Vector quantization using the improved differential evolution algorithm for image compression

Sayan Nag1

Received: 1 November 2017 / Revised: 29 November 2018 / Published online: 16 January 2019
© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract
Vector quantization (VQ) is a popular image compression technique with a simple decoding architecture and high compression ratio. Codebook designing is the most essential part in vector quantization. Linde–Buzo–Gray (LBG) is a traditional method of generation of VQ codebook which results in lower PSNR value. A codebook affects the quality of image compression, so the choice of an appropriate codebook is a must. Several optimization techniques have been proposed for global codebook generation to enhance the quality of image compression. In this paper, a novel algorithm called IDE-LBG is proposed which uses improved differential evolution algorithm coupled with LBG for generating optimum VQ codebooks. The proposed IDE works better than the traditional DE with modifications in the scaling factor and the boundary control mechanism. The IDE generates better solutions by efficient exploration and exploitation of the search space. Then the best optimal solution obtained by the IDE is provided as the initial codebook for the LBG. This approach produces an efficient codebook with less computational time and the consequences include excellent PSNR values and superior quality reconstructed images. It is observed that the proposed IDE-LBG find better VQ Codebooks as compared to IPSO-LBG, BA-LBG and FA-LBG.

Keywords Image compression · Vector quantization · Codebook · Improved differential evolution (IDE) algorithm · Linde–Buzo–Gray (LBG) algorithm · Improved particle swarm optimization (IPSO) algorithm · Bat algorithm (BA) · Firefly algorithm (FA)

Sayan Nag
nagsayan112358@gmail.com

1 Department of Electrical Engineering, Jadavpur University, Kolkata, India
1 Introduction

The major role of image compression lies in medical sciences, internet browsing and navigation applications, TV broadcasting and so on. With the advancement of science and technology, the storage space requirement and the need for reduction in transmission time for digital images are becoming essential concerns. The transmission bandwidth being limited creates a problem. An efficient and proper image compression technique is of prime requirement to tackle the problem of limited bandwidth. This area of study attracted many researchers over the past decades who have suggested several techniques for image compression. Vector quantization (VQ) technique outperforms other techniques such as pulse code modulation (PCM), differential PCM (DPCM), Adaptive DPCM which belongs to the class of scalar quantization methods. Vector quantization (VQ) [1, 2], one of the most popular lossy image compression methods is primarily a c-means clustering approach widely used for image compression as well as pattern recognition [3, 4], speech recognition [5], face detection [6] speech and image coding because of its advantages which include its simple decoding architecture and the high compression rate it provides with low distortion—hence its popularity. Vector Quantization involves the following three steps: (1) Firstly, the image in consideration is divided into non-overlapping blocks commonly known as input vectors. (2) Next, a set of representative image blocks from the input vectors is selected referred to as a codebook and each representative image vector is referred to as a code-word. (3) Finally, each of the input vectors is approximately converted to a code-word in the codebook, the corresponding index of which is then transmitted.

Codebook training is considered sacrosanct in the process of Vector Quantization, because a codebook largely affects the quality of image compression. Such significance of Codebook training gave new impetus to many researchers leading to the proliferation of researches to design codebook using several projected approaches. Vector Quantization methods are categorized into two classes: crisp and fuzzy [7]. Sensitive to codebook initialization, Crisp VQ follows a hard decision making process. Of this type, C-Means algorithm is the most representative of all. Linde et al. propounded the famous Linde–Buzo–Gray (LBG) algorithm in this respect. Starting with the smallest possible codebook size the gradual parlay in the size occurs by using a splitting procedure [8]. By implanting specific functions (utility measures) in the learning process the performance of the LBG algorithm is significantly enhanced. Fuzzy VQ incorporates fuzzy C-Means algorithm. The approach is similar to that of fuzzy cluster analysis. Each training vector is posited to be a property of multiple clusters with membership values which makes the learning a soft decision making process [9]. The drawback of LBG Algorithm is that it gets stuck in a local optimum. In order to overcome such problem, a clustering algorithm known as enhanced LBG is proposed [10]. By changing the lower utility code-words to the one with higher utility the problem of local optimal with LBG is overcome.

Recently researches involve the association of the evolutionary optimization algorithms with the LGB algorithm. This conglomeration for design the codebook
has a significant impact in improving the results of the LGB algorithm. Rajpoot et al. [11] formulated a codebook using an Ant Colony Optimization (ACO) algorithm yielding fascinating results thereby clearly depicting the improvement in the results obtained by the hybridization of ACO with LBG. Codebook is optimized abreast appointing the vector coefficients in a bidirectional graph; next, a appropriate implementation of placing pheromones on the edges of the graph is expounded in the ACO-LBG Algorithm. The speed of convergence of the ACO-LBG algorithm is proliferated by Tsaia et al. [12] by discarding the redundant calculations. Particle Swarm Optimization (PSO) is also an adaptive swarm optimization approach based on updating the global best (gbest) and local best (lbest) solutions. The ease of improvement and fast convergence to an expected solution attracted many researchers to apply PSO for solving optimization problems. Particle swarm optimization along with vector quantization [13] overcomes the drawbacks of the LBG algorithm. Evolutionary fuzzy particle swarm optimization algorithm [14] is an efficient and robust algorithm in terms of performances compared to that of the LBG learning algorithms. Quantum particle swarm algorithm (QPSO) is yet another PSO put forward by Wang et al. [15] for solving the 0–1 knapsack problem. Applied together with LBG, the QPSO-LBG algorithm outperforms LBG algorithm as well.

Yang et al. devised a new method for image compression which involves an Evolutionary clustering based vector quantization primarily focused on One-Step Gradient Descent Genetic Algorithm (OSGD-GA). The particular algorithm is formulated for optimizing the codebooks of the low-frequency wavelet coefficient by expounding the consequence of every coefficient involved abreast employing fuzzy membership values for automatic clustering [16]. The Firefly Algorithm (FA) is an efficient Swarm Intelligence tool which is largely applied to many engineering design problems nowadays. Firefly Algorithm is inspired by the social activities of fireflies. In presence of a firefly with brighter intensity, the one with lower intensity value move towards the former and in the absence of the brighter firefly then one with lower intensity moves in a random fashion. Horng [17] successfully applied a newly developed Firefly Algorithm (FA) for designing the codebook for vector quantization. An efficient algorithm by the name of Honey Bee Mating Optimization (HBMO) is applied for codebook generation yielding a good quality reconstructed image with modicum of distortion in Image Compression Problems [18]. Dynamic programming is used as an efficient optimization technique for layout optimization of interconnection networks by Tripathy et al. [19]. Tsolakis et al. [20] presented a fast fuzzy vector quantization technique for compression of gray scale images. Based on a crisp relation, an input block is assigned to more than one code-word following a fuzzy vector quantization method [21]. LBG Algorithm alone cannot guarantee optimum results since the initialization of the codebook has a significantly impact on the quality of results. Thus, an attempt has been made in this respect where an Improved PSO-LBG (IPSO-LBG) applied to design codebook for Vector Quantization showed fascinating results when compared to other algorithms like QPSO-LBG and PSO-LBG [22]. Contextual region is encoded giving high priority with high resolution Contextual vector quantization (CVQ) algorithm [23]. Huanga et al. [24] devised a dynamic learning vector– scalar quantization for compressing ECG images. Bat
Algorithm (BA) is yet another advanced nature inspired algorithm which is based on the echolocation behavior of bats with variation in the pulse rates of loudness and emission. BA in association with LBG is applied as BA-LBG for fast vector quantization in image compression [25]. Results obtained from BA-LBG when compared with QPSO-LBG, HBMO-LBG, PSO-LBG and FA-LBG showed that BA-LBG outperforms the other mentioned algorithms. Another efficient VQ scheme is developed where optimal codebooks are generated by employing a bacterial foraging optimization (BFO) technique, which mimics the foraging behavior of *Escherichia coli* generating high quality reconstructed images with high PSNR values [26]. A cuckoo search algorithm based vector quantization (CS-LBG) is proposed for image compression which generates efficient codebooks by varying all possible parameters of CS leading to intensification and diversification of the algorithm [27]. Although, the PSNR value using this approach is enhanced and the number of parameters to be tuned is less, the algorithm suffers from slower convergence issues.

Optimization problems are applied in various domains of Engineering Design, Structural Optimization, Economics and Scheduling Assignments. These problems have some mathematical models and Objective Functions. Two varieties of such problems exist: Unconstrained (without constraints) and Constrained (with Constraints) involving both continuous as well as discrete variables. They are usually non-linear in nature. The task of finding the optimal solutions is difficult with several restraints being active at the global optima. Traditional methods for solving these problems are Gradient Descent, Dynamic Programming and Newton Methods. But they are computationally inefficient. This mark the advent of the meta-heuristic algorithms which provide feasible solutions in a reasonable amount of time. The list of meta-heuristics include Genetic Algorithm (GA) [28], Particle Swarm Optimization (PSO) [29], Gravitational Search Algorithm (GSA) [30], Ant Colony Optimization (ACO) [31, 32], Stimulated Annealing (SA) [33, 34], Plant Propagation Algorithm (PPA) [35, 36] and so on [37, 38].

Differential Evolution (DE) is one such metaheuristic algorithm used for fast convergence and less computation time. It optimizes a problem by trying to improve a candidate solution through generations based on a fitness function and using a crossover strategy which is different from that of the Genetic Algorithm. DE is modified to Improved DE (IDE) and used to generate high quality codebook with LBG for vector quantization for image compression. The sole LBG algorithm has an innate deficit which makes it a hard decision-making strategy thereby disregarding the chances of a training vector being in the possession of more than one codebook vector. In particular, this scheme also suffers from high computation burden. Thus coupling of LBG with meta-heuristic algorithms is an efficient option. With significantly larger value of particle velocity, PSO-LBG has a tendency to encounter instability in convergence [39]. Unlike PSO-LBG, IPSO-LBG can dynamically adapt the different coefficients, namely, individual, social coefficients and the weight value of the evolution function [22]. But, here again many parameters are required to be adjusted. FA-LBG similarly encounters a difficulty when there is a paucity in the number of significant brighter fireflies within the search space [40]. BA-LBG though performing better than FA-LBG in terms of convergence speed, the number of parameters to be tuned exceeds when compared with that of FA-LBG [25]. Differential Evolution, unlike other algorithms (PSO, FA, GA) is
simple structured and easy-to-implement algorithm. But it has a drawback in dealing with high dimensional complex problems that it gets stuck in local optimum resulting in premature convergence. The performance depends on the selection of the scale or mutation factor which determines the exploitation and exploration abilities. So a proper selection of this parameter is to be done to enhance the performance of the algorithm. The aim of the paper is emphasized on two facets: firstly, designing a codebook giving a globally optimal solution and secondly, lessening the computational burden. Keeping these facets in mind, the IDE-LBG algorithm, an improved version of DE is developed. A comparison with other algorithms like IPSO, BA and FA clearly reveals that IDE outperforms all of the other mentioned algorithm in terms of efficiency as judged from the PSNR values of the images.

The paper is organized as follows: In Sect. 2, codebook design using generalized LBG Vector Quantization Algorithm is discussed. In Sect. 3, the proposed method of IDE-LBG Algorithm is presented. Section 4 includes the experimental results, discussions and comparisons. Eventually, Sect. 5 contains the concluding part.

2 Methods of codebook design for vector quantization

The vector quantization (VQ) is a block (vector) coding technique which is to be optimized for image compression. Designing a proper codebook lessens the distortion between reconstructed image and original image with a modicum of computational time. Let us consider an Image of size $N \times N$ which is to be vector quantized. It is then subdivided into $N_b$ blocks with size $n \times n$ pixels, where, $N_b = \left( \frac{N}{n} \right)^2$. These subdivided image blocks are known as training vectors each of which are of size $n \times n$ pixels are represented with $X_i$ (where $i = 1, 2, 3, \ldots, N_b$). In the process of Codebook design, a Codebook has a set of code words, where the $i$th Code-word is represented as $C_i$ (where $i = 1, 2, \ldots, N_c$) where $N_c$ is the total number of Code-words in Codebook. Every subdivided image vector is approximately represented by the index of a Code-word using a closest match technique. The approach is based on the minimum Euclidean distance between the vector and corresponding code-words. In the decoding phase, the same codebook is used to translate the index back to its corresponding Code-word for image reconstruction. Then the distortion between the actual and the reconstructed image is calculated. The distortion $D$ between training vectors and the codebook is given as:

$$D = \frac{1}{N_c} \sum_{j=1}^{N_c} \sum_{i=1}^{N_b} u_{ij} \times ||X_i - C_j^2||$$

Subject to the following restraints:

$$D = \sum_{j=1}^{N_c} u_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, N_b\}$$

$$D = \sum_{j=1}^{N_c} u_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, N_b\}$$
The other two necessary conditions which are to be followed for an optimal vector quantizer are:

- The partition \( R_j \forall j = 1, 2, 3, \ldots, N_c \) must meet the following criterion:
  \[
  R_j \supset \{ x \in X : d(c, C_j) < d(x, C_k), \ \forall k \neq j \}
  \]
- The Code-word \( C_j \) should be defined as the centroid of \( R_j \) such that,
  \[
  C_j = \frac{1}{N_j} \sum_{i=1}^{N_j} x_i, \ \forall x_i \in R_j
  \]

where \( N_j \) is the net number of vectors which belong to \( R_j \).

### 2.1 LBG vector quantization algorithm

The procedure for the most commonly used method for codebook design in Vector Quantization, the Lloyd-Buzo-Gray (LBG) Algorithm is provided as follows:

Steps:

1. Initialization: Iteration counter is set as \( m = 1 \) and the initial distortion \( D_1 = 0 \). Initial codebook \( C_1 \) is of size \( N \).
2. One block (vector) from the set of training vectors is selected and compared with all code-words from codebook, \( C_{m-1} = \{ Y_i \} \) to find the closest code-word using nearest neighbor condition and then the code-word set is added.
3. Once all the blocks are grouped into sets of code-words using the above mentioned step, the centroids of all the code-word sets are evaluated to give an improved codebook \( C_m \).
4. Next, the total distortion of all code-word sets from the respective centroid points are calculated. The average distortion is given as \( D_m \).
5. If \( |(D_{m-1} - D_m)| \leq \varepsilon \), the algorithm terminates and the current codebook is recorded as the final answer. Otherwise, \( m = m + 1 \), the current codebook is used as the initial codebook for step 1 and the entire process is then repeated.

LBG Algorithm ensures the reduction in distortion with increase in iteration number. But there is no assurance regarding the resulting codebook, whether it will become an optimum one or not. It is seen that the initial condition of the codebook has a significant effect on the results. So the initial codebook selection is of prime importance in the LBG Algorithm. Thus several strategies are
proposed to combine efficient heuristics along with LBG Algorithm, thereby tackling the problem of initial codebook selection with these attempts.

2.2 FA-LBG algorithm

Introduced by Yang in 2008, the Fire-fly Algorithm (FA) got its motivation from the flashing motifs and traits of fireflies [38]. The brightness of a firefly is related to the value of the cost function. The firefly with lower fitness value (less bright) approaches towards the firefly having higher fitness value (more bright). FA-LBG algorithm is better than the sole LBG algorithm since the reconstructed images generated from the former are of higher quality than those generated from the latter. In the FA-LBG algorithm the codebooks are considered as fireflies. The steps are provided below in details:

1. The LBG algorithm is run once and the codebook obtained is assigned as a brighter codebook.
2. Parameters $\alpha$, $\beta_0$ and $\gamma$ are initialized. Rest of the codebooks are initialized randomly.
3. Fitness values of each codebook are computed using the following equation.

\[
Fitness(C) = \frac{1}{D(C)}
\]  

where $D(C)$ is already computed using Eq. (1).

4. If there exists a codebook with lower fitness value, then it moves towards the brighter codebook as governed by the following Eqs. (7)–(10). For the following equations, $X_i$ is a randomly chosen codebook and $X_j$ is the brighter codebook.

\[
r_{ij} = \sqrt{\sum_{k=1}^{N_c} \sum_{h=1}^{L} (X^h_{ik} - X^h_{jk})^2}
\]  

\[
\beta = \beta_0 \cdot \exp (-\gamma \cdot r_{ij})
\]  

\[
X^h_{jk} = (1 - \beta)X^h_{ik} + \beta X^h_{jk} + u_{jk}
\]  

where $u_{ij}$ is random number between 0 and 1, $k = 1, 2, \ldots, N_c$, $h = 1, 2, \ldots, L$.

5. If the chosen firefly is unable to notice brighter fireflies in parameter space, then its motion is governed by the following equation:

\[
X^h_{ik} = X^h_{ik} + u^h_{jk}
\]  

where $k = 1, 2, \ldots, N_c$ and $h = 1, 2, \ldots, L$.

6. Until stopping condition is reached steps 3–5 are repeated.
2.3 BA-LBG algorithm

The Bat algorithm (BA) is a nature inspired metaheuristic algorithm for global optimization developed by Yang [41]. It mimics the echolocation behavior of bats with two regulating parameters, namely, pulse rate and loudness. Using the BA-LBG algorithm, the peak signal to noise ratio (PSNR) and the quality of the reconstructed image is better than that given by the LBG as well as FA-LBG algorithm. Also BA-LBG has faster convergence than that of FA-LBG algorithm [25]. The codebooks are considered as the bats in the BA-LBG algorithm. The steps of BA-LBG algorithm are as follows:

1. The number of codebooks ($N$), loudness ($A$), velocity ($V$), pulse rate ($R$), minimum frequency ($Q_{min}$) and maximum frequency ($Q_{max}$) are initialized first.
2. LBG algorithm is run once and the codebook obtained is assigned as one of initial codebooks $X_1$ and other codebooks $X_i$ ($i=2, 3, \ldots, N-1$) are selected in a random fashion.
3. Fitness of all codebooks are computed and the fittest codebook is assigned to be $X_{best}$.
4. Position of each element of each codebook is updated as in Eq. (13) by modifying the frequency as in Eq. (11) and updating the velocities as in Eq. (12).

$$Q_i(t+1) = Q_{max}(t) + (Q_{min}(t) - Q_{max}(t)) \ast R$$  \hspace{1cm} (11)

$$v_i(t+1) = v_i(t) + (X_i(t) - X_{best}(t)) \ast Q_i(t+1)$$  \hspace{1cm} (12)

$$X_i(t+1) = X_i(t) + v_i(t)$$  \hspace{1cm} (13)

5. Step size for random walk ($w$) is generated randomly within 0–1. If it is greater than pulse rate ($R$) then the codebooks are moved around the best codebook following the Eq. (14):

$$X_i(t+1) = X_{best}(t) + w \ast R$$  \hspace{1cm} (14)

6. A number is generated randomly. If it is found to be less than the loudness parameter and the new codebook is found to be fitter than the old codebook, the new codebook is accepted.
7. The codebooks are ranked according to their fitness scores and the current best $X_{best}$ is obtained.
8. Step 2–6 are repeated until stopping condition is reached.

2.4 IPSO-LBG algorithm

The PSO was proposed by Kennedy and Eberhart [29] which mimics the social behavior of bird flocking or fish schooling. In existence, there are basically two categories of PSO models: gbest and lbest models. PSO gbest model was used by Chen et al. [42] to design a codebook where the result of a LBG algorithm
was considered as the initial gbest particle thereby proliferating the speed of convergence of the algorithm. In PSO, codebooks or particles (as in PSO) undergoes modification in their values based on their previous experiences. The particle is constructed by $N_c \times N_b$ pixels, where $N_c$ is the size and $N_b$ is the length of the codebook. The improvisation in the PSO is introduced thereby promoting the diversity of particles where the weight value, individual coefficient and social coefficient are randomly generated and updated. In effect, the particles are allowed to move nonlinearly covering all the searching space [22]. The steps of the PSO algorithm is as follows:

1. The LBG algorithm is run once and the codebook obtained is assigned as the global best codebook (gbest) for the next IPSO Algorithm to be followed.
2. The rest codebooks are initialized with random numbers and their corresponding velocities.
3. Next, the fitness values are computed for each codebook.

$$Fitness(C) = \frac{1}{D(C)}$$ (15)

where $D(C)$ is already computed using Eq. (1).
4. If codebook new fitness value is better than old fitness (pbest), then its corresponding new fitness is assigned as pbest.
5. The highest fitness value among all the codebooks is chosen and if it is better than gbest, then gbest is replaced with the highest fitness value.
6. Velocities are updated for each particle along with their position.

$$v_{ij}^{n+1} = w_n * v_{ij}^n + c_1^n * r_1^n * (pbest_{ij}^n - x_{ij}^n) + c_2^n * r_2^n * (gbest_{ij}^n - x_{ij}^n)$$ (16)

$$w^n = w_{min} + \left( \frac{\text{iter}_{max} - \text{iter}}{\text{iter}_{max}} \right)^{\alpha} (w_{max} - w_{min})$$ (17)

$$c_1^n = c_{1min} + \left( \frac{\text{iter}_{max} - \text{iter}}{\text{iter}_{max}} \right)^{\beta} (c_{1max} - c_{1min})$$ (18)

$$c_2^n = c_{2max} + \left( \frac{\text{iter}_{max} - \text{iter}}{\text{iter}_{max}} \right)^{\gamma} (c_{2min} - c_{2max})$$ (19)

where $j$ is the number of solutions, $i$ is position of the particle, $w$, $c_1$, $c_2$ are weight, individual and social learning rates respectively. $r_1$ and $r_2$ are random numbers in the interval $[0, 1]$. $\alpha$, $\beta$, $\gamma$ are uniformly distributed random numbers in the interval $[0, 2]$.
7. Until a stopping criterion is satisfied Steps 3–7 are repeated.
3 Proposed IDE-LBG vector quantization algorithm

FA coupled with LBG was designed to produce near global codebook but it fails to perform in situations where there is a scarcity of significant brighter fireflies in the search domain [40]. IPSO coupled with LBG can generate an efficient codebook but many parameters are required to be adjusted [22]. BA-LBG gives a global codebook and is faster than FA-LBG. But, here again the number of parameters to be tuned is more when compared with that of the FA-LBG [25]. So IDE-LBG is developed that gives a global codebook in minimum time.

Differential Evolution (DE) is a powerful population-based efficient global optimization technique which was posited by Storn and Price [43]. It has been successfully applied to diverse fields including pattern recognition, communication and so on. In our algorithm we have used the DE/current to best/1 scheme. We have modified DE to IDE with certain modifications in the mutation strategy and in the boundary control strategy. In the mutation strategy we have considered a different scaling factor. The boundary control strategy is improved such that when the limits are crossed then based on a probability there can be either of the two possibilities: (1) the solutions are adjusted within the boundary limits by making the values equal to the values of the limits, or, (2) by randomly generating a fresh candidate within the bounds and replacing the one which violated the limits.

The training image vectors or blocks are demarcated into different groups. One block from each group is selected at random as the initial candidates. A candidate is constructed by \( N_c \times 16 \) pixels, where \( N_c \) is the size of a codebook. In the process, all the candidates of the entire population are moved nonlinearly to cover all searching space, a process known as exploration. In each generation, \( G \), IDE uses the mutation and crossover operations to engender a trial vector \( U_{i}^{G} \) for each individual vector \( X_{i}^{G} \), known as target vector in the present population. The mutation and crossover operations along with the fitness function are described below:

(a) Mutation:

There exists an associated mutant vector, \( V_{i}^{G} = \{ v_{1i}^{G}, v_{2i}^{G}, \ldots, v_{di}^{G} \} \) for each target vector \( X_{i}^{G} = \{ x_{1i}^{G}, x_{2i}^{G}, \ldots, x_{di}^{G} \} \) at generation \( G \) in the current population where \( d \) is the dimension of the search space. The DE/current to best/1 scheme dictates the mutation operations as follows:

\[
V_{i}^{G} = X_{i}^{G} + F \times (X_{\text{best}}^{G} - X_{i}^{G}) + F \times (X_{r_{1}}^{G} - X_{r_{2}}^{G})
\]

where \( r_{1} \) and \( r_{2} \) are random integers which are essentially mutually different and lies in the range \([1, NP]\). \( F \) is called the Weighting Factor, \( F=3 \times \text{randn} \times p \), where \( \text{randn} \sim N(0,1) \), \( N \) being the standard normal distribution. \( X_{\text{best}}^{G} \) is by far the fittest individual in the population at generation \( G \).

\[ \square \] Springer
(b) Crossover operation:

Post-mutation period, a binomial crossover strategy is adopted to generate a trial vector $U_i^G = \{u_{G1}^i, u_{G2}^i, \ldots, u_{Gd}^i\}$, where $d$ is the dimension. $CR$ is the crossover operator, a value between 0 and 1. We have considered $CR=0.9$. So, $\forall j \in \{1, 2, \ldots, d\}$

$$U_i^G = \begin{cases} v_{ji}^G, & \text{if } (\text{rand}_j(0, 1) \leq CR, \text{ or }, j = j_{\text{rand}}) \\ x_{ji}^G, & \text{otherwise} \end{cases}$$  \hspace{1cm} (21)

where $v_{ji}$ belongs to the mutant vector $V_i$ and $x_{ji}$ belongs to the target vector $X_i$.

(c) Fitness function:

Size of the testing images is $N \times N$, $I$ is the recompressed image pixels, and $\bar{I}$ is the compressed image pixels. The fitness function is given in the following set of Eqs. (8–10). First Mean Square Error (MSE) is calculated, then Peak Signal to Noise Ratio (PSNR) is evaluated which is nothing but the fitness function of the problem.

$$MSE = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (I_{ij} - \bar{I}_{ij})^2}{N \times N}$$  \hspace{1cm} (22)

$$PSNR = 10 \times \log \left( \frac{255^2}{MSE} \right)$$  \hspace{1cm} (23)

$$f_{\text{fitness}} = \max(PSNR)$$  \hspace{1cm} (24)

3.1 Motivation for modifications

Even though Differential Evolution is simple structured algorithm, it has a drawback in dealing with high dimensional complex problems that it gets stuck in local optimum resulting in premature convergence. The performance depends on the selection of the scale or mutation factor which determines the exploitation and exploration abilities. So a proper selection of this parameter is to be done to enhance the performance of the algorithm which serves as the motivation for modification of the traditional algorithm. The modifications in the IDE algorithm with respect to the traditional DE approach lie in the mutation strategy and in the boundary control strategy. Both of them contribute to the betterment of the algorithm both in terms of exploration and exploitation. The boundary control strategy is adopted to enhance the performance of the algorithm such that when the limits are crossed then based on a probability there can be either of the two cases: (i) the solutions are adjusted within the boundary limits by making the values equal to the values of the limits, or, (ii) by randomly generating a fresh candidate within the bounds and replacing the one which violated the limits. The value of this probability is taken to be 0.5 and a random number is generated which if less than this probability case (i) is executed,
else case (ii) is executed. The boundary controlled strategy being already mentioned, the mutation strategy is explained below.

Considering the mutation strategy, the mutation or scale factor $F$ is the deciding factor of the exploration and exploitation capabilities of the algorithm. Different strategies have been adopted throughout the years to find a suitable value or rather values of the scale factor for the best performance of the DE algorithm. Some suggested a value of 0.5 while others recommended an interval of $[0.4, 0.8]$ for choosing the mutation factor [44, 45]. So, the choice of this scale factor, $F$ has always been an area of research. Considering the adopted mutation strategy, we selected $F = 3 \times \text{randn} \times p$, where $\text{randn} \sim N(0,1)$, $N$ being the standard normal distribution and $p$ is a linearly decreasing factor. The value of $p$ gradually decreases from 0.7 to 0.3 during the span of ‘$n$’ generations. The values of 0.7 and 0.3 have been selected on two conditions, firstly, by setting the random number to its expected value of 0.5 to start with we can achieve a scale factor of value 1.05 and to end with a scale factor of value 0.45 and secondly by trying various options of the boundary values of this decreasing factor for this problem, the considered interval gave the best results, that is, essentially these values have been determined experimentally. Initially a higher value of the scale factor contributes to better exploration of the search space and in a latter period a lesser value of the scale factor improves the exploitation properties of the algorithm. Now, if the scale factor only involves the linearly decreasing factor then the algorithm may result in premature convergence. So introduction of a random variable along with a decreasing factor allows for the probabilistic fluctuations of the scale factor to avoid the local optima also contributing to better exploitation abilities.

### 3.2 Steps of IDE-LBG algorithm

The detailed steps for IDE-LBG algorithm for finding optimal codebook are described as follows:

1. Parameters and Functions: Fitness function $= f_{\text{fitness}}$, iteration parameter $=\text{iter}= 1$, number of generations $=N_{\text{Gen}}$, codebook size $= N_c$, and population size $=NP$.
2. Testing images are divided into $N_b$ non-overlapping image blocks or input vectors $(b_k; k = 1, 2, \ldots, N_b)$. They are put into a training set $tSet$, which of each block is $n \times n$ pixels, the total pixels of each image block is evaluated and the blocks are sorted in the increasing order of pixel number. Here we have considered $n = 4$.
3. $N_b$ blocks are separated into $N_c$ groups according to the index number of blocks, each group having $\frac{N_b}{N_c}$ number of blocks. Blocks from groups are selected at random to form the Code-words to form the initial population of size $NP$.
4. At current iteration $G$, the individual candidates of the population are updated using the mutation and crossover strategies using Eqs. (6) and (7). Boundary control is applied so that the solutions remain within the limits of the search space.

© Springer
5. The fitness of the off-springs are calculated and compared. If any individual is better than $X_{\text{best}}^G$, $X_{\text{best}}^{G+1}$ is updated with value equal to that of the fittest individual, otherwise $X_{\text{best}}^{G+1} = X_{\text{best}}^G$.

6. If current value of iteration parameter $\text{iter}$ reaches $N_{\text{Gen}}$, then $X_{\text{best}}^{G_{\text{Gen}}}$ is considered as the optimal solution obtained to be used as an initial codebook for the LBG algorithm, otherwise $\text{iter} = \text{iter} + 1$ and the process is repeated again from step 4.

7. Global best candidate solution $X_{\text{best}}^{G_{\text{Gen}}}$ obtained from IDE is used as an initial codebook for LBG Algorithm where each Code-word essentially represents the centroid point of current Code-word set.

8. The LBG Algorithm is then applied whose steps are provided in detail in the previous section (Sect. 2).

4 Experimental results

In this section, we performed computational experiments and compared the results to show the efficacy of our newly propounded IDE-LBG approach for VQ for Image Compression. All the simulations are carried out in MATLAB 2013a in a workstation with Intel Core i3 2.9 GHz processor. Five tested 512 × 512 grayscale images, namely, ‘LENA’, ‘PEPPER’, ‘BABOON’, ‘GOLDHILL’ and ‘LAKE’, were selected to be used as training images to train different size of Codebooks and as the basis of comparison of proffered IDE-LBG with other algorithms. The selected images are compressed with IDE-LBG, IPSO-LBG, BA-LBG and FA-LBG and the results are compared. The testing image which is to be compressed is subdivided into non-overlapping image blocks of size 4 × 4 pixels. Each subdivided image block is a training vector of (4 × 4) 16 dimensions. The training vectors to be encoded are $\left( \frac{512 \times 512}{4} \right)$ 16384 in number. Eight different codebook sizes are used for comparison with values 8, 16, 32, 64, 128, 256, 512 and 1024. The members of the initial population of IDE is chosen at random from each group. Eventually, the best solution (Codebook) obtained by the IDE is used as the initial Codebook for the LBG algorithm-the problem of LBG with random initialization as mentioned before is

| Table 1 Parameters for FA-LBG |
|-------------------------------|
| No. of runs | 15 |
| Alpha ($\alpha$) | 0.01 |
| Upper bound of search space | Maximum gray level present in the image |
| Lower bound of search space | Minimum gray level present in the image |
| Population size | 30 |
| Beta minimum ($\beta_0$) | 1 |
| Gamma ($\gamma$) | 1 |
overcome. We have considered bitrate per pixel (bpp) to evaluate the data size of the compressed image for different codebook sizes and the Peak Signal to Noise Ratio (PSNR) values are calculated for individual codebook sizes. Then the plots of PSNR
values versus corresponding bpp values give a clear picture of the quality of reconstructed images with different algorithms considered, thus giving an idea about the effectiveness of the proposed IDE-LBG algorithm.

$$bpp = \frac{\log_2 N_c}{k}$$ (25)

where $N_c$ is the size of the codebook and $k$ is the size of the non-overlapping image bloc.

Tables 1, 2, 3, 4 and 5 show the experimental setup for simulating the various algorithms considered for comparison. The total number of independent runs considered is 15, that is, all the experiments were run 15 times and the average values of all these 15 runs are considered for comparison. The stopping criteria is determined by the maximum number of fitness evaluations. The parameters of the algorithms used for the simulation and comparison purposes are selected experimentally. Convergence analysis of various algorithms give an idea of the possible ranges of the parameters which are to be tuned for the algorithms to converge. Dynamical system theories give a suitable approach to analyze various algorithms only to verify the importance of the parameter values. System matrices can be constructed from the updating equations and sufficient conditions and parameter ranges can be obtained considering the Eigen values of the system matrices [46, 47]. The position of the Eigen values in the complex plane decides the behavior of the algorithm, the quickness of an algorithm in terms of convergence can be judged (Figs. 1, 2). However the dynamical models are simple representations of the complex systems, so the actual parameter ranges may differ slightly depending on the application, but, these models assist in providing good starting ranges of values of parameters for

| Table 5 Parameters for IDE-LBG |
|--------------------------------|
| No. of runs                  | 15                |
| Upper bound of search space  | Maximum gray level present in the image |
| Lower bound of search space  | Minimum gray level present in the image |
| Number of particles (NP)     | 30                |
| Weighing factor (F)          | $3 \ast \text{randn} \ast p$, where $	ext{randn} \sim N(0,1)$ and $p =$ linearly decreasing parameter |
| Crossover rate (Cr)          | 0.9               |
| Scheme                       | DE/current to best/1 |

Fig. 1 The five testing images of size $512 \times 512$
the algorithms. For setting the values of parameters for DE algorithm, the mutation factor of 0.5 and Crossover probability of 0.9 are common choices as recommended by Storn and Price [44]. For Firefly Algorithm the values of the parameters are determined experimentally which are commonly used for majority of the problems with minor fine-tuning if required [48, 49]. Table 6 shows the average PSNR values obtained by the IDE-LBG, IPSO-LBG, DE-LBG, BA-LBG, FA-LBG and LBG algorithms for different codebook sizes $N_c=8, 16, 32, 64, 128, 256, 512$ and 1024. Horng and Jiang [18] simulated five different algorithms in C++6.0, with population size = 100 and number of iterations = 50. PSNR values obtained by different algorithms are also reported in the same paper, but since the platform (here Matlab) and the value of parameters, namely, the iteration number and the population size is different, there will be some dissimilarity in the final results obtained for PSNR values of FA-LBG in this paper with that obtained in [17]. The results from the Table 6 (plots are in Fig. 3) confirmed that the fitness (PSNR) of the five test images using the IDE-LBG algorithm is higher than the IPSO-LBG, DE-LBG, BA-LBG, and FA-LBG algorithm, yet the difference in PSNR values of IDE-LBG and IPSO-LBG is not very distinct for most of Codebook sizes for various images. Also, the differences in the PSNR values obtained by DE-LBG and IDE-LBG are noticeable. Clearly, the improvisation significantly led to the improvement in results. Five reconstructed images of Codebook size 128 and block size 16 are also displayed in Fig. 2. It is clearly visible that the reconstructed image quality obtained using the IDE-LBG algorithm has the edge over the IPSO-LBG, DE-LBG, BA-LBG and FA-LBG algorithms. Also it is noticeable that for all algorithms the PSNR values are better than the sole LBG algorithm. It is advisable to compare algorithms based on the number of fitness evaluations rather than on the number of generations because
Table 6  PSNR values of image compression for 5 different 512×512 images with 8 different codebook sizes

| Nc  | Method | Lena    | Baboon  | Goldhill | Pepper  | Barbara |
|-----|--------|---------|---------|----------|---------|---------|
| 8   | IDE-LBG| 25.83   | 20.89   | 26.97    | 25.80   | 25.33   |
|     | IPSO-LBG| 25.75   | 20.54   | 26.22    | 25.79   | 25.29   |
|     | DE-LBG  | 25.72   | 20.49   | 26.30    | 25.78   | 25.31   |
|     | BA-LBG  | 24.18   | 19.21   | 25.11    | 24.75   | 24.35   |
|     | FA-LBG  | 24.03   | 19.19   | 24.48    | 24.64   | 23.59   |
|     | LBG     | 24.01   | 18.27   | 24.35    | 24.21   | 23.53   |
| 16  | IDE-LBG | 27.19   | 21.11   | 27.33    | 26.55   | 26.14   |
|     | IPSO-LBG| 27.04   | 21.08   | 27.11    | 26.41   | 25.97   |
|     | DE-LBG  | 27.04   | 21.10   | 27.15    | 26.48   | 25.98   |
|     | BA-LBG  | 25.69   | 20.18   | 26.33    | 25.96   | 24.89   |
|     | FA-LBG  | 25.44   | 20.14   | 25.91    | 25.33   | 24.51   |
|     | LBG     | 25.41   | 19.44   | 25.79    | 25.11   | 23.69   |
| 32  | IDE-LBG | 28.51   | 21.29   | 28.88    | 27.62   | 27.59   |
|     | IPSO-LBG| 28.39   | 21.23   | 28.54    | 27.33   | 27.55   |
|     | DE-LBG  | 28.38   | 21.22   | 28.62    | 27.41   | 27.52   |
|     | BA-LBG  | 25.92   | 20.24   | 26.38    | 26.59   | 26.39   |
|     | FA-LBG  | 25.73   | 20.26   | 26.36    | 26.58   | 26.38   |
|     | LBG     | 25.43   | 19.62   | 25.81    | 25.29   | 23.77   |
| 64  | IDE-LBG | 29.39   | 22.84   | 28.66    | 29.25   | 28.21   |
|     | IPSO-LBG| 29.37   | 22.78   | 28.48    | 28.95   | 28.17   |
|     | DE-LBG  | 29.38   | 22.66   | 28.56    | 28.93   | 28.19   |
|     | BA-LBG  | 26.48   | 21.86   | 27.48    | 28.89   | 27.99   |
|     | FA-LBG  | 26.23   | 21.75   | 27.24    | 28.21   | 27.85   |
|     | LBG     | 25.45   | 19.64   | 25.82    | 25.31   | 23.93   |
| 128 | IDE-LBG | 30.46   | 24.11   | 29.93    | 30.84   | 29.99   |
|     | IPSO-LBG| 30.44   | 24.05   | 29.84    | 30.65   | 29.94   |
|     | DE-LBG  | 30.42   | 24.08   | 29.88    | 30.71   | 29.98   |
|     | BA-LBG  | 28.23   | 22.44   | 28.33    | 29.92   | 29.11   |
|     | FA-LBG  | 27.74   | 22.21   | 28.19    | 29.71   | 28.98   |
|     | LBG     | 25.52   | 19.71   | 25.88    | 25.38   | 23.86   |
| 256 | IDE-LBG | 31.47   | 24.79   | 30.64    | 31.66   | 30.51   |
|     | IPSO-LBG| 31.46   | 24.67   | 30.63    | 31.62   | 30.48   |
|     | DE-LBG  | 31.46   | 24.68   | 30.58    | 31.64   | 30.46   |
|     | BA-LBG  | 28.94   | 22.95   | 29.19    | 30.54   | 29.35   |
|     | FA-LBG  | 28.47   | 22.58   | 28.94    | 30.45   | 29.25   |
|     | LBG     | 25.58   | 19.83   | 25.93    | 25.42   | 24.32   |
| 512 | IDE-LBG | 32.51   | 25.66   | 31.53    | 32.42   | 31.48   |
|     | IPSO-LBG| 32.49   | 25.31   | 31.48    | 32.40   | 31.39   |
|     | DE-LBG  | 32.48   | 25.28   | 31.48    | 32.39   | 31.38   |
|     | BA-LBG  | 29.59   | 23.80   | 30.58    | 31.56   | 30.25   |
|     | FA-LBG  | 29.45   | 23.56   | 30.49    | 31.55   | 30.12   |
|     | LBG     | 25.62   | 19.95   | 25.98    | 25.57   | 24.28   |
| 1024| IDE-LBG | 33.25   | 26.09   | 32.29    | 33.69   | 32.89   |
Table 6 (continued)

| Nc  | Method   | Lena   | Baboon | Goldhill | Pepper | Barbara |
|-----|----------|--------|--------|----------|--------|---------|
|     |          |        |        |          |        |         |
|     | IPSO-LBG | 33.19  | 26.08  | 32.26    | 33.67  | 32.89   |
|     | DE-LBG   | 33.10  | 26.08  | 32.27    | 33.61  | 32.87   |
|     | BA-LBG   | 30.89  | 24.44  | 31.23    | 32.38  | 31.31   |
|     | FA-LBG   | 30.81  | 23.68  | 30.79    | 32.44  | 30.22   |
|     | LBG      | 25.68  | 20.02  | 26.01    | 25.68  | 24.46   |

Fig. 3 Average PSNR of six VQ methods for 5 test images of size 512 × 512 (Color figure online)

Table 7 Average number of fitness evaluations of the algorithms taken into consideration for population size of 30 and 25 number of generations

| Algorithm | Average no. of FE |
|-----------|------------------|
| FA-LBG    | 1200             |
| BA-LBG    | 1060             |
| IPSO-LBG  | 950              |
| DE-LBG    | 860              |
| IDE-LBG   | 855              |
Table 8 Average computation time (in seconds) of 5 different 512×512 images with 8 different codebook sizes using 6 different algorithms

| Nc | Method | IDE-LBG | IPSO-LBG | DE-LBG | BA-LBG | FA-LBG | LBG |
|----|--------|---------|----------|--------|--------|--------|-----|
| 8  | Lena   | 276.33766 | 313.03061 | 276.99594 | 337.23012 | 870.26505 | 3.36778 |
|    | Baboon | 279.71777 | 296.06271 | 285.85108 | 305.6591 | 713.64487 | 4.11691 |
|    | Goldhill | 286.8328 | 298.59222 | 295.76749 | 312.08367 | 678.24273 | 3.89279 |
|    | Pepper | 300.19214 | 317.95078 | 314.07197 | 319.48706 | 682.97637 | 3.5852 |
|    | Barbara | 271.91972 | 318.76814 | 315.80807 | 337.29771 | 654.19126 | 3.92124 |
|    | Average | 296.04251 | 323.74464 | 344.94695 | 375.04343 | 5.70866 |
| 16 | Lena   | 315.3594 | 325.91827 | 323.74464 | 344.94695 | 757.04343 | 5.01866 |
|    | Baboon | 303.03685 | 322.67492 | 319.13898 | 335.38259 | 933.55194 | 5.47301 |
|    | Goldhill | 333.71474 | 339.49553 | 334.12514 | 399.64173 | 923.06468 | 5.14901 |
|    | Pepper | 309.74158 | 336.62164 | 317.19666 | 340.12584 | 625.96152 | 5.30032 |
|    | Barbara | 306.00725 | 312.57896 | 309.67945 | 312.08367 | 716.12849 | 4.95383 |
|    | Average | 319.92534 | 331.39938 | 325.79814 | 337.29771 | 798.91776 | 5.17898 |
| 32 | Lena   | 359.28686 | 369.4495 | 366.57749 | 375.5422 | 884.72189 | 6.37241 |
|    | Baboon | 363.83859 | 376.16598 | 364.06143 | 450.77278 | 793.25865 | 6.1981 |
|    | Goldhill | 369.17656 | 408.78158 | 383.7649 | 399.64173 | 888.95883 | 6.48971 |
|    | Pepper | 351.17988 | 367.89398 | 351.24514 | 393.38944 | 673.87486 | 6.33959 |
|    | Barbara | 354.31385 | 374.39712 | 354.61843 | 383.25114 | 803.6703 | 6.95169 |
|    | Average | 371.82397 | 377.67385 | 371.85952 | 381.0423 | 807.38227 | 6.47038 |
| 64 | Lena   | 330.85992 | 346.83183 | 346.64903 | 374.09504 | 785.76614 | 8.64055 |
|    | Baboon | 354.38469 | 360.45818 | 356.00296 | 378.08556 | 860.86505 | 9.10834 |
|    | Goldhill | 368.51957 | 381.52522 | 373.50364 | 390.11364 | 972.48237 | 9.16293 |
|    | Pepper | 347.0825 | 377.4938 | 349.04201 | 393.25369 | 653.68308 | 8.7445 |
|    | Barbara | 337.56835 | 398.87776 | 386.33821 | 407.37226 | 886.62223 | 11.2522 |
|    | Average | 362.9405 | 369.76224 | 364.62166 | 373.5029 | 832.85233 | 9.38172 |
| 128 | Lena | 687.59887 | 789.66443 | 720.43765 | 830.77755 | 997.50768 | 18.4820 |
|    | Baboon | 646.89551 | 705.45175 | 665.33123 | 842.57484 | 972.81035 | 21.01316 |
|    | Goldhill | 608.78751 | 656.58192 | 619.61864 | 674.41401 | 1084.2336 | 15.98994 |
|    | Pepper | 569.98191 | 592.24318 | 571.95434 | 611.39697 | 1047.0451 | 13.6080 |
|    | Barbara | 638.18516 | 760.3404 | 648.36463 | 798.04335 | 1086.8423 | 26.34064 |
|    | Average | 650.78394 | 683.46334 | 666.90392 | 727.72511 | 1038.3794 | 19.08673 |
| 256 | Lena | 656.62966 | 697.96462 | 671.39743 | 828.62454 | 831.93281 | 23.9346 |
|    | Baboon | 777.14936 | 799.13787 | 781.59727 | 828.54203 | 1022.7756 | 26.71431 |
|    | Goldhill | 718.2245 | 750.94635 | 750.23458 | 863.95371 | 876.39546 | 27.41266 |
|    | Pepper | 745.00019 | 790.3884 | 746.12201 | 795.04054 | 806.87203 | 25.20058 |
|    | Barbara | 690.99872 | 832.58436 | 794.9447 | 895.59867 | 968.19969 | 23.77171 |
|    | Average | 756.58256 | 772.23642 | 761.5089 | 798.33111 | 895.22945 | 25.34787 |
| 512 | Lena | 1268.0183 | 1518.1913 | 1321.934 | 1738.3446 | 1982.5007 | 79.8515 |
|    | Baboon | 1474.6925 | 1774.4546 | 1503.5888 | 2210.7848 | 2329.5884 | 85.45867 |
|    | Goldhill | 1772.7643 | 1783.318 | 1779.635 | 1981.4339 | 2236.6837 | 76.21621 |
|    | Pepper | 1259.8191 | 1451.8307 | 1418.3186 | 1705.8437 | 1762.8646 | 71.8232 |
|    | Barbara | 1243.8908 | 1305.9474 | 1294.9085 | 1372.3504 | 1520.8729 | 109.3033 |
extra fitness evaluations are consumed in local search or due to generation of off-springs [50]. Table 7 shows the average number of fitness evaluations of the various algorithms used in the study averaged over all the runs and for all the images and considering a population size of 30. For each image considered for the study and for each codebook the only thing that changes is the size of the codebook. Though the number of fitness evaluations may remain more or less the same the computation time increases with the increase in the codebook sizes. Also it is clear from the average values of fitness evaluations that out of all the algorithms on an average basis IDE-LBG has the least value followed by IPSO-LBG, BA-LBG and FA-LBG. Between IDE-LBG and DE-LBG the difference in the number of fitness evaluation values is very less. Also all these results are to certain extent supported by the average computation values of the algorithms. We have considered the population size to be 30 and the number of fitness evaluations per run to be 1500 which is used as the stopping criteria for the algorithms. Table 8 shows the average computation times of different algorithms with different bitrates. From the observations of Table 8 we infer that though the LBG algorithm has the least computational time, it has the least PSNR and thus yields a bad reconstructed image. The average computation time of IDE-LBG Algorithm is around 1.11 times faster than the BA-LBG Algorithm and 1.98 times faster than the FA-LBG Algorithm. Also, it is seen than the proposed IDE-LBG Algorithm is 1.03 times faster than the DE-LBG Algorithm, so, more or less the computation speed is the same, though improvements can be noticed in the PSNR values. As the convergence speed of IDE-LBG Algorithm is fast, it can be suitably applied to design a fast codebook for vector quantization.

The fitness values of all the methods are statistically compared using a non-parametric statistical Wilcoxon’s rank sum test [51] which is conducted considering a 5% significance level to determine the difference of the results from the best performing method with that obtained from the other compared methods in a statistically significant way. A null hypothesis is accepted when the p value between two methods is more than the significance level of 5% which assumes that there is no difference between the values of the two methods tested. On the other hand if the p value between two methods is less than the significance level of 5% then the null hypothesis is rejected. Although it is to be noted that the p values obtained between IDE-LBG and DE-LBG are higher than those obtained by the remaining pairs, all the p values reported in the Table 9 are less than the significance level of 5%, thus it

| Nc | Method | IDE-LBG | IPSO-LBG | DE-LBG | BA-LBG | FA-LBG | LBG |
|----|--------|---------|----------|--------|--------|--------|-----|
| 1024 | Lena   | 1471.1444 | 1565.4357 | 1518.1678 | 1800.9723 | 1846.4633 | 84.53063 |
|     | Baboon | 1696.8561 | 1901.1487 | 1775.835 | 3295.6582 | 4024.9241 | 157.38647 |
|     | Goldhill | 2428.4587 | 2616.2299 | 2525.5145 | 3150.6187 | 4019.8717 | 159.51384 |
|     | Pepper | 1729.6512 | 2353.9365 | 2116.793 | 2409.1516 | 2647.4327 | 162.78376 |
|     | Barbara | 1813.5291 | 2285.6211 | 2267.8715 | 2801.304 | 2838.1847 | 164.98141 |
|     | Average | 2234.624 | 2419.207 | 2359.0802 | 2478.6901 | 3005.4307 | 170.11031 |
Table 9 *p* values from the Wilcoxon’s rank test of the compared algorithms taken into consideration for the study

| Nc  | Method | IPSO-LBG versus IDE-LBG | DE-LBG versus IDE-LBG | BA-LBG versus IDE-LBG | FA-LBG versus IDE-LBG |
|-----|--------|-------------------------|-----------------------|-----------------------|-----------------------|
| 8   | Lena   | 0.0001993               | 0.0020574             | 6.443E−05             | 3.355E−06             |
|     | Baboon | 0.0006884               | 0.0077422             | 5.874E−05             | 7.426E−06             |
|     | Goldhill | 0.0003113           | 0.0076146             | 9.709E−05             | 8.683E−06             |
|     | Pepper | 0.0001776               | 0.0042979             | 3.755E−05             | 6.882E−06             |
|     | Barbara | 0.0006381             | 0.0081043             | 3.74E−05              | 2.14E−06              |
|     | Average | 0.000178              | 0.0079474             | 2.136E−05             | 3.944E−06             |
| 16  | Lena   | 0.0005966               | 0.0030606             | 6.455E−05             | 4.896E−06             |
|     | Baboon | 0.0002367               | 0.0080374             | 1.096E−05             | 3.011E−06             |
|     | Goldhill | 0.0003136            | 0.0057779             | 1.006E−05             | 4.113E−06             |
|     | Pepper | 0.0001944               | 0.0020106             | 7.866E−05             | 2.987E−06             |
|     | Barbara | 0.0006432             | 0.0096798             | 4.382E−05             | 6.978E−06             |
|     | Average | 0.0007823             | 0.0048938             | 6.589E−05             | 1.187E−06             |
| 32  | Lena   | 0.0009404               | 0.0026871             | 2.735E−05             | 7.999E−06             |
|     | Baboon | 0.0005388               | 0.0079206             | 4.02E−05              | 2.802E−06             |
|     | Goldhill | 0.0001335           | 0.0070597             | 4.353E−05             | 4.572E−06             |
|     | Pepper | 0.0006489               | 0.0015346             | 3.227E−05             | 7.75E−06              |
|     | Barbara | 0.0007268             | 0.002128              | 1.388E−05             | 1.014E−06             |
|     | Average | 0.000107              | 0.004808              | 6.59E−05              | 7.257E−06             |
| 64  | Lena   | 0.0005781               | 0.0019794             | 6.354E−05             | 1.352E−06             |
|     | Baboon | 0.0002209               | 0.0018873             | 1.506E−05             | 1.766E−06             |
|     | Goldhill | 0.0002766            | 0.0038573             | 3.233E−05             | 2.254E−06             |
|     | Pepper | 0.0003259               | 0.0090363             | 7.062E−05             | 5.602E−06             |
|     | Barbara | 0.000266              | 0.0029083             | 8.657E−06             | 9.147E−06             |
|     | Average | 0.000736              | 0.0060201             | 3.203E−05             | 1.745E−06             |
| 128 | Lena   | 0.0006602               | 0.0098914             | 1.787E−05             | 2.652E−06             |
|     | Baboon | 0.0004571               | 0.001666              | 6.873E−05             | 4.084E−06             |
|     | Goldhill | 0.0009846           | 0.0046197             | 6.245E−05             | 1.628E−06             |
|     | Pepper | 0.0004432               | 0.0024502             | 7.605E−05             | 8.724E−06             |
|     | Barbara | 0.0004157            | 0.0071698             | 3.012E−05             | 5.353E−06             |
|     | Average | 0.0008492             | 0.0063774             | 3.42E−05              | 3.062E−06             |
| 256 | Lena   | 0.0005073               | 0.0048038             | 3.66E−05              | 5.627E−06             |
|     | Baboon | 0.0007683               | 0.004819              | 4.351E−05             | 1.336E−06             |
|     | Goldhill | 0.000122            | 0.0036117             | 3.243E−05             | 6.572E−06             |
|     | Pepper | 0.0009612               | 0.0094216             | 4.633E−05             | 2.481E−06             |
|     | Barbara | 0.0007875            | 0.0078339             | 7.432E−05             | 7.463E−06             |
|     | Average | 0.0001953             | 0.007134              | 4.686E−05             | 2.2E−06               |
| 512 | Lena   | 0.0001887               | 0.0084122             | 1.833E−05             | 1.719E−06             |
|     | Baboon | 0.0006994               | 0.0090495             | 5.214E−05             | 7.057E−06             |
|     | Goldhill | 0.0002382           | 0.0095811             | 5.455E−05             | 6.829E−06             |
|     | Pepper | 0.0001329               | 0.0082828             | 7.511E−05             | 1.29E−06              |
is an evidence against the null hypothesis indicating that the IDE-LBG fitness values are statistically significant and have not occurred by chance (Fig. 4).

Table 10 shows the PSNR values for image compression of two different color images (768×512) obtained from the Kodak Color Image Database (http://r0k.us/graphics/kodak/). In Table 10 the PSNR values obtained using the proposed IDE-LBG Algorithm is compared with that obtained using a standard JPEG2000 compressor. Figure 5 shows the reconstructed color images. It is revealed that the
proposed algorithm yields significantly better results than what achieved using standard JPEG2000 compressor. So to obtain a significantly better PSNR value the proposed heuristic approach can be used. Although it is known that heuristic approaches consumes more time as compared to direct methods a compromise can be made for obtaining better results in terms of metrics like PSNR or MSE values.

| bpp  | Method   | Kodim-03 | Kodim-23 |
|------|----------|----------|----------|
| 0.25 | DE-LBG   |          |          |
|      | JPEG-2000|          |          |
| 0.5  | DE-LBG   |          |          |
|      | JPEG-2000|          |          |

Fig. 5 The reconstructed color images (Color figure online)
5 Conclusion

In this study, an Improved Differential Evolution (IDE) Algorithm based codebook training has been presented for image compression. The peak signal to noise ratio (PSNR) of vector quantization is optimized by using IDE-LBG algorithm where PSNR is considered as the fitness function for the optimization problem. The algorithm parameters have been tuned suitably for efficient codebook design. The proposed IDE-LBG algorithm is tested on five standard 512×512 grayscale images. Results reveal the potency of the proposed algorithm when compared to IPSO-LBG, DE-LBG, BA-LBG and FA-LBG algorithms. It is observed that using the proposed IDE-LBG algorithm the PSNR values and the quality of reconstructed image obtained are much better than that obtained from the other algorithms in comparison for eight different codebook sizes. Also the proposed algorithm is compared with standard JPEG2000 compressor using two color images from the Kodak database and the PSNR values obtained clearly suggests that the proposed algorithm outperforms the JPEG2000 compressor though consuming more time than the JPEG2000 compressor. So, a trade-off can be made between the two depending on the situation and requirement. Compared to BA-LBG and DE-LBG the convergence speed is more or less the same. So, significant improvement is required to reduce computation time using further modifications in future. The future works include formulation of hybrid algorithms to enhance the performance of the algorithms so that better results are obtained in a significantly lesser time. Using different fitness measures like SSIM instead of PSNR can also lead to the betterment of results.

Acknowledgements The author wishes to acknowledge Jadavpur University for providing the license of the Matlab version used in this work.

References

1. R.M. Gray, Vector quantization. IEEE Signal Process. Mag. 1(2), 4–29 (1984)
2. D. Ailing, C. Guo, An adaptive vector quantization approach for image segmentation based on SOM network. Neurocomputing 149, 48–58 (2015)
3. H.B. Kekre, Speaker recognition using vector quantization by MFCC and KMCG clustering algorithm, in IEEE International Conferences on Communication, Information & Computing Technology (ICCICT), IEEE, Mumbai, 2012, pp. 1–5
4. C.W. Tsai, C.Y. Lee, M.C. Chiang, C.S. Yang, A fast VQ codebook generation algorithm via pattern reduction. Pattern Recognit. Lett. 30, 653–660 (2009)
5. S.K. Frank, R.E. Aaron, J. Hwang, R.R. Lawrence, Report: a vector quantization approach to speaker recognition. AT&T Tech. J. 66(2), 14–16 (2014)
6. C.H. Chan, M.A. Tahir, J. Kittler, M. Pietikäinen, Multiscale local phase quantization for robust component-based face recognition using kernel fusion of multiple descriptors. IEEE Trans. Pattern Anal. Mach. Intell. 35(5), 1164–1177 (2013)
7. G.E. Tsekouras, D. Darzentas, I. Drakoulaki, A.D. Niros, Fast fuzzy vector quantization, in IEEE International Conference on Fuzzy Systems (FUZZ), IEEE, Barcelona, 2010, pp. 1–8
8. Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantizer design. IEEE Trans. Commun. 28(1), 702–710 (1980)
9. G.E. Tsekouras, D.M. Tsolakis, Fuzzy clustering-based vector quantization for image compression, in Computational Intelligence in Image Processing, ed. by A. Chatterjee, P. Siarry (Springer, Berlin, 2012), pp. 93–105
10. G. Patane, M. Russo, The enhanced LBG algorithm. Neural Netw. 14, 1219–1237 (2002)
11. A. Rajpoot, A. Hussain, K. Saleem, Q. Qureshi, A novel image coding algorithm using ant colony system vector quantization, in International Workshop on Systems, Signals and Image Processing, Poznan, Poland, 2004, pp. 13–15
12. C.W. Tsai, S.P. Tsengh, C.S. Yangc, M.C. Chiangb, PREACO: a fast ant colony optimization for codebook generation. Appl. Soft Comput. 13, 3008–3020 (2013)
13. M. Kumar, R. Kapoor, T. Goel, Vector quantization based on self-adaptive particle swarm optimization. Int. J. Nonlinear Sci. 9(3), 311–319 (2010)
14. H.M. Feng, C.Y. Chen, F. Ye, Evolutionary fuzzy particle swarm optimization vector quantization learning scheme in image compression. Expert Syst. Appl. 32, 213–222 (2007)
15. Y. Wang, X.Y. Feng, X.Y. Huang, D.B. Pu, W.G. Zhou, Y.C. Liang et al., A novel quantum swarm evolutionary algorithm and its applications. Neurocomputing 70, 633–640 (2007)
16. S. Yang, R. Wu, M. Wang, L. Jiao, Evolutionary clustering based vector quantization and SPIHT coding for image compression. Pattern Recognit. Lett. 31, 1773–1780 (2010)
17. M.H. Horng, Vector quantization using the firefly algorithm for image compression. Expert Syst. Appl. 39(1), 1078–1091 (2012)
18. M.H. Horng, T.W. Jiang, Image vector quantization algorithm via honey bee mating optimization. Expert Syst. Appl. 38(3), 1382–1392 (2011)
19. P.K. Tripathy, R.K. Dash, C.R. Tripathy, A dynamic programming approach for layout optimization of interconnection networks. Eng. Sci. Technol. 18, 374–384 (2015)
20. D. Tsolakis, G.E. Tsekouras, A.D. Niros, A. Rigos, On the systematic development of fast fuzzy vector quantization for gray scale image compression. Neural Netw. 36, 83–96 (2012)
21. G.E. Tsekouras, A fuzzy vector quantization approach to image compression. Appl. Math. Comput. 167(1), 539–5605 (2005)
22. C. Ping-Yi, J.T. Tsai, J.H. Chou, W.H. Ho, H.Y. Shi, S.H. Chen, Improved PSO-LBG to design VQ codebook, in 2013 Proceedings of InSICE Annual Conference (SICE), 2013 Sep 14, IEEE, pp. 876–879
23. S.M. Hosseini, A. Naghsh-Nilchi, Medical ultrasound image compression using contextual vector quantization. Comput. Biol. Med. 42, 743–750 (2012)
24. B. Huanga, Y. Wanga, J. Chen, ECG compression using the context modeling arithmetic coding with dynamic learning vector–scalar quantization. Biomed. Signal Process. Control 8, 59–65 (2013)
25. C. Karri, U. Jena, Fast vector quantization using a Bat algorithm for image compression. Eng. Sci. Technol. Int. J. 19(2), 769–781 (2016). ISSN 2215-0986. https://doi.org/10.1016/j.jestch.2015.11.003
26. N. Sanyal, A. Chatterjee, S. Munshi, Modified bacterial foraging optimization technique for vector quantization-based image compression, in Computational Intelligence in Image Processing, 2013. Springer, Berlin, pp. 131–152
27. K. Chiranjeevi, U.R. Jena, Image compression based on vector quantization using cuckoo search optimization technique. Ain Shams Eng. J. 9(4), 1417–1431 (2018)
28. S.-J. Wu, P.T. Chow, Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. Eng. Optim. 24(2), 137–159 (1995)
29. R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in Proceedings of the 6th International Symposium on Micro Machine and Human Science (MHS’95), IEEE, Nagoya, Japan, October 1995, pp. 39–43
30. E. Rashedi, H. Nezamabadi-pour, S. Saryazdi, GSA: a gravitational search algorithm. Inf. Sci. 179(13), 2232–2248 (2009)
31. M. Dorigo, G.D. Caro, Ant algorithms for discrete optimization. Artif. Life 5(3), 137–172 (1999)
32. M. Dorigo, L.M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evolut. Comput. 1(1), 53–66 (1997)
33. C. Zhang, H.-P. Wang, Mixed-discrete nonlinear optimization with simulated annealing. Eng. Optim. 21(4), 277–291 (1993)
34. E.H.L. Aarts, J.H.M. Korst, P.J.M. van Laarhoven, Simulated annealing, in Local Search in Combinatorial Optimization, 1997, pp. 91–120
35. Nag S. Adaptive Plant Propagation Algorithm for Solving Economic Load Dispatch Problem. arXiv preprint arXiv:1708.07040. 4 Aug 2017
36. Nag S. A Type II Fuzzy Entropy Based Multi-level Image Thresholding Using Adaptive Plant Propagation Algorithm. arXiv preprint arXiv:1708.09461. 23 Aug 2017
37. J.C. Bansal, H. Sharma, S.S. Jadon, Artificial bee colony algorithm: a survey. Int. J. Adv. Intell. Paradigm. 5(1), 123–159 (2013)
38. X.S. Yang, Firefly algorithms for multimodal optimization, LNCS, vol. 5792 (Springer, Heidelberg, 2009), pp. 169–178
39. D.P. Rini, S.M. Shamsuddin, S.S. Yuhaniz, Particle swarm optimization: technique, system and challenges. Int. J. Comput. Appl. 14(1), 19–27 (2011)
40. G.T. Chandra Sekhar, R.K. Sahu, A.K. Baliarsingh, S. Panda, Load frequency control of power system under deregulated environment using optimal firefly algorithm. Int. J. Electr. Power Energy Syst. 74, 195–211 (2016)
41. X.S. Yang, A new metaheuristic bat-inspired algorithm, in Nature Inspired Cooperative Strategies for Optimization (NISCO 2010), vol. 284, Studies in Computational Intelligence, Springer, Berlin, 2010, pp. 65–74
42. Q. Chen, J.G. Yang, J. Gou, Image compression method using improved PSO vector quantization, in First International Conference on Neural Computation (ICNC 2005), vol. 3612, Lecture Notes on Computer Science, 2005, pp. 490–495
43. R. Storn, K. Price, Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. 11, 341–359 (1997)
44. Rainer Storn, Kenneth Price, Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. 11(4), 341–359 (1997)
45. D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, K.V. Price, New ideas in optimization (McGraw-Hill Ltd, London, 1999)
46. Ioan Cristian Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection. Inf. Process. Lett. 85(6), 317–325 (2003)
47. S. Chen et al., Global convergence analysis of the bat algorithm using a markovian framework and dynamical system theory. Expert Syst. Appl. 114, 173–182 (2018)
48. X. He et al., Global convergence analysis of the flower pollination algorithm: a discrete-time Markov chain approach. Proc. Comput. Sci. 108, 1354–1363 (2017)
49. X.-S. Yang, Nature-inspired metaheuristic algorithms (Luniver Press, New York, 2010)
50. M. Črepinšek, S.H. Liu, M. Mernik, Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them. Appl. Soft Comput. 19, 161–170 (2014)
51. S. García et al., A study on the use of non-parametric tests for analyzing the evolutionary algorithms’ behaviour: a case study on the CEC’2005 special session on real parameter optimization. J. Heuristics 15(6), 617 (2009)

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.