarm algorithms to the color quantification problem, but these studies tend to focus on other image-related processing problems.

Omran et al. (2005) applied the Particle swarm optimization algorithm. This method is combined with the K-means algorithm, which is applied to each particle in a probabilistic way. In this case each particle of the swarm represents a quantized palette and its quality is computed by the mean squared error. The particles move on the search space and this movement improves the quantized palette they represent. When the process ends, the best particle in the swarm defines the palette that will be used to obtain the quantized image. The main disadvantage of this method is the execution time.

Ozturk et al. (2014) combine the Artificial bee colony algorithm with K-means in a similar way to that proposed by Omran et al., where the objective function is the mean squared error and K-means is applied in a probabilistic way, but artificial bees are used instead of particles. In this case, a set of food sources is considered, each of which represents a quantized palette. Operations are applied that simulate the behavior of honey-collecting bees, which allow the palettes to be improved. When such operations conclude, the best food source represents the solution to the problem. This method consumes more time than the Particle swarm optimization algorithm.

Ghanbarian et al. (2007) applied an ant-based clustering method to perform color quantization. First, the pixels of the original image are spread on a grid, and then a set of artificial ants moves these pixels. Each ant takes one pixel and moves it to another position on the grid where there are similar pixels. With these movements, the ants define a set of clusters, each of which includes similar pixels. The process of creating the clusters can be slow.

Pérez-Delgado (2015) proposed another method that applies artificial ants. In this case, each ant represents a pixel of the original image and the ants are organized in a tree structure according to the similarity of the pixels they represent. In this way, the pixels of each subtree define a cluster and are represented in the quantized image by the color of that subtree. The results of this method are influenced by the value of a parameter used to determine when an ant connects to the tree.

3. The shuffled frog-leaping algorithm

SFLA is inspired by the behavior of a group of frogs when they are seeking for food (Eusuff and Lansey, 2003; Eusuff et al., 2006). Two main behaviors are imitated: leaping and shuffling. A frog leaps to find a position that has more food than the current one, and then shuffles to exchange information. The algorithm considers a population of frogs, each representing a solution to the problem of interest. This population is divided into groups, called memeplexes, and the frogs of each group perform a local search. After this, the frogs of all the groups are shuffled so as they exchange information.

Let us consider an optimization problem defined on an r-dimensional space, with an objective function $f(x)$ associated with it. The solution to this problem is an element $x_j = (x_1^j, \ldots, x_r^j)$ of the search space that maximizes (or minimizes) the objective function.

To solve the optimization problem using the SFLA algorithm, a set of R frogs is considered. Each frog i has a position associated with it, $x_i = (x_1^i, \ldots, x_r^i)$, which represents a solution to the problem. The fitness or quality of such solution is computed by applying the objective function to the position of the frog: $fitness_i = f(x_i)$.

Algorithm 1 summarizes the operations of SFLA. The first operation places the frogs on random positions of the search space. Next, an iterative process is applied to a certain number of iterations T_{max} (the process could also stop when a predetermined error is reached). When the iterations end, the position of the frog with the best fitness defines the solution to the problem.

The first operation of each iteration sorts the frogs by decreasing fitness. Once the frogs have been sorted, they are divided into several memeplexes. Let m represent the number of memeplexes to be defined and $r_1, r_2, \ldots, r_R$ the list of frogs sorted by fitness. Such list is divided into consecutive sublists of size m: $(r_1, \ldots, r_m)$, $(r_{m+1}, \ldots, r_{2m})$, $\ldots$, $(r_{km+1}, \ldots, r_R)$; obviously, if R is not a multiple of m, the last sublist will contain less than m elements. Next, the frogs in the same position of each sublist are associated with the same memeplex: $r_1, r_{m+1}, \ldots, r_{km+1}$ are associated with the memeplex 1, $r_2, r_{m+2}, \ldots, r_{km+2}$ are associated with the memeplex 2, and so on.

The next operation of the algorithm (lines 6 to 21) applies a local search that considers the frogs of a memeplex independent of the remaining frogs in the population. An iterative process applies a certain number of iterations, J_{max}, to each memeplex in an attempt to improve the solution found by the frogs of that particular memeplex. This process first determines the best and the worst frog in the memeplex. Then, a candidate position is computed to move the worst frog to it. This candidate position, x'_{worst}, is computed by applying Eqs. (1) and (2), where ρ is a random value between 0 and 1, x_{best} is the position of the best frog in the memeplex and x_{worst} is the position of the worst frog in the memeplex. D is limited to take values between $-D_{max}$ and D_{max}, where D_{max} is the maximum amount of change allowed in a frog's position.

$$x'_{worst} = x_{worst} + D \tag{1}$$

$$D = \rho(x_{best} - x_{worst}) \tag{2}$$

If the candidate position has better fitness than the current position of the worst frog, then x_{worst} is replaced with x'_{worst}; otherwise another candidate position is computed by Eqs. (1) and (3). Eq. (3) has the same structure as Eq. (2), but in this case the best frog in the memeplex is replaced by the best frog in the population, whose position is denoted as x_g.

$$D = \rho(x_g - x_{worst}) \tag{3}$$

If the second candidate position computed for the worst frog in the memeplex has better fitness than its current position, the new value replaces the previous one, or else the worst frog moves to a random position in the search space.

The previous operations are applied to each memeplex independently. When all the memeplexes have been processed, the frogs of the m groups are combined (shuffled) to redefine a single group.

When SFLA has performed T_{max} iterations, the position of the frog with the best fitness, labeled x_g, defines the solution to the problem.

4. SFLA applied to solve the color quantization problem — SFLA-CQ

This paper proposes the application of the SFLA algorithm to perform color quantization. To distinguish the general algorithm from the proposal presented in this article, which is to solve the color quantization problem, they are identified as SFLA and SFLA-CQ, respectively.

SFLA-CQ considers a set of R frogs each representing a quantized palette; that is, the position of frog r is the palette $x_r = (c_1^r, \ldots, c_q^r)$, where c_j^r is an RGB color: $c_j^r = (R_j^r, G_j^r, B_j^r)$.

To compute the fitness or quality of the quantized palette represented by a frog, an objective function must be defined for the problem. In this case, the objective function considered is the mean squared error (MSE),

Algorithm 1 Shuffled frog-leaping algorithm

```
1: Initialize the swarm of frogs
2: for t = 1 to T_max do
3:    Compute the fitness of each frog
4:    Sort the frogs by fitness
5:    Create the m memeplexes
6:    for each memeplex do
7:       for j = 1 to J_max do
8:          Determine the best and the worst frog in this memeplex
            (denoted best and worst, respectively)
9:          Apply Eqs. (1) and (2) to compute a candidate position for
            the worst frog in this memeplex, x'_worst
10:         if x'_worst is better than the position of the worst frog then
11:            Take x'_worst as the new position of the worst frog
12:         else
13:            Apply Eqs. (1) and (3) to compute a candidate position for
               the worst frog in this memeplex, x'_worst
14:            if x'_worst is better than the position of the worst frog then
15:               Take x'_worst as the new position of the worst frog
16:            else
17:               Move the worst frog to a random position
18:            end if
19:         end if
20:      end for
21:   end for
22:   Shuffle the frogs of the memeplexes
23: end for
24: Take the frog with the best fitness as the solution to the problem
```

computed by Eq. (4), where p_i is the RGB color of a pixel of the original image and p'_i is the color of the pixel in the same position, but in the quantized image. This objective function must be minimized.

$$MSE = \frac{1}{n}\sum_{i=1}^{n} \|p_i - p'_i\|^2 \tag{4}$$

The MSE is a value commonly used in the color quantization literature to evaluate the quality of a quantized image (Celebi, 2009, 2011; Scheunders, 1997; Pérez-Delgado, 2015). Moreover, this error has also bee used in Omran et al. (2005) and Ozturk et al. (2014) to apply swarm-based methods to color quantization.

The input information for SFLA-CQ is the original image and the result is a quantized palette with q colors, in which case such palette must be used to generate the quantized image.

The structure of the SFLA-CQ algorithm is basically the same as that of the general SFLA algorithm. Nevertheless, some operations must be adapted to the problem of interest. In addition, SFLA-CQ includes some extra operations not included in SFLA. The next two subsections explain both types of operations in detail.

4.1. SFLA operations adapted for color quantization

The following operations of the general SFLA algorithm must be adapted to perform color quantization:

- the fitness of the solution (quantized palette) represented by a frog must be computed based on the objective function of the problem.
- the initialization step of the algorithm (line 1 of Algorithm 1) must define a set of initial quantized palettes.
- the selection of a candidate solution for improving the worst frog in a memeplex (lines 9, 13 and 17 of Algorithm 1) must define valid quantized palettes.

4.1.1. The fitness of a frog

To compute the fitness of a palette x_r using Eq. (4), it is first necessary to define the color p'_i used to represent the pixel p_i in the quantized image.

The operations applied to compute the fitness of the palette $x_r = (c^r_1, \ldots, c^r_q)$ are the following:

1. for each pixel p_i of the original image, the closest color of the palette x_r is determined. For this purpose, p_i is compared to the q elements of x_r, and the closest of such elements, j, is selected to represent p_i in the quantized image.
   This operation determines which pixels of the original image will be represented in the quantized image by the same color of the quantized palette.
2. each color c^r_j of the quantized palette is replaced with the average color of all the pixels associated with it in the previous step.
3. the fitness of x_r is computed by Eq. (4), where p'_i corresponds to the element j of the quantized palette associated with p_i in the first operation. Note that in the first step the element j of the palette was selected to represent to p_i in the quantized image, but the color c^r_j used in Eq. (4) is the new value computed in the second step.

It should be noted that before applying Eq. (4) to compute the fitness, each color of the quantized palette is replaced with the average color of all the pixels in the original image associated with that element of the palette. Computational results show that better images are obtained when the palette is recalculated in this way before calculating the fitness. If this operation is not applied, the execution time decreases, but the quantized image is worse.

To reduce the execution time of the SFLA-CQ algorithm, when the position of a frog is established, the fitness computed for such position is also stored, so that the fitness is recomputed only when a new position must be defined for the frog.

4.1.2. Initialization of the swarm of frogs

The first operation of the algorithm associates a palette with each frog. To define the initial palettes, first the initial position of each frog is defined by selecting q random pixels of the original image. Therefore, the initial position (palette) associated with frog r includes q colors, $x_r = (c^r_1, \ldots, c^r_q)$, where each element c^r_i is a random pixel of the original image.

Once the initial position of each frog has been defined, the fitness of this position is computed, since this value is used in the following operations of the algorithm. As described in the previous section, when the fitness of x_r is calculated, the colors of the palette are recalculated. Therefore, each random value c^k_i is replaced with the average color of all the pixels in the original image for which c^k_i is the closest color among all the elements in the palette.

4.1.3. Candidate palette for the worst frog in a memeplex

When a candidate position is determined for the worst frog in a memeplex, the result of applying Eq. (1) is always limited to taking values within $[0, 255]$ in order to use valid RGB colors.

On the other hand, when the worst frog must move to a random position (line 17 of Algorithm 1), the new palette is generated in the same way described in Section 4.1.2 to define each initial palette.

4.2. New operations of SFLA-CQ algorithm

SFLA-CQ algorithm includes new operations not included in SFLA:

- before applying the algorithm, the pixels of the original image are sampled in order to reduce the execution time of the method.
- the best frog is improved.
- when the algorithm ends, the best quantized palette is used to define the quantized image.

(a) Plane (b) Lake (c) Girl (d) Mandrill (e) Peppers (f) Lenna

(g) Landscape (h) Headbands (i) Dessert (j) Snowman (k) Cathedrals beach (l) Beach

Fig. 1. Images used for the tests.

4.2.1. Sampling of the original image

The size of the original image influences the execution time of the algorithm. If the n pixels of the image are considered when the algorithm is applied, better quantized images can be obtained, but with more computational effort. Therefore, the proposed algorithm considers the possibility of applying the operations to a subset of pixels of the original image. Instead of considering the n pixels of the image, in this case, a subset of n' pixels is selected with $n' < n$.

To take pixels of the whole image, they are sampled at a distance that is determined by the parameter *step*. Given the list of pixels that defines the successive rows of the original image, $\{p_1, \ldots, p_n\}$, the sampled image includes the pixels that are at a distance given by *step*: $\{p_1, p_{1+step}, p_{1+2step}, p_{1+3step}, \cdots, \}$.

The sampled image is used to compute the fitness of the palettes. On the contrary, the selection of a random palette (in the initialization step or when a random candidate position is selected for the worst frog) does not consider the sampled image. This operation considers all the pixels of the original image in order to avoid that the palettes could include very similar colors if the set is too small.

4.2.2. Improvement of a frog

The SFLA algorithm only tries to improve the worst solution (frog) in each memeplex. Nevertheless, the SFLA-CQ algorithm also tries to improve the best solution in the population, in order to obtain better results. This new operation is applied at the end of each iteration of the algorithm, after applying the shuffling operation (after line 22 of Algorithm 1).

The improvement applied consists of recomputing the fitness of the palette represented by the best frog. As described in Section 4.1.1, when the fitness of such frog is computed, the palette represented by this frog is also recomputed. As a result, the fitness of the new palette is better than the fitness of the previous one.

Computational experiments show that better results are obtained if the improvement procedure is applied to all of the frogs in the population, but this also increases the execution time. In this case the improvement in the quality of the quantized image may be less useful due to the time requirements needed for its execution. For this reason, the improvement is applied to only one frog.

4.2.3. Generation of the quantized image

When the iterative process ends, the quantized palette associated with the global best frog defines the solution to the problem: $x_g = \{c_1^g, \ldots, c_q^g\}$, with $c_k^g = (R_k^g, G_k^g, B_k^g)$. This palette is used to define the quantized image. To generate such image, the entire original image must be considered (even though the algorithm has used sampled data), since it is necessary to decide which color of the quantized palette will represent each point of the original image. For this purpose, each pixel p_i of the original image is compared to each color of the quantized palette, and the closest color of the palette is used to represents p_i in the quantized image.

If the SFLA-CQ algorithm uses sampled data (if $step > 1$), the colors of x_g are computed based on a subset of pixels of the original image, which is small when *step* is large. Therefore, to better adjust the palette to the entire set of pixels of the original image, the palette is updated before generating the quantized image. This update consists of replacing each color c_k^g with the average color of all the pixels of the original image associated with c_k^g in the operation described in the previous paragraph. This last operation improves the result, especially for large values of *step* which uses a sampled image with very few points.

Table 1
Test images: name, number of rows and columns (pixels) and number of different colors.

Image	Dimensions	Colors	Image	Dimensions	Colors
Plane	512 × 512	77041	Landscape	600 × 450	93255
Lake	512 × 512	168459	Headbands	600 × 450	93303
Girl	512 × 512	79228	Dessert	600 × 450	103792
Mandrill	512 × 512	230427	Snowman	450 × 600	92413
Peppers	512 × 512	185525	Cathedrals beach	600 × 450	66305
Lenna	512 × 512	148279	Beach	600 × 450	124335

5. Results and discussion

The SFLA-CQ algorithm was coded in C language and was applied to the 12 images shown in Fig. 1. The first 6 are images commonly used to test color quantification methods (they are available at Weber, 2018) and the remaining ones are new test images defined by the author of this article, which are available at Pérez-Delgado (2018). The features of the test images are summarized in Table 1.

Tests were performed on a PC running under Linux operating system, with 8 GBytes of RAM and AMD Ryzen 7 1800X Turbo processor (4.0 GHz). The value selected for the parameters of the algorithm was: $R = 8$, $m = 2$, $D_{max} = 5$, $J_{max} = 4$ and $T_{max} = 10$. Although the general tests were performed with the previous values, this section also analyzes the results obtained with different values of each parameter.

To evaluate the effect of data sampling on the error and the execution time, several values were considered for the *step* parameter: $step = \{1000, 500, 200, 100, 10, 5, 2, 1\}$. These values of *step* generate sampled images with the following number of pixels: {263, 525, 1311, 2622, 26215, 52429, 131072, 262144} for images (a) to (f) of Fig. 1; {270, 540, 1350, 2700, 27000, 54000, 135000, 270000} for the remaining images.

Four palette sizes were considered: $q = \{32, 64, 128, 256\}$. Twenty independent tests were performed for each image, palette size and *step*

Table 2
Results of 20 independent tests with $m = 2$, $R = 8$, $J_{max} = 4$, $T_{max} = 10$, $D_{max} = 5$ and several values of the *step* parameter. (MSE_m: minimum MSE; MSE_a: average MSE; *dev*: standard deviation; T: execution time (milleseconds)).

		32 colors				64 colors				128 colors				256 colors			
	step	MSE_m	MSE_a	*dev*	T	MSE_m	MSE_a	*dev*	T	MSE_m	MSE_a	*dev*	T	MSE_m	MSE_a	*dev*	T
Plane	1000	93.64	129.28	18.6	60	70.64	84.46	10.3	123	52.34	55.00	1.7	260	41.70	44.94	1.1	487
	500	106.55	125.58	12.6	73	59.05	74.25	8.8	139	40.34	46.42	3.5	287	33.02	35.11	1.2	575
	200	81.55	114.09	14.2	97	56.26	64.18	6.1	182	36.44	42.28	3.1	379	28.29	29.68	0.9	739
	100	80.69	108.13	13.7	139	53.73	62.82	7.0	261	36.63	39.93	2.1	514	24.94	27.32	1.4	1026
	10	90.95	111.98	11.4	854	47.91	57.47	5.3	1649	32.28	36.49	1.9	3146	21.83	23.36	0.8	6507
	5	102.13	123.11	14.0	1672	52.96	58.61	4.4	3305	31.38	34.97	2.2	6180	21.70	23.34	1.0	12669
	2	81.55	105.41	12.2	3885	52.37	60.55	7.1	7769	33.06	35.88	1.5	15081	21.47	23.21	1.0	30198
	1	81.09	112.42	14.8	7755	46.89	59.76	8.1	15400	33.41	37.55	2.4	29939	21.30	23.39	1.2	60463
Lake	1000	239.88	251.46	5.8	62	172.04	183.16	6.5	126	133.58	138.42	2.7	254	109.95	114.16	2.0	506
	500	220.38	225.90	4.4	72	154.72	160.77	3.2	146	110.27	117.66	4.0	281	81.10	85.36	1.9	584
	200	214.41	219.64	2.8	98	146.13	151.91	3.3	186	99.06	104.90	2.6	366	71.31	75.80	2.3	745
	100	208.97	214.14	3.8	140	137.70	143.80	3.6	256	94.86	97.44	1.7	511	64.52	66.64	1.2	998
	10	203.49	206.91	2.2	892	131.80	136.00	3.0	1580	87.62	89.37	1.1	3105	58.14	60.16	1.0	6257
	5	204.12	206.64	1.9	1716	132.68	136.19	3.0	3075	86.83	89.40	1.1	6165	57.32	59.16	0.8	11931
	2	203.63	205.92	1.6	4181	131.30	133.94	1.4	7430	86.84	88.51	1.1	14856	56.59	58.45	0.8	28967
	1	202.55	205.86	1.8	8147	131.51	133.61	1.4	14749	86.22	88.14	1.0	29328	56.64	57.95	0.8	57507
Girl	1000	125.39	136.27	6.4	62	94.81	100.25	4.8	126	69.84	75.11	4.3	244	62.25	63.69	0.8	491
	500	121.19	133.56	4.8	71	87.77	93.36	4.5	147	53.55	70.95	5.1	281	57.09	58.94	2.3	561
	200	86.27	102.80	12.0	96	55.71	58.38	1.8	198	35.50	39.29	2.8	375	23.25	27.52	2.4	773
	100	85.19	91.27	6.0	140	53.11	55.29	1.5	273	33.17	36.81	1.4	539	23.26	25.81	1.8	1057
	10	82.98	89.16	7.4	881	50.32	52.51	1.0	1697	30.57	33.72	1.1	3316	20.36	21.79	0.8	6468
	5	82.01	87.62	3.4	1697	50.94	52.98	1.0	3252	31.23	33.30	0.9	6222	19.98	21.76	1.0	12201
	2	84.13	87.30	2.4	4171	49.48	52.07	1.3	7820	31.65	33.26	0.8	15327	19.80	21.70	1.0	29450
	1	83.06	87.10	4.2	8088	50.98	52.41	0.9	15490	31.81	33.11	0.8	29753	19.94	21.33	0.8	58985
Mandrill	1000	416.42	437.95	14.4	63	268.61	293.55	8.3	127	196.42	202.00	3.8	259	158.07	161.63	2.0	512
	500	394.12	407.30	7.2	73	261.29	268.25	4.8	144	175.79	184.33	4.4	330	127.66	130.94	1.8	609
	200	390.86	397.43	5.6	102	250.75	255.48	3.1	191	168.56	172.18	2.5	402	114.05	117.79	1.7	733
	100	380.61	387.04	3.8	144	244.97	249.38	2.7	273	161.88	164.30	1.3	562	108.38	110.17	1.2	1011
	10	375.29	382.09	3.7	892	239.15	240.77	1.2	1721	153.60	154.77	0.7	3500	99.68	100.58	0.6	6586
	5	375.52	379.65	3.0	1688	237.90	239.69	1.6	3310	152.83	154.03	0.8	6722	98.70	99.41	0.5	12806
	2	375.85	381.48	4.2	4102	236.18	239.03	1.8	7966	152.37	153.69	1.2	16734	97.82	98.55	0.5	31460
	1	373.72	379.10	3.7	8089	236.31	238.87	1.6	15768	151.85	153.01	0.8	32843	97.45	98.05	0.4	65011
Peppers	1000	282.67	300.94	8.1	60	186.37	206.31	8.3	125	142.67	149.69	5.3	259	117.66	121.83	3.9	490
	500	253.84	259.03	4.3	70	161.87	168.56	4.8	142	114.95	120.74	4.0	284	83.75	87.92	1.9	560
	200	238.76	256.71	12.8	96	143.20	149.44	5.8	181	93.70	98.42	2.3	359	64.48	67.65	1.4	727
	100	232.24	242.20	8.1	141	141.96	148.67	4.6	259	90.31	93.24	2.5	506	61.75	63.01	0.8	1004
	10	230.45	234.86	2.5	884	137.09	143.66	3.2	1635	85.61	88.45	3.3	3221	55.78	57.24	0.7	6323
	5	231.35	236.74	5.2	1678	134.75	141.25	4.3	3109	84.72	89.21	4.2	6069	55.51	56.56	0.6	12313
	2	230.63	235.59	3.6	4026	135.27	141.83	3.7	7518	84.48	87.10	2.9	15149	55.02	55.89	0.6	30413
	1	230.14	233.98	2.3	7962	134.47	140.55	3.6	14767	83.84	87.20	3.4	29386	54.71	55.60	0.5	58755
Lenna	1000	127.70	140.25	7.22	63	84.59	92.45	4.77	126	62.43	65.17	1.71	265	50.51	51.52	0.88	514
	500	130.21	132.79	2.12	72	82.81	86.32	2.25	142	54.20	56.44	1.16	296	41.16	41.79	0.62	610
	200	124.07	128.08	2.48	98	76.63	79.41	1.57	192	52.11	53.12	0.68	409	35.57	36.82	0.74	829
	100	120.93	123.40	2.35	144	75.46	77.13	1.13	280	50.38	51.11	0.69	577	34.07	34.86	0.53	1142
	10	118.65	121.56	2.21	955	73.16	74.79	1.01	1865	47.06	48.23	0.76	3714	31.62	31.96	0.31	7626
	5	119.13	121.27	1.63	1908	73.04	73.87	0.53	3548	46.80	47.74	0.75	7193	31.08	31.56	0.25	14665
	2	119.34	120.50	0.74	4579	72.76	74.32	1.52	8589	46.78	47.32	0.40	17888	30.65	31.19	0.39	35801
	1	118.37	120.37	1.21	8654	72.41	73.76	0.98	17022	46.88	47.64	0.57	35134	30.69	31.06	0.24	70477

value; the results of these tests appear in Tables 2 and 3. To evaluate the quality of the solution, the MSE was considered (Eq. (4)). The tables show the minimum error, the average error and the standard deviation, as well as they show the average execution time.

It can be observed that the size of the sampled image influences the execution time and the quality of the final image. When using sampled images with few pixels, worse quantized images are obtained, but the result is obtained faster. However, even for sampled images that include very few pixels, the algorithm is capable of generating acceptable quantized images. This can be seen in Fig. 2, which shows the Lenna image with 32 colors obtained with different values of the *step* parameter.

In general, it is observed that the average MSE obtained for tests that consider $step \leq 10$ is very similar. Therefore, SFLA-CQ can generate good quantized images with less time consumption if the tests consider $step = 10$.

The worst MSE results are obtained for the Mandrill image, which is the image that includes more different colors. The next images with the worst MSE are Peppers and Lake, which also include many different colors. On the other hand, the image with fewer different colors (Cathedrals) generates the best MSE results.

The best quantized image obtained by SFLA-CQ with $step = 1$ for each image and palette size is included as supplementary material.

5.1. The effect of the algorithm parameters

This section includes results of several additional tests that allow the effect of the algorithm parameters on the solution to be analyzed. To reduce the length of this section, the comparison is applied to a single image of the test set (the Lenna image) and only considers the smallest and largest palettes. When a parameter is analyzed, the others take the values used to perform the tests reported in Tables 2 and 3. For each parameter analyzed, a figure shows the execution time and the average MSE obtained for the Lenna image quantized to 32 and 256 colors. Since the range of time variation increases as parameter *step* decreases, two sub-figures with different scales are used to represent the execution time, each of which shows the result for half of the *step* values. In general, line charts are used to show the execution time, since

Table 3
Results of 20 independent tests with $m = 2$, $R = 8$, $J_{max} = 4$, $T_{max} = 10$, $D_{max} = 5$ and several values of the *step* parameter. (MSE_m: minimum MSE; MSE_a: average MSE; dev: standard deviation; T: execution time (milliseconds)).

		32 colors				64 colors				128 colors				256 colors			
	step	MSE_m	MSE_a	dev	T	MSE_m	MSE_a	dev	T	MSE_m	MSE_a	dev	T	MSE_m	MSE_a	dev	T
Landsca.	1000	141.81	158.30	10.3	66	104.02	112.31	4.6	132	85.38	91.13	3.1	266	75.85	79.52	2.1	525
	500	103.10	115.00	6.8	75	65.27	70.69	3.2	150	42.13	45.21	1.9	300	31.96	33.30	0.7	630
	200	141.16	154.22	7.1	106	98.43	107.00	4.6	216	70.43	77.31	5.6	440	55.70	59.45	1.8	960
	100	104.82	107.35	2.0	145	59.64	63.28	1.8	295	39.81	42.17	1.7	573	27.22	28.82	1.2	1212
	10	98.94	102.31	2.8	949	54.54	57.72	1.5	1915	33.79	35.48	0.9	3759	22.03	22.87	0.5	7244
	5	98.89	102.78	3.2	1813	55.80	57.52	1.4	3590	33.31	35.27	1.1	7317	21.10	22.58	0.5	14228
	2	98.49	102.33	2.6	4381	55.53	57.55	1.1	8755	32.45	33.99	0.8	17527	20.91	22.16	0.7	35041
	1	98.67	102.73	2.5	8626	55.23	56.69	1.3	17644	33.55	34.73	1.0	34613	20.80	22.05	0.5	69494
Headban.	1000	162.88	175.58	10.0	61	117.44	124.24	4.8	128	95.33	99.53	2.5	270	77.34	83.25	3.6	518
	500	138.58	155.97	12.4	71	92.78	104.45	5.8	144	64.59	72.81	5.2	298	47.50	51.80	1.8	611
	200	148.52	162.08	9.2	93	107.02	116.01	4.9	197	79.44	90.13	5.0	422	61.81	65.63	2.2	876
	100	128.65	148.30	9.6	133	87.76	93.02	4.3	259	58.44	64.11	3.3	530	41.16	45.74	2.7	1150
	10	124.60	132.01	5.1	812	74.13	82.56	6.0	1663	48.36	51.35	2.0	3323	31.67	33.02	1.2	6914
	5	121.32	129.06	5.0	1558	72.79	79.38	4.0	3171	47.28	51.08	2.0	6428	30.88	32.54	1.1	13190
	2	118.49	126.88	6.5	3801	72.30	79.79	2.8	7509	45.42	49.93	2.3	15347	29.79	32.12	1.1	32177
	1	115.02	129.77	8.9	7418	73.44	79.12	3.2	14991	46.69	49.82	1.7	31290	30.04	31.44	1.2	64268
Dessert	1000	164.43	183.22	12.1	64	110.16	121.32	7.0	139	82.33	86.34	2.6	270	67.74	70.85	2.0	518
	500	139.86	150.18	8.0	71	89.88	97.44	3.7	155	58.72	65.49	2.8	312	44.85	47.78	1.6	625
	200	155.19	169.59	11.9	103	89.53	100.01	5.6	208	57.10	63.34	4.1	430	41.63	44.02	1.9	907
	100	130.68	142.17	5.3	137	75.83	83.73	3.2	311	50.37	54.08	1.9	572	35.18	37.42	1.6	1219
	10	121.86	128.21	4.2	900	68.89	71.81	1.8	1792	42.05	44.20	0.9	3471	26.06	27.35	0.6	6838
	5	121.00	126.82	2.5	1709	66.67	70.71	2.3	3391	41.65	43.12	1.3	6673	24.99	26.51	0.7	13309
	2	123.33	126.84	3.2	4100	66.91	70.75	2.3	8528	40.92	43.21	1.3	17194	25.41	26.09	0.5	32395
	1	121.21	127.15	5.3	8063	68.36	71.60	2.2	16641	41.03	42.84	0.9	31867	25.45	26.46	0.4	64671
Snowman	1000	136.07	152.86	6.1	64	89.92	99.00	5.5	128	65.91	68.53	2.4	260	54.28	56.01	1.0	540
	500	139.15	149.95	7.4	71	86.71	95.10	4.7	145	55.67	64.21	4.3	295	44.67	47.01	1.6	623
	200	134.79	146.57	6.6	98	82.00	88.94	4.2	196	55.00	59.48	3.4	400	37.36	41.49	2.3	826
	100	130.02	145.57	9.9	140	75.62	85.09	5.2	275	48.86	53.93	2.1	554	35.92	38.49	1.6	1178
	10	118.15	126.04	8.7	905	63.31	69.29	3.2	1765	38.56	41.10	1.4	3286	24.66	25.91	0.7	6633
	5	115.08	129.12	9.3	1711	63.32	67.98	2.4	3217	37.85	39.69	1.4	6376	23.52	25.06	0.6	12239
	2	116.18	124.21	8.5	4197	64.23	68.60	2.8	7824	38.06	39.80	1.3	15172	23.68	24.81	0.6	30480
	1	115.58	123.03	4.9	8165	63.01	67.81	2.9	15644	38.05	39.54	1.1	30058	23.17	24.34	0.6	61678
Cathedr.	1000	118.79	130.99	7.3	63	87.11	96.19	4.8	131	71.89	76.88	4.0	265	62.19	65.29	3.1	523
	500	68.23	77.08	6.3	71	42.06	46.39	2.8	144	27.65	30.41	1.7	295	20.25	21.37	0.5	620
	200	97.85	113.37	12.1	107	66.19	75.19	7.5	216	49.43	54.60	2.4	436	36.65	43.22	2.9	902
	100	63.33	67.54	4.1	141	36.62	39.67	2.2	283	23.46	25.54	1.5	573	15.74	17.17	1.0	1189
	10	60.97	63.89	2.3	906	32.80	35.26	1.1	1778	20.64	21.57	0.6	3597	13.46	14.21	0.3	7498
	5	59.95	63.44	3.2	1722	33.52	36.07	1.8	3447	20.05	21.16	0.6	6990	13.15	13.86	0.3	14389
	2	60.06	64.18	3.4	4207	32.93	35.92	1.6	8210	19.70	21.22	0.7	16977	13.23	13.73	0.3	35035
	1	58.86	63.92	3.5	8206	33.32	35.43	1.0	16362	20.18	21.11	0.5	33128	13.00	13.52	0.3	70015
Beach	1000	155.80	169.41	9.0	62	95.24	101.45	3.8	132	66.26	69.52	2.0	275	53.45	55.99	1.0	530
	500	150.97	156.29	4.9	73	86.61	91.67	2.3	147	55.29	57.80	1.6	310	39.79	41.05	0.8	657
	200	148.61	156.90	5.2	100	88.27	92.07	2.6	207	54.43	58.73	2.0	425	37.60	38.74	0.8	935
	100	141.48	149.27	3.8	149	78.73	84.05	2.7	289	49.36	52.50	1.9	595	33.46	35.02	0.8	1221
	10	137.07	142.77	4.0	939	75.18	77.76	2.0	1851	45.57	47.11	0.8	3836	28.91	30.06	0.6	7456
	5	136.05	142.55	4.2	1805	74.36	76.92	2.2	3631	44.80	46.78	1.2	7698	28.33	29.34	0.4	14402
	2	135.51	140.50	4.2	4394	74.97	77.78	2.3	8706	45.36	46.61	0.8	20240	27.85	29.42	0.7	35390
	1	136.67	143.34	6.2	8480	73.29	77.07	3.0	17244	45.42	46.72	0.7	41639	28.01	28.93	0.6	71549

they allow evolution to be easily seen. However, bar charts are used to make the graph clearer when the lines are very close.

The *step* parameter is not analyzed in this section because Tables 2 and 3 already include the results for several values of this parameter. Regarding the other parameters, the number of iterations of the algorithm (T_{max}) and the number of improvement iterations applied to each memeplex (J_{max}) are those that have more influence on SFLA-CQ results. If the value of both parameters increases, better images can be obtained, but the execution time also increases. Therefore, it is necessary to choose between quality and speed when assigning value to these parameters.

The following subsections analyze the effect of each parameter.

5.1.1. The effect of T_{max}

The number of iterations of the algorithm greatly influences the execution time. Fig. 3 shows the results for several values of this parameter: $T_{max} = \{1, 5, 10, 15, 20\}$. It is clearly observed that, as the number of iterations increases, the error decreases and the execution time increases. The largest reduction in the MSE error is obtained during the initial iterations of the algorithm, especially for the smallest *step* values. In addition, the increase in the execution time is approximately linear with the number of iterations.

5.1.2. The effect of J_{max}

The number of improvement iterations applied to each memeplex also influences the execution time of the algorithm. When the value of this parameter is large, the local search of each memeplex is intensified and this increases the execution time.

Fig. 4 compares the results of $J_{max} = \{4, 8, 16\}$. This figure shows that as J_{max} increases the execution time also increases, especially for the smallest *step* values. Nevertheless, the MSE value does not always improve with the increase of J_{max}.

5.1.3. The effect of R

Fig. 5 shows the results for various population sizes: $R = \{8, 16, 32, 64\}$. It can be observed that the number of frogs in the population has less influence on the execution time than the previous parameters.

(a) $step = 1000$ (b) $step = 500$

(c) $step = 200$ (d) $step = 10$

(e) $step = 2$ (f) $step = 1$

Fig. 2. Lenna image with 32 colors generated by SFLA-CQ with several values of the *step* parameter.

The increase in time is not linear with the number of frogs, since the iterative operations of this algorithm only modify the worst frog in each memeplex and the best frog in the population, instead of all the frogs.

On the other hand, a population with more frogs does not always guarantee better quantized images. When the population includes more frogs, more elements of the search space (palettes) are considered to select the best one. However, the operations of the algorithm mainly work on the worst frog in each memeplex, independently of the number of frogs in the memeplex.

5.1.4. The effect of m

Fig. 6 shows the results of several memeplexes, $m = \{2, 4, 6, 8\}$. Since the results reported in Tables 2 and 3 consider 4 frogs per memeplex, the same value is considered in this case.

It is clear that the value of this parameter influences the execution time: more memeplexes require more operations to optimize them. When m increases, better images are obtained in some cases, but this is not true for all cases.

The number of memeplexes influences the execution time more than the number of frogs, since each additional memeplex requires J_{max} iterations of improvement of the said memeplex.

It should be noted that the number of frogs in the population is the same for the results in Figs. 5 and 6. Both figures show the results obtained for populations with 8, 16, 32 and 64 frogs, but with different number of frogs per memeplex. When both figures are compared, it is clear that the execution time varies more when m is modified. This is because the algorithm tries to improve each memeplex, so the execution time is proportional to the number of memeplexes. On the other hand,

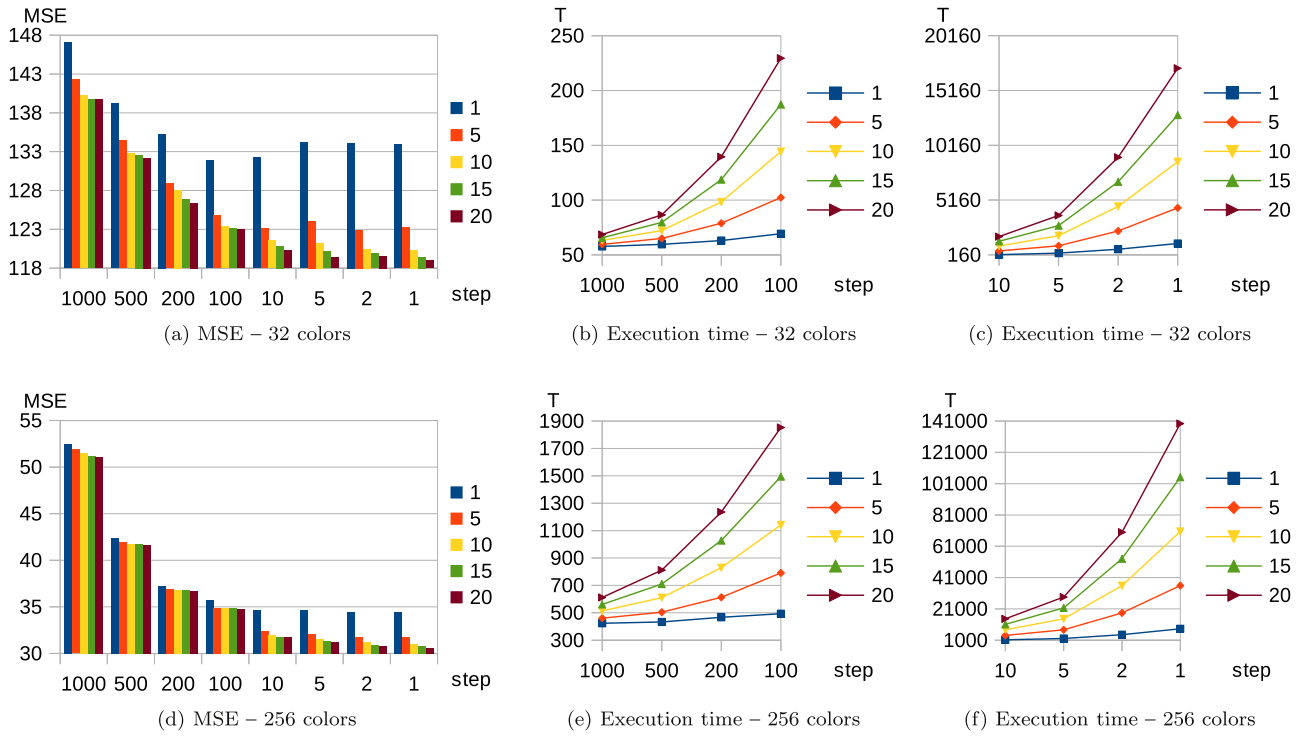


Fig. 3. MSE and execution time (milliseconds) for Lenna image quantized to 32 and 256 colors with $T_{max} = \{1, 5, 10, 15, 20\}$.

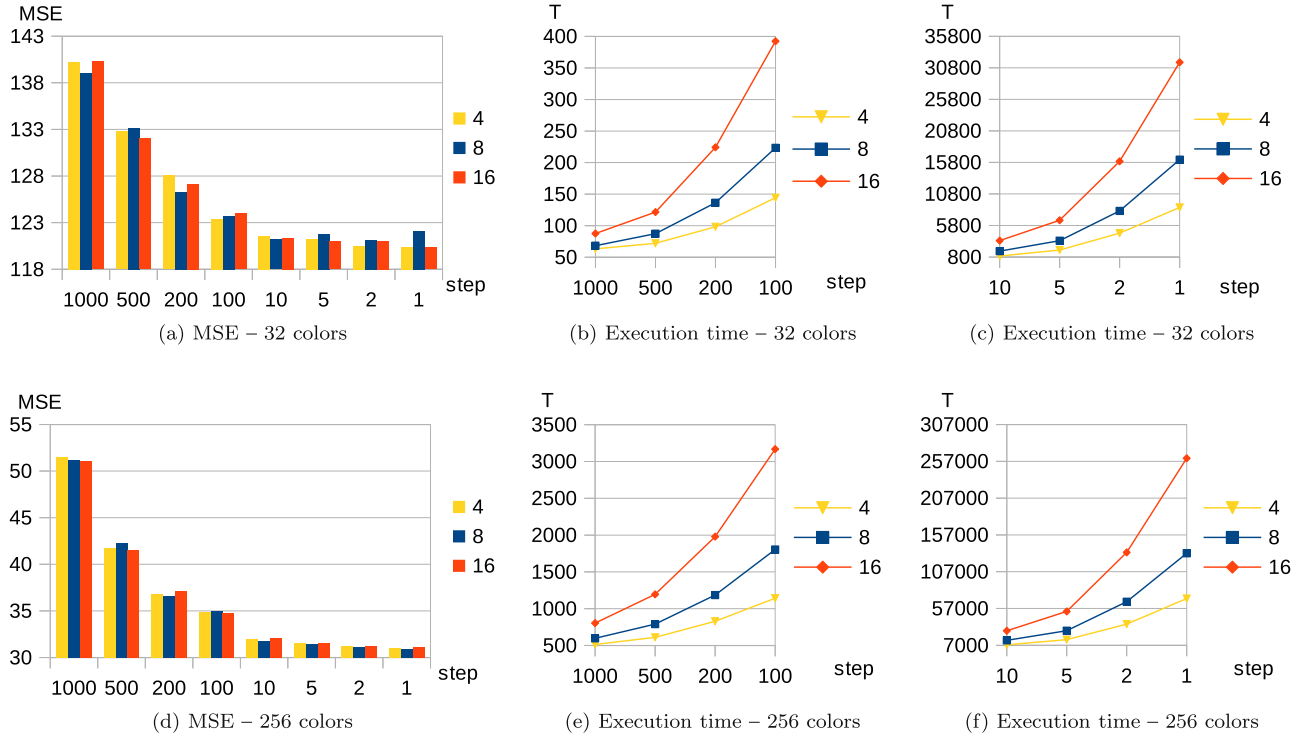


Fig. 4. MSE and execution time (milliseconds) for Lenna image quantized to 32 and 256 colors with $J_{max} = \{4, 8, 16\}$.

this improvement considers the best and the worst frogs, not all the frogs of the memeplex; for this reason, the amount of frogs included in a memeplex does not influence the execution time as much.

5.1.5. The effect of D_{max}

The maximum change allowed in a frog's position was defined taking into account the range of variation of the variables in the color quantization problem: $[0, 255]$. Fig. 7 shows results for several values: $D_{max} = \{5, 10, 25, 50, 255\}$.

This parameter has little influence on the error of the final image and the range of variation of the results represented in the figures is narrow, especially for 256 colors. For 32 colors the maximum variation is 3.94 (obtained for $step = 500$) and the minimum is 0.70 (obtained for $step = 10$) and for 256 colors the maximum range is

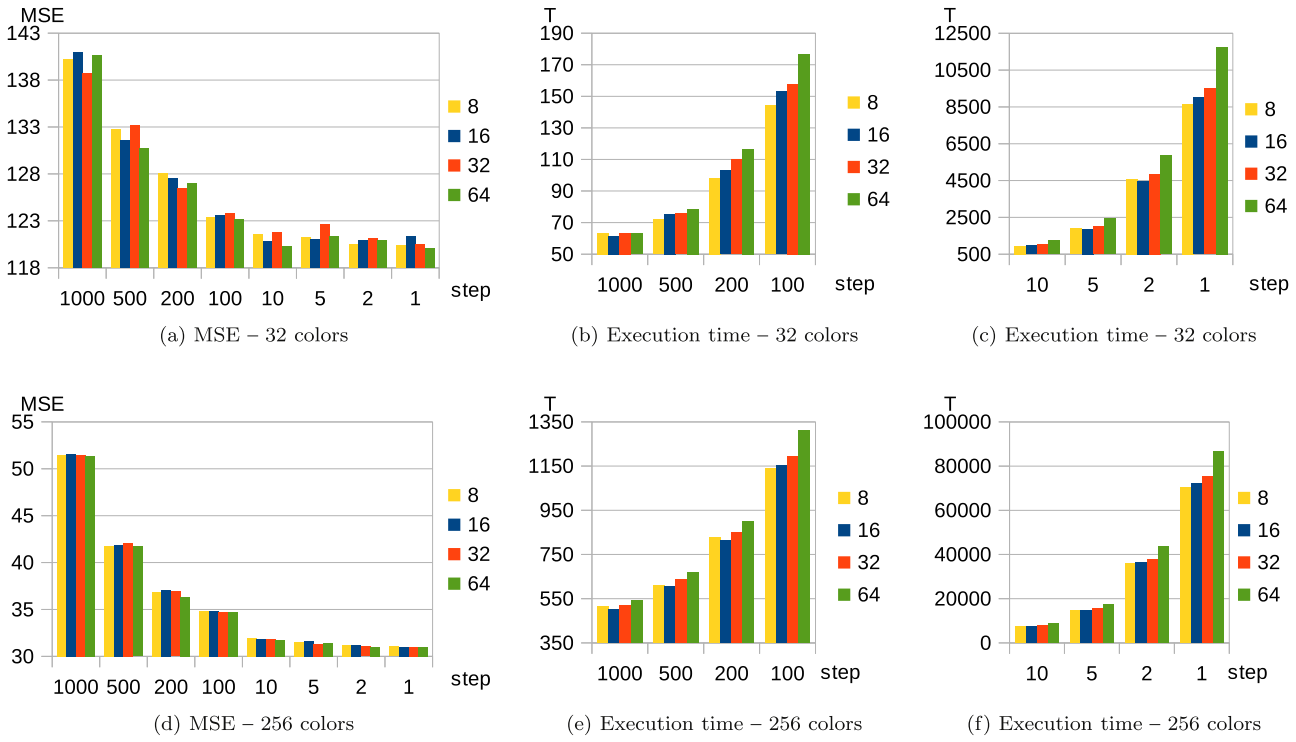


Fig. 5. MSE and execution time (milliseconds) for Lenna image quantized to 32 and 256 colors with $R = \{8, 16, 32, 64\}$.

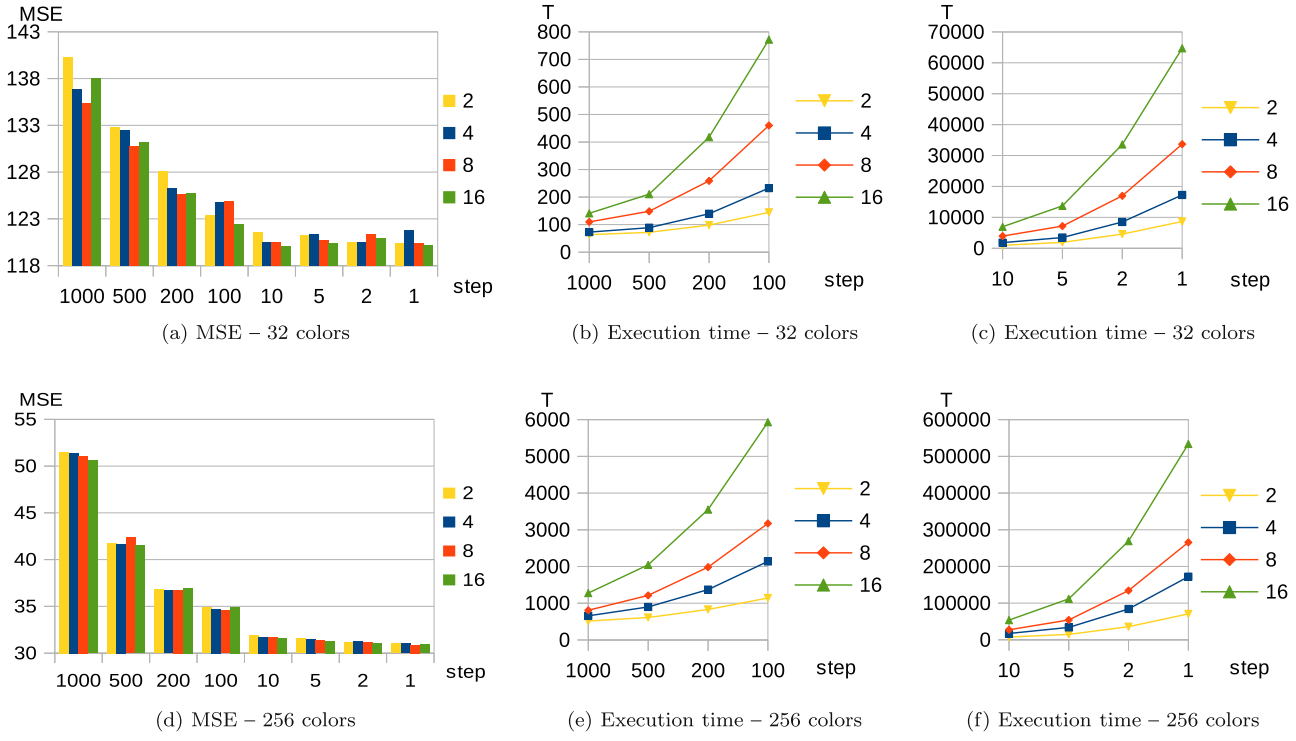
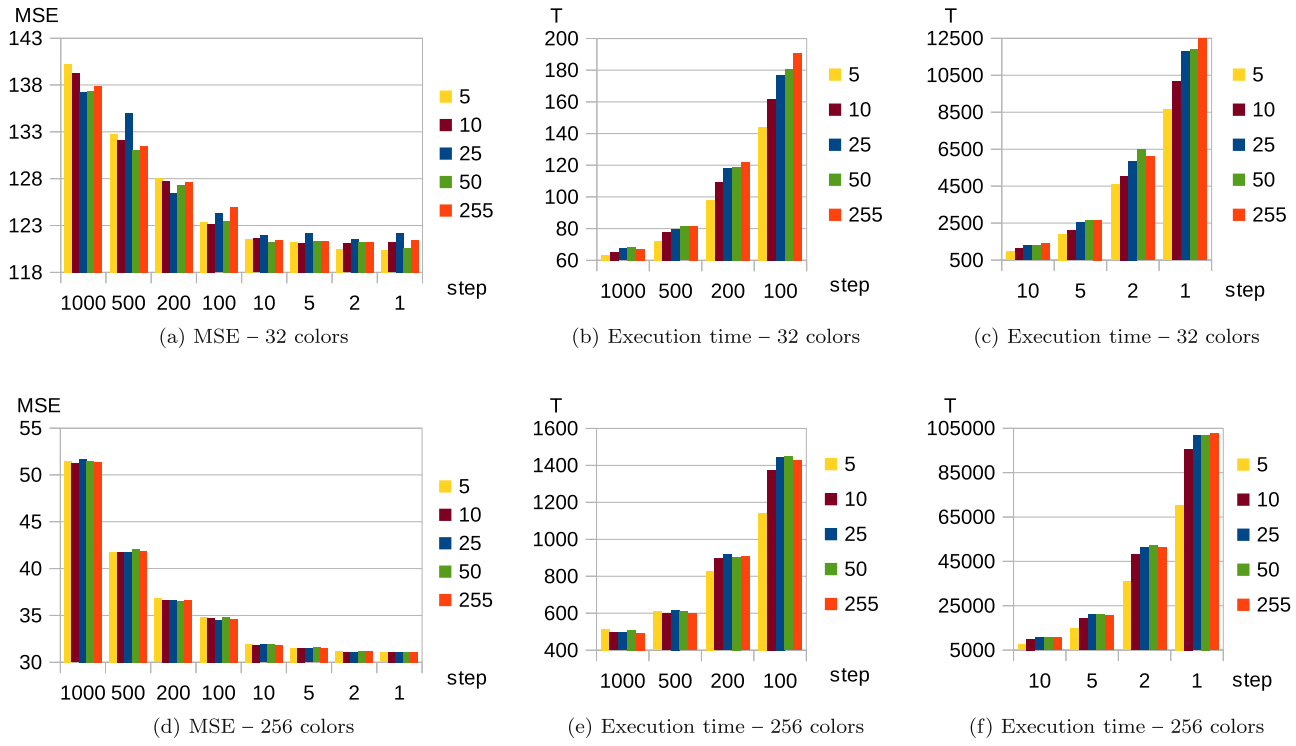


Fig. 6. MSE and execution time (milliseconds) for Lenna image quantized to 32 and 256 colors with $m = \{2, 4, 8, 16\}$.

0.52 and the minimum is 0.08, obtained for $step = 1000$ and $step = 1$, respectively.

In general, the execution time increases with the value of D_{max} and the shortest execution time is obtained for $D_{max} = 5$. Large values of D_{max} allow a wide range of variation in the colors of the palette when Eq. (2) is applied. In many cases this obtains a candidate palette that is worse than the current palette of the worst frog in the memeplex. In such cases, after generating the first candidate palette (line 9 of Algorithm 1), the second candidate palette must be generated (line 13 of Algorithm 1), and in many cases a random candidate palette must finally be selected (line 17 of Algorithm 1). All of these operations increase the execution time, although the error of the final image does not always improve. In any case, it should be noted that this parameter has less influence on the execution time than T_{max}, J_{max} and m.



(a) MSE – 32 colors (b) Execution time – 32 colors (c) Execution time – 32 colors

(d) MSE – 256 colors (e) Execution time – 256 colors (f) Execution time – 256 colors

Fig. 7. MSE and execution time (milliseconds) for Lenna image quantized to 32 and 256 colors with $D_{max} = \{5, 10, 25, 50, 255\}$.

5.2. The new operations of the algorithm

5.2.1. The improvement of a frog at the end of each iteration of the algorithm

This section analyzes the effect on the final solution of the improvement applied to the best frog in the population.

To analyze this improvement, several tests were performed that applied the improvement to other frogs. Three possibilities were analyzed:

- the improvement was applied to a random frog of the population.
- no frog was improved.
- all frogs in the population were improved.

Fig. 8 compares the results obtained in previous cases using the results of improving the best frog in the population.

It can be observed that the improvement of all of the frogs is the most time-consuming option, while not improving any frog is the least time-consuming option. The execution time is very similar for both cases that improve only one frog.

When the average MSE is analyzed, the worst results correspond to the case that does not improve any of the frogs (especially for $step \leq 10$), followed by the case in which a random frog of each memeplex is improved. Furthermore, the best results are obtained when all of the frogs are improved. Since this last option is the most time-consuming, a good alternative would be the case that improves the best frog, because it can generate good images with low computational cost.

5.2.2. The effect of recomputing the colors of a palette

As described in Section 4.1.1, when the fitness of a frog is calculated, the colors of the palette associated with that frog are recalculated. To analyze the effect of this operation, Fig. 9 compares the results obtained for the Lenna image in 3 cases:

- the colors of a palette are recalculated whenever the fitness is calculated (this is the case considered by SFLA-CQ).
- this operation is never performed.
- said operation is performed only when the palette includes randomly taken pixels; that is, it is executed when defining the initial position of each frog of the population (line 1 of Algorithm 1) and when choosing a random position for the worst frog in a memeplex (line 17 of Algorithm 1).

It is clearly observed that the MSE of the quantized image is worse if the palettes are not recalculated, especially for the case in which this operation does not apply to any palette. In addition, the effect of sampling the input image is less when the palette is not recalculated, since MSE results do not improve when the sampled image includes more pixels.

The case that recalculates the palettes every time the fitness is calculated is the one that consumes less time (it should be remembered that fitness is not recalculated in all iterations of the algorithm, but only when the position of the frog is modified). Although such calculation requires a number of operations proportional to the number of pixels of the input image (n'), the final execution time of the algorithm increases if this operation is not performed. This is due to the fact that when the palettes are not recalculated, the selection of a candidate position for the worst frog in a memeplex often involves assigning a random position to the frog. In such cases, the algorithm calculates two candidate solutions and finally selects a random position. Therefore, the three operations applied to modify the position of the worst frog increase the total execution time of the algorithm.

5.3. SFLA-CQ compared to other methods

To analyze the quality of the proposed solution, it was compared to several color quantization methods described in Section 2.2: the method proposed in Wu (1991) (WU), Variance-based method (VB), Octree (OC), Neuquant (NQ), Binary splitting (BS), K-means (KM), the Ant-tree for color quantization method proposed in Pérez-Delgado (2015) (ATCQ), Particle swarm optimization combined with K-means (PSO+KM) and Artificial bee colony combined with K-means (ABC+KM). The results of 20 independent tests are shown in Tables 4 and 5.

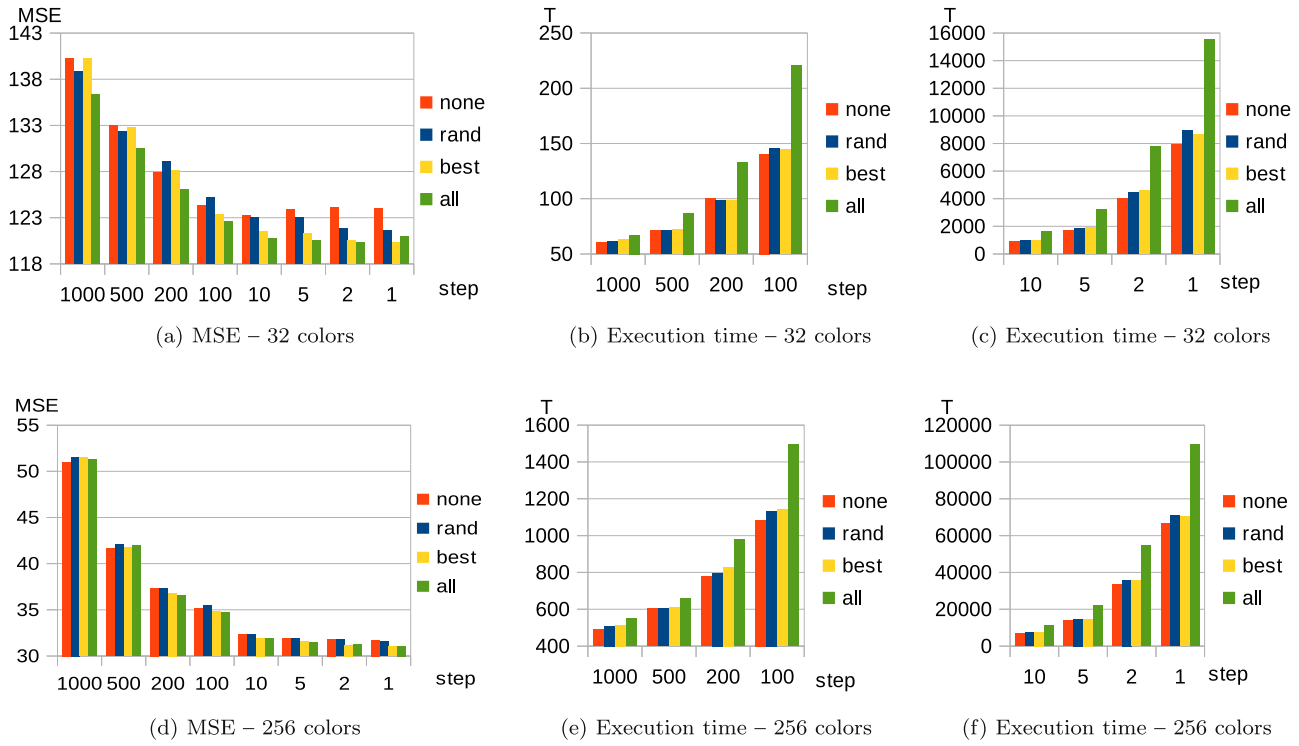


Fig. 8. MSE and execution time (milliseconds) for Lenna image quantized to 32 and 256 colors when different frogs are improved (*best*: the best frog in the population; *rand*: a random frog of the population; *all*: all the frogs in the population; *none*: no frog is improved).

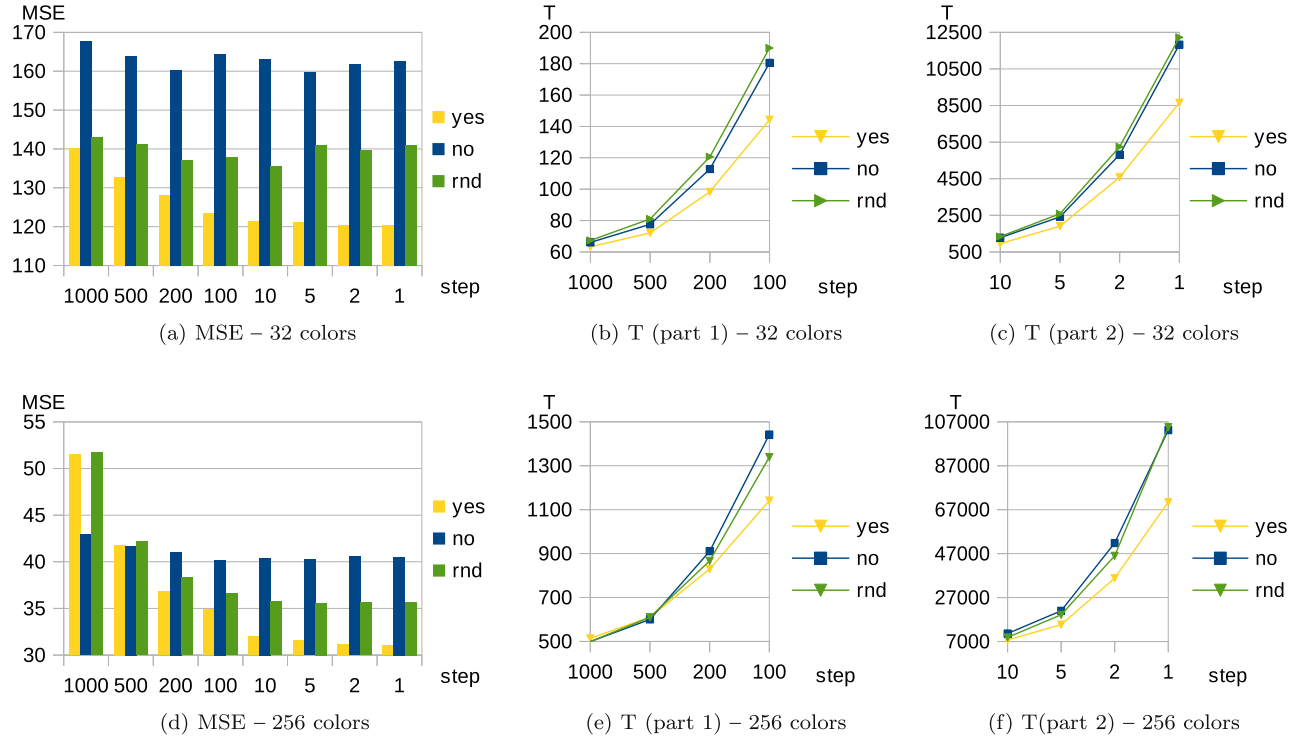


Fig. 9. MSE and execution time (milliseconds) for Lenna image quantized to 32 and 256 colors when several strategies are applied to recompute the colors of a palette when the fitness of such palette is computed (*yes*: colors recomputed for all the palettes; *no*: colors not recomputed for any palette; *rnd*: colors recomputed only when a random palette is selected).

For the 3 iterative methods compared (KM, PSO+KM and ABC+KM), the same number of iterations applied to SFLA-CQ were executed (10 iterations). The sampling factor of Neuquant was set to 1 in order to obtain the best solution that this method is able to generate. The parameters of PSO+KM were set as follows: the cognitive and the social parameters were set to 1.49; the inertia was set to 0.72; the range for the velocity of the particles was $[-5, 5]$ and the probability of application of K-means was set to 0.1 (these values were proposed by Omran et al., 2005). In addition, the number of particles was set to 8 (because SFLA-CQ considers 8 frogs) and only 5 iterations of

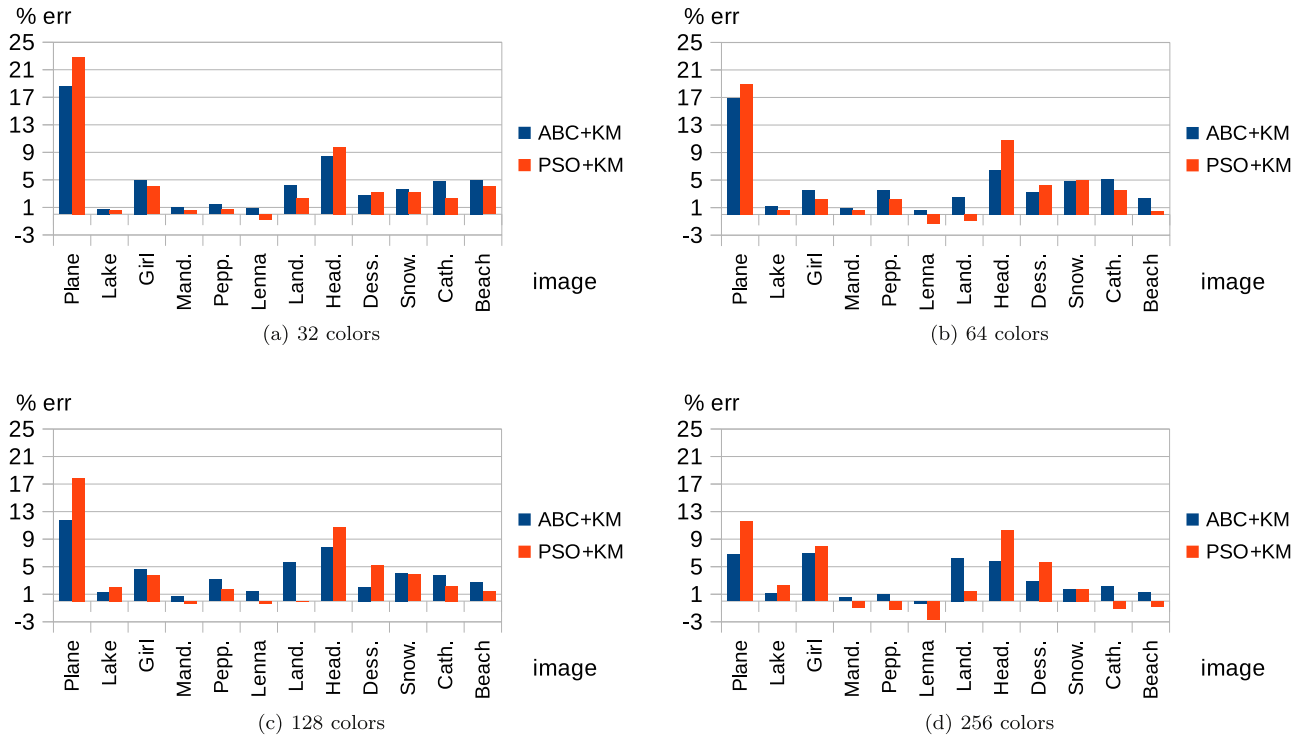


Fig. 10. Percentage of error reduction obtained by PSO+KM and ABC+KM with respect to SFLA-CQ with $step = 1$.

K-means were performed (to reduce the execution time of the method). ABC+KM was executed with 8 food sources (to equal the number of frogs used by SFLA-CQ) and the same number of iterations of K-means and the probability of application of this method that was used when applying PSO+KM. The values given to the α parameter of ATCQ were established taking into account the palette size, as proposed in Pérez-Delgado (2015): $\alpha \in [0.20, 0.30]$ for 32 colors; $\alpha \in [0.25, 0.35]$ for 64 colors; $\alpha \in [0.30, 0.40]$ for 128 colors and $\alpha \in [0.40, 0.50]$ for 256 colors.

A general comparison of the methods showed that SFLA-CQ with $step \leq 10$ can be considered as good cases to be compared with other methods. With these $step$ values, SFLA-CQ obtains better quantized images than the other methods in many cases. In addition, the results corresponding to palettes with fewer colors are also improved by SFLA-CQ with large $step$ values. In terms of execution time, splitting methods are generally faster than clustering-based methods, and this also happens when such methods are compared to SFLA-CQ. On the other hand, the proposed method can obtain better images with less execution time than the other clustering-based methods analyzed; it is worth noting the great difference in the execution time with respect to PSO+KM and ABC+KM.

The following paragraphs compare the results of each method in more detail:

- SFLA-CQ with $step \leq 100$ improves the average MSE results of VB for all the images and palette sizes. In addition, for the Lenna, Mandrill, Lake and Peppers images this happens for all the values of $step$ analyzed. The tests corresponding to the largest $step$ values consume less time than VB for palettes with less than 256 colors. Therefore, SFLA-CQ can obtain better images than VB in less time for palettes with 128 colors as a maximum (only Girl image with 128 colors consumes a little more time); for 256 colors, the quantized images obtained are better, but they consume more time.
- OC is improved for almost all the cases reported in the tables. The results are not improved only for 9 out of 384 cases (12 images × 4 palettes × 8 $step$ values = 384), all of them corresponding to the three greatest values of $step$ and palettes with 128 or 256 colors. Although OC consumes less time than SFLA-CQ for palettes with 128 or 256 colors, SFLA-CQ can obtain better images of 32 and 64 colors than those obtained by OC and in less time.
- Wu's method is the least time-consuming method among all the methods compared. SFLA-CQ with $step \leq 10$ improves the MSE results of Wu's method for all the images and palettes except for the Plane image. Moreover, the error of some images has also been improved for all the palette sizes with $step$ values larger than 10 (Lenna and Mandrill for $step \leq 200$; Peppers, Cathedrals and Beach for $step \leq 100$), and even for some images of 32 and 64 colors the results are also improved with the two largest $step$ values.
- Except for the Plane image, the average MSE obtained by BS is worse than the results of SFLA-CQ with $step \leq 10$ for images with 32 and 64 colors. In addition, this is also true for 10 of the 12 images with 128 colors and for 8 images with 256 colors. On the other hand, SFLA-CQ also obtains for the smallest palettes better images than BS for $step$ values greater than 10 (with $step \geq 100$ it improves 11 images of 32 colors and 9 of 64 colors). Since the execution time of SFLA-CQ is shorter than that of BS for the two smallest palettes and the largest $step$ values ($step \geq 100$ for 32 colors and $step \geq 200$ for 64 colors), the first method can generate better images in less time for both palettes.
- SFLA-CQ generates images with better MSE (minimum and average) than KM for $step \leq 10$. On the other hand, SFLA-CQ with $step \geq 5$ consumes less time than KM for all the images and palette sizes. Therefore, the proposed method can generate better solutions than KM with less time consumption for all the images and palette sizes considered.
- SFLA-CQ obtains better average error than NQ for all the images and palette sizes with $step \leq 10$, except for Plane with 128 or 256 colors and Headbands with 256 colors. Since the results for most images with 32 colors are improved with $step \leq 500$, the smallest palette size can obtain better images using SFLA-CQ rather than NQ and in less time.

Table 4
Results of some color quantization methods: WU: Wu's method; VB: Variance-based method; OC: Octree; NQ: Neuquant; BS: Binary splitting; KM: K-means. (q: palette size; T: average execution time (milliseconds); MSE: mean squared error (for KM the minimum MSE (MSE_m), the average value (MSE_a) and the standard deviation (dev) are included)).

		WU		VB		OC		NQ		BS		KM			
	q	MSE	T	MSE	T	MSE	T	MSE	T	MSE	T	MSE_m	MSE_a	dev	T
Plane	32	85.45	13	123.56	218	342.23	141	123.70	127	83.12	152	99.04	141.13	23.72	3079
	64	51.33	13	80.73	256	225.98	143	57.57	183	46.22	204	56.72	91.63	25.98	5954
	128	32.60	14	52.60	309	133.59	151	29.71	273	28.06	253	33.05	48.12	15.13	11839
	256	21.66	14	36.85	372	52.31	158	21.09	474	18.42	355	22.70	25.33	1.74	23199
Lake	32	249.81	13	357.26	366	922.04	141	274.45	154	245.65	181	210.08	222.64	12.97	3058
	64	161.34	13	251.70	455	466.14	143	164.26	212	162.89	235	140.25	147.09	5.56	5931
	128	102.55	13	170.97	561	198.49	153	100.88	313	107.00	293	92.76	96.78	4.77	11718
	256	66.47	13	120.10	676	159.21	153	65.70	514	68.36	378	60.66	63.03	1.61	23236
Girl	32	107.55	13	232.43	142	477.64	148	123.89	138	111.28	192	89.25	117.71	21.30	3089
	64	63.03	13	129.43	160	163.08	160	66.32	201	63.07	239	54.13	67.93	13.87	5979
	128	36.84	13	82.22	193	89.86	159	38.30	307	38.46	293	33.75	36.14	1.95	11805
	256	23.59	14	54.42	208	50.79	159	23.86	496	23.65	389	21.59	23.06	0.78	23567
Mandrill	32	468.39	13	531.03	335	1094.11	144	456.13	149	441.30	220	384.72	394.47	5.18	3060
	64	288.33	13	346.58	409	576.19	150	272.25	216	286.93	254	244.46	247.86	2.66	5947
	128	186.33	13	248.02	496	357.13	152	168.22	320	183.51	312	156.81	159.15	2.05	12018
	256	118.65	13	181.94	600	195.82	153	109.34	513	117.85	429	100.68	102.22	1.11	23561
Peppers	32	279.27	12	451.10	312	777.14	151	283.69	152	311.66	193	240.75	259.36	14.74	3037
	64	165.37	13	304.21	380	495.51	144	166.37	222	173.75	245	140.87	149.32	5.18	5939
	128	102.31	13	212.68	440	308.76	152	95.69	323	106.85	292	87.78	92.87	3.64	11678
	256	66.08	13	147.18	528	156.24	153	63.08	530	68.72	374	57.48	60.18	3.24	23251
Lenna	32	158.61	12	203.07	242	482.03	143	152.13	134	138.44	198	122.00	127.80	5.88	3047
	64	99.16	13	135.86	292	212.92	148	85.60	203	82.44	265	76.32	81.33	4.49	5937
	128	61.79	13	94.68	345	140.53	155	53.73	310	53.04	301	48.69	50.23	1.03	11665
	256	39.53	13	69.41	401	74.12	160	34.88	508	35.28	379	32.55	33.29	0.51	23261
Landscape	32	131.31	14	164.35	230	576.99	145	139.03	133	129.18	221	104.55	111.03	5.02	3163
	64	72.20	13	113.61	272	185.19	152	70.13	201	66.61	262	58.28	61.59	3.85	6143
	128	42.75	13	84.46	328	149.46	148	37.17	306	36.48	325	33.78	37.32	1.56	12025
	256	25.75	14	61.90	367	53.17	173	23.75	520	22.18	419	22.58	24.25	1.02	24243
Headbands	32	142.63	13	184.82	295	430.86	148	188.99	141	149.52	212	135.96	151.13	17.66	3182
	64	87.52	13	121.28	374	192.74	151	99.49	210	84.86	249	85.05	93.86	7.05	6100
	128	53.42	14	79.90	471	128.22	149	52.12	326	51.30	303	53.34	58.39	3.46	12172
	256	33.83	14	53.29	557	53.17	168	30.77	522	31.28	384	34.62	35.61	0.78	23900
Dessert	32	160.65	13	191.72	256	426.95	146	176.26	127	152.16	203	132.09	146.45	15.46	3145
	64	90.42	13	118.69	307	203.65	153	90.70	183	85.29	242	72.94	77.31	4.23	6214
	128	52.66	13	82.38	363	118.32	150	51.09	285	48.50	298	45.49	46.95	1.84	11902
	256	32.71	14	56.92	426	67.32	157	30.21	485	29.42	387	28.15	29.53	0.89	25109
Snowman	32	161.12	13	216.35	284	559.57	151	202.14	137	163.63	206	124.17	153.03	20.54	3157
	64	89.53	13	118.13	366	334.53	147	90.05	191	87.54	255	71.76	76.05	3.70	6109
	128	49.69	13	70.03	477	134.31	154	44.23	294	46.97	315	42.74	47.05	3.22	12074
	256	29.85	14	43.68	518	84.45	155	27.51	514	27.48	415	26.21	28.05	1.38	24010
Cathedrals	32	81.90	13	105.84	261	316.46	146	93.72	120	72.42	188	65.46	74.03	7.81	3153
	64	45.28	13	61.50	324	109.02	153	48.67	179	38.57	212	36.29	39.91	3.11	6072
	128	27.31	14	40.15	386	69.79	154	24.33	275	22.43	256	21.98	23.83	1.76	11943
	256	18.10	14	26.16	408	45.83	158	15.28	476	13.29	337	14.72	15.23	0.43	24149
Beach	32	177.34	13	211.50	334	446.43	147	178.32	134	179.49	191	148.65	165.21	12.26	3140
	64	101.92	13	123.49	419	309.23	154	91.86	193	93.22	218	77.11	81.69	3.14	6143
	128	59.64	13	81.50	535	134.36	156	52.71	299	52.30	274	46.93	49.94	1.64	11945
	256	36.33	14	52.92	641	81.77	158	32.95	487	31.89	351	29.95	31.23	0.58	24724

- For $step \leq 100$, SFLA-CQ gets better quantized images than ATCQ for all the images and palette sizes except for Snowman and Headbands with 256 colors, which are improved when $step \leq 10$ is considered. Moreover, as the palette size decreases, SFLA-CQ improves the ATCQ results with larger $step$ values. For example, when 32 or 64 colors are considered, SFLA-CQ improves all the images with $step \leq 500$ and almost all with $step = 1000$ (only 3 cases are not improved for $step = 1000$: Cathedrals with 32 or 64 colors and Headbands with 64 colors). As far as the speed of the algorithm is concerned, SFLA-CQ consumes more time than ATCQ; the results are only comparable in some cases when considering the smallest palette and $step = 1000$.
- When the results of PSO+KM and ABC+KM are analyzed, it is revealed that both methods are highly time consuming: both consume much more time than SFLA-CQ even when the case with $step = 1$ is compared. In addition, the increase in the execution time does not mean that both algorithms generate much better images than those obtained by SFLA-CQ. This aspect can be seen in Fig. 10, which shows the percentage of error reduction obtained using both methods with respect to the solution of SFLA-CQ with $step = 1$. It can be observed that the best results of PSO+KM and ABC+KM are obtained for the Plane and Headbands images, obtaining an improvement of between 11.55% and 22.85% for the Plane image when PSO+KM is applied. Nevertheless, for the rest of the images the improvement is less than 5% in most cases and is close to 1% in some cases. In addition, SFLA-CQ improves the results of PSO+KM in 11 out of 48 cases (12 images × 4 palettes = 48), most of them corresponding to images with 128 or 256 colors. In summary, the results clearly show that PSO+KM and ABC+KM consume much more time than SFLA-CQ, but the quality of the images they generate does not improve much in most cases.

Table 5
Results of some color quantization methods: ATCQ: Ant-tree for color quantization method; PSO+KM: Particle swarm optimization combined with K-means; ABC+KM: Artificial bee colony combined with K-means. (q: palette size; T: average execution time (milliseconds); MSE_m: minimum MSE; MSE_a: average MSE; dev: standard deviation of MSE).

		ATCQ				PSO+KM				ABC+KM			
	q	MSE_m	MSE_a	dev	T	MSE_m	MSE_a	dev	T	MSE_m	MSE_a	dev	T
Plane	32	109.16	258.99	221.23	50	67.38	86.74	11.6	91483	71.63	91.53	11.3	146343
	64	58.30	89.27	34.31	79	44.53	48.43	2.4	178939	46.34	46.99	0.2	551584
	128	35.49	55.06	13.94	120	28.56	30.84	2.1	342849	31.63	33.14	1.4	543689
	256	23.76	27.42	4.07	306	20.16	20.68	0.5	691193	20.53	21.80	0.6	1172338
Lake	32	268.93	5188.44	6798.47	53	203.94	204.76	0.6	88728	203.29	204.35	0.5	143206
	64	178.38	203.05	36.12	125	131.64	132.71	0.7	175367	130.84	131.93	0.7	278690
	128	118.89	136.96	20.93	211	85.67	86.33	0.4	354141	86.19	87.04	0.6	574691
	256	76.29	81.71	5.25	428	56.23	56.64	0.4	698147	56.78	57.28	0.4	1135666
Girl	32	152.06	184.92	27.15	58	82.69	83.58	0.6	90522	82.37	82.79	0.4	504507
	64	94.85	111.85	22.00	99	50.62	51.26	0.6	177422	50.06	50.57	0.2	287817
	128	55.38	70.78	10.19	164	30.63	31.84	0.7	351931	30.68	31.58	0.6	585449
	256	28.07	35.21	6.57	379	19.18	19.64	0.8	704688	19.65	19.86	0.3	1094486
Mandrill	32	758.39	5880.82	3821.17	24	373.32	377.12	1.8	91611	373.66	375.28	1.2	148121
	64	393.09	1913.96	2584.09	63	236.34	237.40	0.9	175347	235.88	236.60	0.4	276857
	128	220.56	382.85	159.20	142	152.43	153.69	1.0	343677	151.37	151.83	0.3	564477
	256	134.34	153.95	20.02	394	97.67	98.93	0.5	706416	97.14	97.54	0.2	1091333
Peppers	32	419.31	456.19	39.45	63	230.88	232.32	0.9	91035	227.26	230.62	1.4	145258
	64	189.87	341.02	107.39	95	135.39	137.38	1.4	190334	134.03	135.47	2.3	285673
	128	116.72	193.17	97.40	190	84.74	85.69	0.7	350765	83.74	84.45	0.5	553814
	256	72.96	81.01	9.57	433	55.68	56.29	0.4	686947	54.78	55.09	0.2	1095412
Lenna	32	160.74	204.23	45.96	58	119.79	121.35	1.5	93219	118.82	119.21	0.4	140119
	64	94.94	127.94	36.99	102	73.36	74.73	1.1	180290	72.73	73.24	0.4	273863
	128	58.87	73.93	16.50	182	47.15	47.84	0.5	355805	46.69	46.97	0.2	551584
	256	38.68	40.94	2.10	415	31.33	31.91	0.4	703331	30.97	31.15	0.1	1063209
Landscape	32	135.93	280.86	130.97	61	97.17	100.37	2.8	96169	97.08	98.45	0.9	146573
	64	77.21	138.80	68.66	110	55.63	57.22	1.6	188092	54.22	55.25	0.5	287501
	128	45.84	75.20	26.95	187	32.88	34.75	1.0	355077	32.18	32.76	0.3	568052
	256	26.76	31.35	6.65	457	20.93	21.73	0.4	727046	20.50	20.66	0.1	1109933
Headbands	32	161.00	208.74	50.03	58	113.56	117.07	1.8	106756	115.25	118.88	3.0	143921
	64	100.67	122.54	20.05	100	68.88	70.54	1.6	184585	70.41	74.01	2.1	287435
	128	56.24	74.41	17.65	169	42.81	44.50	1.0	370737	44.15	45.94	1.4	547842
	256	34.79	38.35	2.96	367	27.59	28.20	0.5	710804	28.79	29.64	0.6	1137880
Dessert	32	203.68	304.38	101.49	60	120.48	123.03	1.3	105843	122.06	123.69	1.2	150113
	64	129.26	167.68	43.94	107	67.47	68.56	0.8	181495	68.60	69.30	0.6	291267
	128	72.66	96.75	21.72	199	39.98	40.63	0.5	370856	40.52	41.97	0.7	548795
	256	36.94	43.25	3.99	477	24.46	24.96	0.3	720788	25.39	25.70	0.2	1133866
Snowman	32	188.61	279.52	71.22	65	116.44	119.12	2.2	109983	115.79	118.49	1.9	162902
	64	99.49	145.65	36.93	117	63.36	64.39	0.9	178304	61.85	64.47	1.4	294426
	128	56.97	76.14	26.57	221	36.83	38.01	0.7	362277	37.09	37.92	0.5	564005
	256	32.99	36.73	3.31	510	22.82	23.91	0.6	717805	23.33	23.91	0.3	1117948
Cathedrals	32	98.42	121.41	19.39	56	61.07	62.41	1.6	95577	59.35	60.82	1.0	148623
	64	57.97	78.78	19.62	84	32.95	34.16	0.6	186254	32.89	33.60	0.4	293007
	128	36.78	51.64	12.76	126	20.15	20.65	0.4	365408	19.96	20.32	0.3	564088
	256	19.91	24.09	4.78	297	13.33	13.67	0.2	710510	12.99	13.24	0.2	1149640
Beach	32	263.38	320.03	90.30	52	135.40	137.56	1.3	95092	134.64	136.19	1.3	152183
	64	141.02	183.60	41.15	87	75.03	76.66	1.2	181030	74.23	75.25	0.8	285653
	128	76.25	96.27	16.55	149	45.36	46.05	0.6	362801	44.69	45.47	0.5	576216
	256	46.52	63.89	15.43	332	28.39	29.15	0.7	701736	28.08	28.57	0.3	1125092

To facilitate the comparison of the results, Fig. 11 shows the percentage of error reduction obtained by SFLA-CQ with $step = 10$ with respect to the methods not shown in Fig. 10; in this case lines are used instead of bars to make the figures clearer. As previously described, it can be observed that the proposed method improves the images obtained by the other methods in almost all cases. To facilitate the interpretation of Figs. 10 and 11, the values represented in both figures are included as supplementary material.

To analyze the statistical significance of the improvement obtained by SFLA-CQ with respect to the other methods, the Wilcoxon test was applied (Corder and Foreman, 2009). This test compares two methods to determine that there is no significant difference between their results. In this case, the test was applied twice: the first to compare the average MSE and the second to compare the average execution time. Based on the discussion presented in the previous paragraphs, the Wilcoxon test was applied to compare the results of SFLA-CQ with $step = 10$ and those of each of the other methods. Table 6 shows the results with a significance level equal to 0.05. The p-value obtained in all cases indicates that the differences between each pair of methods compared are significant. When comparing the MSE values, the sums of ranks show that SFLA-CQ is significantly better than the other methods, except PSO+KM and ABC+KM. When the execution time is considered, the results of SFLA-CQ are significantly better than those of KM, PSO+KM and ABC+KM. Therefore, the results of the Wilcoxon test confirm the conclusions presented in this section.

6. Conclusion

This article describes how the Shuffled frog leaping algorithm can be used to reduce the colors of an RGB image. The operations of the basic algorithm have been adapted to solve the color quantization problem. The modified algorithm, called SFLA-CQ, uses the mean squared error

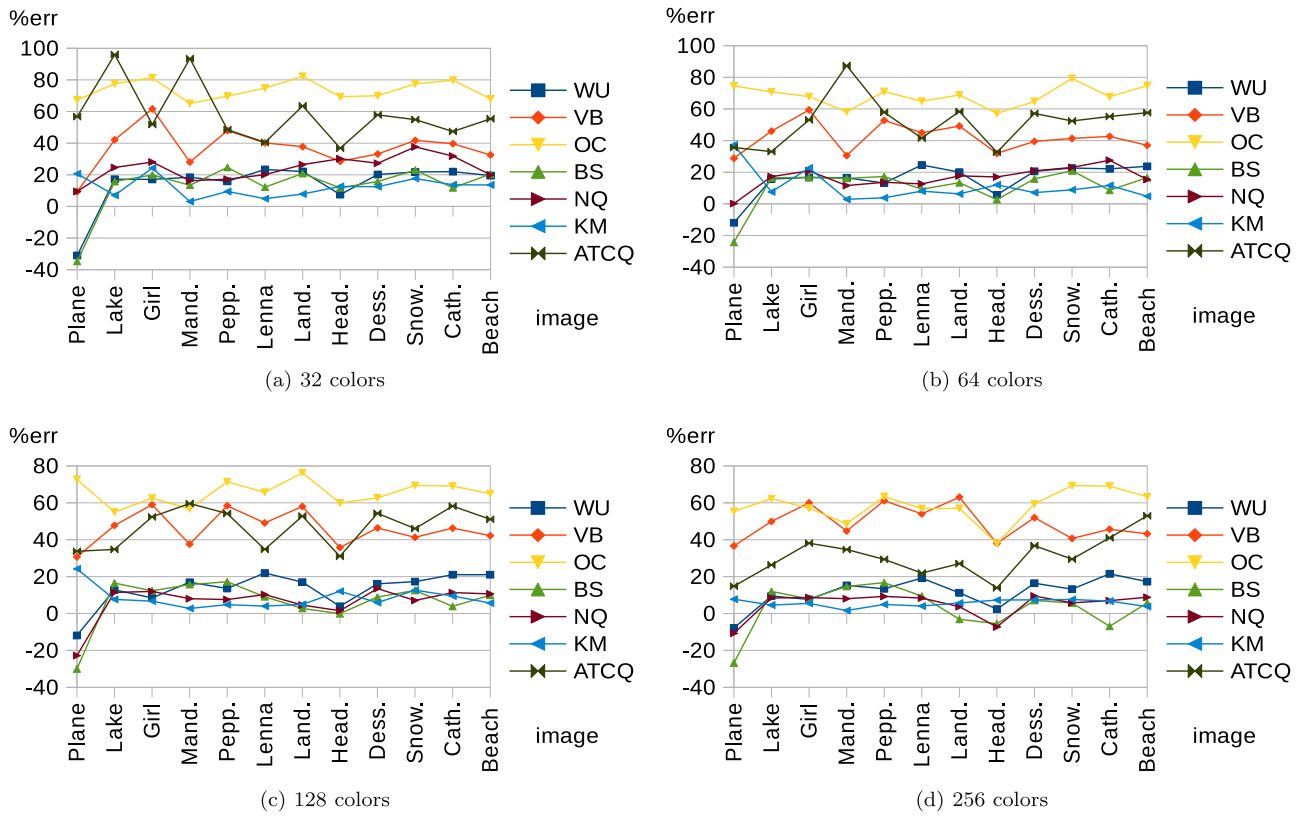


Fig. 11. Percentage of error reduction obtained by SFLA-CQ with $step = 10$ compared to other methods: Wu's method (WU), Variance-based method (VB), Octree (OC), Binary splitting (BS), Neuquant (NQ), K-means (KM), Ant-tree for color quantization method (ATCQ).

Table 6
Results of the Wilcoxon test that compares SLFA-CQ to other color quantization methods: Wu's method (WU), Variance-based method (VB), Octree (OC), Binary splitting (BS), Neuquant (NQ), K-means (KM), Ant-tree for color quantization method (ATCQ), Particle swarm optimization combined with K-means (PSO+KM), Artificial bee colony combined with K-means (ABC+KM). (Z-value: value of the test statistic; p-value: probability value corresponding to the Z-value; $sum+$: sum of positive ranks; $sum-$: sum of negative ranks).

	MSE				Execution time			
Method	Z-value	p-value	$sum+$	$sum-$	Z-value	p-value	$sum+$	$sum-$
WU	−5.4103	0	60.5	1115.5	−6.0308	0	1176	0
VB	−6.0308	0	0	1176	−6.0308	0	1176	0
OC	−6.0308	0	0	1176	−6.0308	0	1176	0
BS	−4.8001	0	120	1056	−6.0308	0	1176	0
NQ	−5.6514	0	37	1139	−6.0308	0	1176	0
KM	−6.0308	0	0	1176	−6.0308	0	0	1176
ATCQ	−6.0308	0	0	1176	−6.0308	0	1176	0
PSO+KM	−6.0308	0	1176	0	−6.0308	0	0	1176
ABC+KM	−6.0308	0	1176	0	−6.0308	0	0	1176

as the objective function and generates a quantized palette that allows the quantized image to be obtained. Given that the size of the images considered influences the execution time of the algorithm, the possibility of working with a subset of pixels of the original image has been analyzed to accelerate the process.

The proposed method has been applied to several images and the results obtained with different values of each parameter of the algorithm have been compared. It is observed that the results corresponding to sampled images that include 10% of the pixels of the original image are good, since the execution time is reduced without greatly reducing the quality of the final image.

The SFLA-CQ method has also been compared to 9 other color quantification methods and, in general, the proposed method obtains better images than the other methods. As expected, the splitting methods that are compared consume less time than the proposed one; however, it should be noted that these methods also obtain worse images. Regarding the clustering-based methods analyzed, it is worth mentioning that there is a significant difference in the execution time between the proposed method and two other swarm-based methods: PSO+KM and ABC+KM. Although both methods obtain better images than SFLA-CQ in some cases, the results are not comparable due to the excessive time required to obtain such results. On the other hand, NQ and ATCQ, which are also clustering-based methods, consume less time than SFLA-CQ, but generate worse images.

Acknowledgments

Funding

This work was supported by the Samuel Solorzano Memorial Foundation of the University of Salamanca, Spain [grant number FS/102015].

Declarations of interest

None.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.engappai.2019.01.002.

References

Alqudami, N., Kim, S.-D., 2016. OpenCL-based optimization methods for utilizing forward DCT and quantization of image compression on a heterogeneous platform. J. Real-Time Image Process. 12 (2), 219–235.

An, N.Y., Pun, C.M., 2014. Color image segmentation using adaptive color quantization and multiresolution texture characterization. Signal Image Video Proc. 8 (5), 943–954.

Azizipanah-Abarghooee, R., Narimani, M.R., Bahmani-Firouzi, B., Niknam, T., 2014. Modified shuffled frog leaping algorithm for multi-objective optimal power flow with FACTS devices. J. Intell. Fuzzy Systems 26 (2), 681–692.

Celebi, M.E., 2009. An effective color quantization method based on the competitive learning paradigm. In: Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition, pp. 876–880.

Celebi, M.E., 2011. Improving the performance of k-means for color quantization. Image Vis. Comput. 29 (4), 260–271.

Chang, C.H., Xu, P., Xiao, R., Srikanthan, T., 2005. New adaptive color quantization method based on self-organizing maps. IEEE Trans. Neural Netw. 16 (1), 237–249.

Cheng, S., Lu, H., Lei, X., Shi, Y., 2018. A quarter century of particle swarm optimization. Complex Intell. Syst. 4, 1–13.

Corder, G.W., Foreman, D.I., 2009. Comparing two related samples: The Wilcoxon signed ranks test. In: Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach. Wiley Online Library, pp. 38–56.

Dekker, A.H., 1994. Kohonen neural networks for optimal colour quantization. Network: Comput. Neural Syst. 5 (3), 351–367.

Eusuff, M.M., Lansey, K., 2003. Optimization of water distribution network design using the shuffled frog leaping algorithm. J. Water Resour. Plan. Manag. 129 (3), 210–225.

Eusuff, M.M., Lansey, K., Pasha, F., 2006. Shuffled frog-leaping algorithm: A memetic meta-heuristic for discrete optimization. Eng. Optim. 38 (2), 129–154.

Fister, I., Yang, X.S., Fister, D., Fister, Jr., I., 2014. Firefly algorithm: A brief review of the expanding literature. In: Cuckoo Search and Firefly Algorithm: Theory and Applications, Studies in Computational Intelligence, vol. 516, Springer, Heidelberg, pp. 347–360.

Fu, X., Wang, C.Y., Chen, C., Wang, C., Jay Kuo, C.C., 2015. Robust image segmentation using contour-guided color palettes. In: Proceedings of the 2015 IEEE International Conference on Computer Vision, pp. 1618–1625.

Garey, M., Johnson, D., Witsenhausen, H., 1982. The complexity of the generalized Lloyd-max problem (corresp.). IEEE Trans. Inform. Theory 28 (2), 255–256.

Gervautz, M., Purgathofer, W., 1990. A simple method for color quantization: Octree quantization. In: Graphics Gems. Academic Press Professional, Inc., San Diego, CA, USA, pp. 287–293.

Ghanbarian, A.T., Kabir, E., Charkari, N.M., 2007. Color reduction based on ant colony. Pattern Recognit. Lett. 28 (12), 1383–1390.

Guo, J., Tang, H., Sun, Z., Wang, S., Jia, X., Chen, H., Zhang, Z., 2015. An improved shuffled frog leaping algorithm for assembly sequence planning of remote handling maintenance in radioactive environment. Sci. Technol. Nucl. Install. 2015, 1–14.

Heckbert, P., 1982. Color image quantization for frame buffer display. In: Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques. ACM, New York, NY, USA, pp. 297–307.

Hu, Y., Su, B., 2008. Accelerated k-means clustering algorithm for colour image quantization. Imaging Sci. J. 56, 29–40.

Kar, A.K., 2016. Bio inspired computing – a review of algorithms and scope of applications. Expert Syst. Appl. 59, 20–32.

Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N., 2014. A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. Artif. Intell. Rev. 42, 21–57.

Kasuga, H., Yamamoto, H., Okamoto, M., 2000. Color quantization using the fast k-means algorithm. Syst. Comput. Japan 31, 33–40.

Kekre, H.B., Natu, P., Sarode, T., 2016. Color image compression using vector quantization and hybrid wavelet transform. Procedia Comput. Sci. 89, 778–784.

Ladgham, A., Sakly, A., Mtibaa, A., 2014. Optimal feature selection based on hybridization of MSFLA and Gabor filters for enhanced MR brain image recognition using SVM. Int. J. Tomogr. Simul. 27 (3), 69–83.

Ladgham, A., Hamdaoui, F., Sakly, A., Mtibaa, A., 2015a. Fast MR brain image segmentation based on modified shuffled frog leaping algorithm. Signal Image Video Process. 9, 1113–1120.

Ladgham, A., Sakly, A., Mtibaa, A., 2015b. Fast and consistent images areas recognition using an improved shuffled frog leaping algorithm. Int. J. Signal Imaging Syst. Eng. 8 (5), 331–342.

Lei, D., Zheng, Y., Guo, X., 2017. A shuffled frog-leaping algorithm for flexible job shop scheduling with the consideration of energy consumption. Int. J. Prod. Res. 55 (11), 3126–3140.

Liu, G.H., Yang, J.Y., Li, Z., 2015. Content-based image retrieval using computational visual attention model. Pattern Recognit. 48 (8), 2554–2566.

Luo, J., Chen, M.R., 2014. Improved shuffled frog leaping algorithm and its multi-phase model for multi-depot vehicle routing problem. Expert Syst. Appl. 41, 2535–2545.

Luo, J., Li, X., Chen, M.R., Liu, H., 2015. A novel hybrid shuffled frog leaping algorithm for vehicle routing problem with time windows. Inform. Sci. 316, 266–292.

Ma, M., Zhu, Q., 2017. Multilevel thresholding image segmentation based on shuffled frog leaping algorithm. J. Comput. Theor. Nanosci. 14 (8), 3794–3801.

Mavrovouniotis, M., Li, C., Yang, S., 2017. A survey of swarm intelligence for dynamic optimization: Algorithms and applications. Swarm Evol. Comput. 33, 1–17.

Mora-Melia, D., Iglesias-Rey, P., Martínez-Solano, F., Muñoz-Velasco, P., 2016. The efficiency of setting parameters in a modified shuffled frog leaping algorithm applied to optimizing water distribution networks. Water 8 (5), 1–14.

Neshat, M., Sepidnam, G., Sargolzaei, M., Toosi, A.N., 2014. Artificial fish swarm algorithm: A survey of the state-of-the-art, hybridization, combinatorial and indicative applications. Artif. Intell. Rev. 42, 965–997.

Omran, M.G., Engelbrecht, A.P., Salman, A., 2005. A color image quantization algorithm based on particle swarm optimization. Inform. 29, 261–270.

Orchard, M.T., Bouman, C.A., 1991. Color quantization of images. IEEE Trans. Signal Process. 39, 2677–2690.

Özdemir, D., Akarun, L., 2002. A fuzzy algorithm for color quantization of images. Pattern Recognit. 35, 1785–1791.

Ozturk, C., Hancer, E., Karaboga, D., 2014. Color image quantization: A short review and an application with artificial bee colony algorithm. Inform. 25, 485–503.

Palomo, E.J., Domínguez, E., 2014. Hierarchical color quantization based on self-organization. J. Math. Imaging Vision 49, 1–19.

Papamarkos, N., Atsalakis, A.E., Strouthopoulos, C.P., 2002. Adaptive color reduction. IEEE Trans. Syst. Man Cybern. B 32, 44–56.

Pérez-Delgado, M.-L., 2015. Colour quantization with ant-tree. Appl. Soft Comput. 36, 656–669.

Pérez-Delgado, M.-L., 2018. Test images for color quantization. http://audax.zam.usal.es/web/mlperez/cq.html (Accessed 21 June 2018).

Ponti, M., Nazaré, T.S., Thumé, G.S., 2016. Image quantization as a dimensionality reduction procedure in color and texture feature extraction. Neurocomputing 173, 385–396.

Prasad, R.D., Kumar, B.S., Ram, K.S., Manoj, B.V., 2016. Content based image retrieval using dominant color and texture features. Int. J. Modern Trends Sci. Technol. 2 (04), 36–41.

Rabie, T., 2017. Color-secure digital image compression. Multimedia Tools Appl. 76 (15), 16657–16679.

Rasti, J., Monadjemi, A., Vafaei, A., 2011. Color reduction using a multi-stage Kohonen self-organizing map with redundant features. Expert Syst. Appl. 38, 13188–13197.

Sarkheyli, A., Zain, A.M., Sharif, S., 2015. The role of basic, modified and hybrid shuffled frog leaping algorithm on optimization problems: A review. Soft Comput. 19, 2011–2038.

Schaefer, G., Zhou, H., 2009. Fuzzy clustering for colour reduction in images. Telecommun. Syst. 40, 17–25.

Scheunders, P., 1997. A comparison of clustering algorithms applied to color image quantization. Pattern Recognit. Lett. 18, 1379–1384.

Shehab, M., Khader, A.T., Al-Betar, M.A., 2017. A survey on applications and variants of the cuckoo search algorithm. Appl. Soft Comput. 61, 1041–1059.

Torkhani, G., Ladgham, A., Sakly, A., Mansouri, M.N., 2017. A novel optimised face recognition application based on modified shuffled frog leaping algorithm. Int. J. Appl. Pattern Recognit. 4, 27–43.

Tripathy, B., Dash, S., Padhy, S.K., 2015. Multiprocessor scheduling and neural network training methods using shuffled frog-leaping algorithm. Comput. Ind. Eng. 80, 154–158.

Verevka, O., Buchanan, J.W., 1995. The local k-means algorithm for colour image quantization. In: Graphics Interface. Canadian Information Processing Society, pp. 128–135.

Wan, S., Prusinkiewicz, P., Wong, S., 1990. Variance-based color image quantization for frame buffer display. Color Res. Appl. 15, 52–58.

Wang, C.H., Lee, C.N., Hsieh, C.H., 2007. Sample-size adaptive self-organization map for color images quantization. Pattern Recognit. Lett. 28, 1616–1629.

Wang, X., Liu, S., Li, Q., Liu, Z., 2018. Underwater sonar image detection: A novel quantum-inspired shuffled frog leaping algorithm. Chin. J. Electron. 27 (3), 588–594.

Weber, A., 2018. USC-SIPI image database, http://sipi.usc.edu/database/database.php/. (Accessed 21 June 2018).

Wen, Q., Celebi, M.E., 2011. Hard versus fuzzy c-means clustering for color quantization. EURASIP J. Adv. Signal Process. 118, 1–12.

Wu, X., 1991. Efficient statistical computations for optimal color quantization. In: Graphics Gems vol. II. Academic Press, pp. 126–133.

Wu, X., 1992. Color quantization by dynamic programming and principal analysis. ACM Trans. Graph. 11, 348–372.

Xiao, Y., Li, B.H., Lin, T., Hou, B., Shi, G., Li, Y., 2016. A self-adaptive shuffled frog leaping algorithm for multivariable PID controller's optimal tuning. In: Theory, Methodology, Tools and Applications for Modeling and Simulation of Complex Systems. Springer, Singapore, pp. 3–16.

Xunli, F.A.N., Feiefi, D.U., 2015. Shuffled frog leaping algorithm based unequal clustering strategy for wireless sensor networks. Appl. Math. 9 (3), 1415–1426.

Zain Eldin, H., Elhosseini, M.A., Ali, H.A., 2015. Image compression algorithms in wireless multimedia sensor networks: A survey. Ain Shams Eng. J. 6 (2), 481–490.