JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 81-102 (2018)
DOI: 10.6688/JISE.2018.34.1.6

High-Quality Codebook Generation of Vector Quantization Using the HT-ABC-LBG Algorithm

SHU-CHIEN HUANG
Department of Computer Science
National Pingtung University
Pingtung City, 900 Taiwan
E-mail: schuang@mail.nptu.edu.tw

This paper proposes a vector quantization (VQ) codebook generation method for image data compression using a combined scheme of Hotelling transform (HT), the artificial bee colony (ABC) algorithm and the Linde-Buzo-Gray (LBG) algorithm. First, the grayscale image is divided into a set of non-overlapping image blocks. Each block is represented by an input vector, and these input vectors are then sorted by Hotelling transform. Second, the ABC algorithm is employed to select some of the sorted vectors to form an initial codebook. Third, this codebook serves as the input of the LBG algorithm to compute vector quantization codebook. The experimental results show that the proposed HT-ABC-LBG algorithm outperforms the FF-LBG algorithm in terms of the quality of the decompressed image and the computation time.

Keywords: vector quantization, image compression, codebook generation, Hotelling transform, artificial bee colony algorithm

1. INTRODUCTION

Image compression addresses the problem of reducing the amount of data required to represent a digital image. There are two types of image compression techniques, namely, lossless and lossy image compression. Lossless compression removes as much redundancy from the source image as possible, and guarantees that the original information will be perfectly recovered from the compressed data. For lossy compression, the reconstructed image contains distortion, which may or may not be visually apparent. Unlike lossless technologies, a relatively high compression ratio can be achieved. Among the various kinds of lossy compression methods, vector quantization (VQ) is one of the most popular compression techniques.

Typically, VQ [1-10] can be divided into three parts: codebook generation, image encoding, and image decoding. The VQ process is depicted in Fig. 1. The input vectors are obtained from image data by extracting non-overlapping blocks of size 4×4 (or 8×8). The goal of the codebook generation procedure is to generate a set of representative codewords to form the codebook. In the image encoding procedure, each image block can be viewed as an input vector. To compress the image, each input vector is compared with the codewords in the codebook to find its closest codeword. Then the index is obtained to encode the input vector. In the image decoding procedure, the corresponding codeword of each VQ index is employed to recover the compressed block. When each compressed block is sequentially reconstructed, the decoded image of VQ is obtained.

Received March 16, 2016; revised June 7, 2016; accepted July 27, 2016.
Communicated by Chung-Lin Huang.

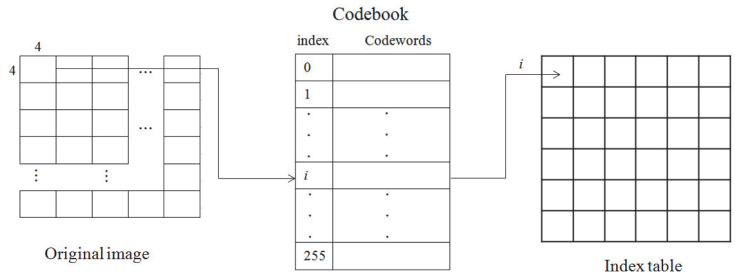


Fig. 1. Vector quantization.

The LBG algorithm [1] is the most commonly used algorithm for codebook design. It is a clustering-based algorithm. However, the LBG technique is extremely sensitive to the initial codebook. Lai *et al.* [2] developed a method to speed up the generation of codebooks through using the codebook displacement between two successive partition processes. Their method can obtain the same distortion as that produced by the LBG algorithm, compared to which it may reduce the computing time by a factor of 35.9-121.2. Tsai *et al.* [3] also presented a method to enhance the performance of LBG and the LBG-based algorithm. Their method is motivated by the observation that input vectors that are static during the training process can be considered as part of the final solutions and thus can be compressed and removed to eliminate the redundant computations in the later iterations. Horng *et al.* [4] applied a swarm algorithm, honey bee mating optimization (HBMO), to construct the codebook of vector quantization. The experimental results showed that their algorithm can increase the quality of the reconstructed images with respect to three other methods, LBG, PSO-LBG, and QPSO-LBG. Horng [5] proposed a method based on the firefly (FF) algorithm to construct the vector quantization codebook. The codebook obtained by the LBG algorithm is assigned to one of the initial solutions, and the other initial solutions are randomly generated. The experimental results showed that the FF-LBG algorithm is as reliable as the HMBO-LBG algorithm; however, FF-LBG needs much less computation time than HBMO-LBG. Karri *et al.* [6] proposed a bat algorithm based vector quantization (BA-LBG) for image compression. The peak signal to noise ratio of vector quantization is optimized by employing the BA technique. The experimental results show that BA-LBG has higher PSNR compared to PSO-LBG, HBMO-LBG and FF-LBG.

In recent years, researchers have attempted to apply nature-inspired methods to the problem of image processing and pattern recognition. In this research, it starts by sorting the input vectors using Hotelling transform, and then makes full use of the searching ability of the artificial bee colony algorithm to generate the initial codebook. Finally, the initial codebook serves as the input of the LBG algorithm. The experimental results show that the proposed algorithm can output better codebooks in a relatively short time. The remainder of this paper is organized as follows. Section 2 presents the Hotelling transform and artificial bee colony algorithm. In Section 3, the new codebook design algorithm is described. In Section 4, the experimental studies for the comparison of the proposed algorithm with the FF-LBG algorithm [5] and BA-LBG algorithm [6] are shown. Finally, conclusions are given in Section 5.

2. HOTELLING TRANSFORM AND THE ABC ALGORITHM

This section describes the fundamental concept of the Hotelling transform [11, 12] and artificial bee colony algorithm [13-22], which are the basis of the proposed algorithm. Table 1 describes the different symbols and their corresponding meanings.

Table 1. Symbols and notations.

Symbols	Descriptions
x_k	a 16-dimensional column vector
M	the number of image blocks
m_x	the mean vector
C_x	the covariance matrix
A	the transformation matrix
y_k	$y_k=A(x_k - m_x)$
$y_k^{(1)}$	the first element of vector y_k
SN	the number of food sources
D	the number of optimization parameters (the codebook size minus one)
s_i	the ith solution
v_i	a new solution produced by using Eq. (5)
$fit(s_i)$	the fitness value of the solution s_i
$obj(s_i)$	the objective function value of the solution s_i
$trial_i$	$trial_i$ holding trial number through which solution s_i cannot be improved
$limit$	a food source which could not be improved with "$limit$" trials is abandoned by its employed bee
MCN	the maximum cycle number for the ABC algorithm
X	the set $X=\{x_0, x_1, x_2, x_3,\ldots, x_{M-1}\}=\{sort\_x_0, sort\_x_1, sort\_x_2,\ldots, sort\_x_{M-1}\}$ contains M vectors
CB_i	the solution s_i corresponds to a codebook CB_i
C	the codebook C serves as the initial codebook for the LBG algorithm
LBG_iter	the maximum iteration number for the LBG algorithm
$rint(i)$	a random integer number

2.1 Hotelling Transform (HT)

In this research, each grayscale image is divided into a set of non-overlapping image blocks of 4×4 pixels. Each block is represented by a 16-dimensional column vector. These vectors are denoted by x_k, $k = 0, 1, 2, \ldots, M–1$. For M vectors, the mean vector m_x and covariance matrix C_x can be calculated by

$$m_x = \frac{1}{M}\sum_{k=0}^{M-1} x_k \tag{1}$$

and

$$C_x = \frac{1}{M}\sum_{k=0}^{M-1} x_k (x_k)^T - m_x (m_x)^T \tag{2}$$

The size of C_x is 16×16. Because C_x is real and symmetric, finding a set of 16 or-

thonormal eigenvectors is always possible. Let e_i and λ_i, $i = 1, 2, \ldots, 16$, be the eigenvectors and corresponding eigenvalues of C_x, arranged in descending order so that $\lambda_j \geq \lambda_{j+1}$ for $j = 1, 2, \ldots, 15$. Let A be a matrix of which the rows are formed from the eigenvectors of C_x, ordered so that the first row of A, denoted by $A^{(1)}$, is the eigenvector corresponding to the largest eigenvalue, and the last row, denoted by $A^{(16)}$, is the eigenvector corresponding to the smallest eigenvalue. That is, $A^{(i)} = (e_i)^T$, $i = 1, 2, \ldots, 16$. The size of A is 16×16.

A is a transformation matrix that maps the $x_k (k = 0, 1, 2, \ldots, M-1)$ into vectors denoted by $y_k (k = 0, 1, 2, \ldots, M-1)$, as follows:

$$y_k = A(x_k - m_x). \tag{3}$$

Eq. (3) is called the Hotelling transform, which is also known as the principal components transform. This transformation is defined in such a way that the first principal component has the largest possible variance. According to Eq. (3), the first element of vector y_k, denoted by $y_k^{(1)}$, can be computed as follows

$$y_k^{(1)} = A^{(1)} (x_k - m_x). \tag{4}$$

The variance of the first elements of the $y_k (k = 0, 1, 2, \ldots, M-1)$ vectors is the eigenvalue λ_1, the variance of the second elements of the $y_k (k = 0, 1, 2, \ldots, M-1)$ vectors is the eigenvalue λ_2, and so on.

2.2 The Artificial Bee Colony (ABC) Algorithm

The artificial bee colony (ABC) algorithm was proposed by Karaboga [13-17]. The colony of artificial bees contains three groups of bees: employed bees, onlooker bees and scout bees. In the ABC algorithm, the first half of the colony consists of employed bees while the second half is constituted of the onlooker bees. Each food source is associated with an employed bee, and each employed bee tries to detect a new food source in the neighborhood of its current food source. An onlooker bee evaluates the information gained from the employed bee and tries to find a new food source in the neighborhood of the selected food. An employed bee whose food source is exhausted by the employed and onlooker bees becomes a scout. The scout bee carries out random searches to discover new sources. The number of food sources, denoted by SN, is equal to the number of employed bees. The position of a food source represents a possible solution to the optimization problem, while the nectar amount of a food source corresponds to the quality (fitness) of the associated solution. Each solution $s_i (i = 0, 1, 2, \ldots, SN-1)$ is a D-dimensional vector. Here, D is the number of optimization parameters.

In the employed bee phase, the ABC algorithm generates a new solution v_i in the neighborhood of its present solution s_i as follows:

$$v_{i,j} = s_{i,j} + \phi_{ij}(s_{i,j} - s_{k,j}), \tag{5}$$

where k is a randomly produced index which is different from i, j is a random dimension index selected from the set $\{0, 1, 2, \ldots, D-1\}$, and ϕ_{ij} is a random number in the range $[-1, 1]$.

In the onlooker bee phase, an artificial onlooker bee selects a food source depending on the probability value associated with that food source, $prob_i$, calculated by the following expression:

$$prob_i = 0.9 \times \frac{fit(s_i)}{\max\{fit(s_0), fit(s_1), ..., fit(s_{SN-1})\}} + 0.1, \quad (6)$$

where $fit(s_i)$ is the fitness value of the solution s_i. After the selection, the onlooker bees try to improve the solutions by using Eq. (5). In the scout bee phase, if a solution s_i is abandoned, then the scout bee discovers a new food source to replace s_i.

The flowchart of the ABC algorithm is given in Fig. 2. The main steps of the ABC algorithm are shown as follows:

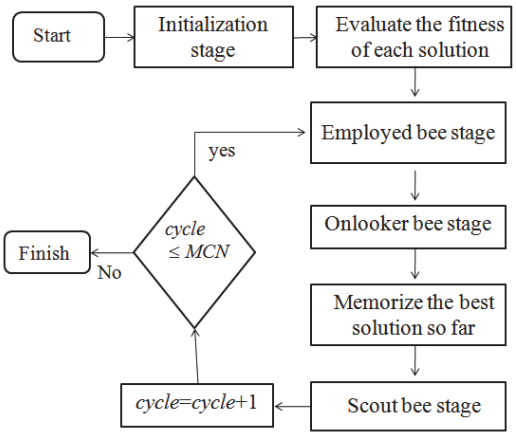


Fig. 2. Flow chart of the ABC algorithm.

1: //Initialization phase
   Generate the initial population s_i, $i = 0, 1, 2, \ldots, SN-1$
   $trial_i = 0$, $i = 0, 1, 2, \ldots, SN-1$
   set *cycle* to 1
2: Evaluate the fitness $fit(s_i)$ of the population
Repeat
3: // Employed bee phase
   For each employed bee
      Produce new solution (food source position) v_i by using Eq. (5)
      Calculate the fitness value $fit(v_i)$
      If $fit(v_i) > fit(s_i)$ then s_i is replaced by v_i and $trial_i$ is set to 0 else $trial_i = trial_i + 1$
   End
4: Calculate the probability values $prob_i$ by using Eq. (6) for the solutions $s_i (i = 0, 1, 2, \ldots, SN-1)$
5: // Onlooker bee phase
   For each onlooker bee
      Select a solution s_i depending on $prob_i$

Produce new solution (food source position) v_i by using Eq. (5)
Calculate the fitness value $fit(v_i)$
If $fit(v_i) > fit(s_i)$ then s_i is replaced by v_i and $trial_i$ is set to 0 else $trial_i=trial_i+1$
End
6: Memorize the best solution so far
7: // Scout bee phase
If there is an abandoned solution ($trial_i > limit$) for the scout then replace it with a new solution which will be randomly produced
$cycle = cycle+1$
Until $cycle \leq MCN$

3. THE NEW CODEBOOK DESIGN ALGORITHM

This method consists of three stages. The first stage is to sort the input vectors by Hotelling transform, the second is to generate the initial codebook by ABC algorithm, and the third is to obtain the codebook of vector quantization by LBG algorithm. The system flowchart is shown in Fig. 3.

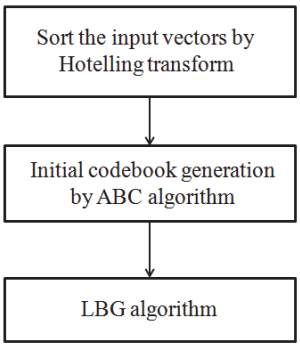


Fig. 3. The system flowchart.

3.1 Sort the Input Vectors by Hotelling Transform

In this research, the input vectors are obtained from image data by extracting non-overlapping blocks of size 4×4. The ith block can be viewed as a 16-dimensional column vector x_i. The set $X=\{x_0, x_1, x_2, x_3, \ldots, x_{M-1}\}$ contains M vectors. The values $y_k^{(1)}=A^{(1)}(x_k - m_x)$ for $x_k(k = 0, 1, 2, \ldots, M-1)$ are computed by using Eq. (4). Then, all the vectors are sorted and the set $X=\{sort\_x_0, sort\_x_1, sort\_x_2, \ldots, sort\_x_{M-1}\}$ is obtained according to the criterion $A^{(1)}(sort\_x_i - m_x) \leq A^{(1)}(sort\_x_{i+1} - m_x)$, $i = 0, 1, 2, \ldots, M-2$.

3.2 Generate the Initial Codebook by ABC Algorithm

In this subsection, the representation of solutions and the fitness function are described, and the search process of the ABC algorithm for finding the initial codebook is also presented.

3.2.1 Representation of solutions

In the initialization phase, the *SN* solution s_i (i = 0, 1, 2, …, SN–1) is generated, where $s_i = (s_{i,0}, s_{i,1}, s_{i,2}, \ldots, s_{i,D\text{-}1})$, $s_{i,j}$ is the real number and the $0 \le s_{i,j} \le s_{i,j+1} \le M$–1 for all j. The value of D is equal to the codebook size minus one. The solution $s_i = (s_{i,0}, s_{i,1}, s_{i,2}, \ldots, s_{i,D\text{-}1})$ corresponds to a codebook CB_i as follows:

$$CB_i = \{sort\_x_{B[0]}, sort\_x_{B[1]}, sort\_x_{B[2]}, \ldots, sort\_x_{B[D]}\}, \tag{7}$$

where $B[0]=\left\lfloor \frac{s_{i,0}}{2} \right\rfloor$,

$B[j]=\left\lfloor \frac{s_{i,j\text{-}1} + s_{i,j}}{2} \right\rfloor, j = 1, 2, \ldots, D\text{–}1,$

$B[D]=\left\lfloor \frac{s_{i,D\text{-}1} + M - 1}{2} \right\rfloor.$

For example, the values of M and D are set to 16,384 and 3, respectively. That is, the codebook size is 4. If the solution $s_i = (s_{i,0}, s_{i,1}, s_{i,2})$ with the values of $s_{i,0}$, $s_{i,1}$ and $s_{i,2}$ are 100, 1,000 and 10,000, respectively, we can compute that $B[0]=\left\lfloor \frac{100}{2} \right\rfloor=50$, $B[1] = \left\lfloor \frac{100+1000}{2} \right\rfloor=550$, $B[2]=\left\lfloor \frac{1000+10000}{2} \right\rfloor=5500$, and $B[3]=\left\lfloor \frac{10000+16384-1}{2} \right\rfloor=13191$. Therefore, the solution s_i corresponds to a codebook $CB_i = \{sort\_x_{50}, sort\_x_{550}, sort\_x_{5500}, sort\_x_{13191}\}$.

3.2.2 Fitness function

The solution $s_i = (s_{i,0}, s_{i,1}, s_{i,2}, \ldots, s_{i,D\text{-}1})$ corresponds to a codebook $CB_i = \{sort\_x_{B[0]}, sort\_x_{B[1]}, sort\_x_{B[2]}, \ldots, sort\_x_{B[D]}\}$. Codeword $sort\_x_{B[0]}$ represents a group of vectors $\{sort\_x_k \mid 0 \le k \le \lfloor s_{i,0} \rfloor\}$. In order to reduce the computation time, the distortion E_0 is approximately computed as follows:

$$E_0 = \left(\sum_{r \in SET_0} \left\| sort\_x_{B[0]}, sort\_x_r \right\| \right) \times \frac{s_{i,0}}{4}, \tag{8}$$

where $SET_0=\{0, \left\lfloor \frac{s_{i,0}}{3} \right\rfloor, \left\lfloor \frac{2 \times s_{i,0}}{3} \right\rfloor, \lfloor s_{i,0} \rfloor\}$, and $||sort\_x_{B[0]}, sort\_x_r||$ is the Euclidean distance tance between $sort\_x_{B[0]}$ and $sort\_x_r$. Codeword $sort\_x_{B[j]}$ (j = 1, 2, …, D–1) represents a group of vectors $\{sort\_x_k \mid \lfloor s_{i,j\text{-}1} \rfloor \le k \le \lfloor s_{i,j} \rfloor\}$, and the distortion E_j (j = 1, 2, …, D–1) is approximately computed as follows:

$$E_j = \left(\sum_{r \in SET_j} \left\| sort\_x_{B[j]}, sort\_x_r \right\| \right) \times \frac{s_{i,j} - s_{i,j-1}}{4} \tag{9}$$

where $SET_j = \{\lfloor s_{i,j\text{-}1} \rfloor, \left\lfloor s_{i,j\text{-}1} + \frac{s_{i,j} - s_{i,j\text{-}1}}{3} \right\rfloor, \left\lfloor s_{i,j\text{-}1} + \frac{2 \times (s_{i,j} - s_{i,j\text{-}1})}{3} \right\rfloor, \lfloor s_{i,j} \rfloor\}$. Codeword

$sort\_x_{B[D]}$ represents a group of vectors $\{sort\_x_k \| s_{i,D\text{-}1}\rfloor \le k \le M-1\}$, and the distortion E_D is approximately computed as follows:

$$E_D = \Big(\sum_{r \in SET_D} \| sort\_x_{B[D]}, sort\_x_r \|\Big) \times \frac{M-1-s_{i,D-1}}{4}, \tag{10}$$

where $SET_D = \{\lfloor s_{i,D\text{-}1} \rfloor, \left\lfloor s_{i,D\text{-}1} + \frac{M-1-s_{i,D\text{-}1}}{3} \right\rfloor, \left\lfloor s_{i,D\text{-}1} + \frac{2\times(M-1-s_{i,D\text{-}1})}{3} \right\rfloor, M-1\}$.

The objective function value $obj(s_i)$ is calculated as follows:

$$obj(s_i) = (E_0 + E_1 + E_2 + \ldots + E_D)/M. \tag{11}$$

Then, the fitness value $fit(s_i)$ can be calculated by the following expression:

$$fit(s_i) = \frac{1}{obj(s_i)+1}. \tag{12}$$

3.2.3 The search process

The employed bee phase: In this phase, the ith (i = 0, 1, 2, ..., SN–1) employed bee produces a new solution v_i by using Eq. (5) and computes the fitness value of the new solution. To ensure that $v_{i,j} \le v_{i,j+1}$, the elements of each solution vector are sorted in ascending order of magnitude. If the fitness of the new solution v_i is higher than that of the old solution s_i, the solution s_i is replaced by v_i and $trial_i$ is set to 0; otherwise the old solution s_i is kept and $trial_i$ is increased by 1.

The onlooker bee phase: First, the probability value $prob_i$ is calculated by using Eq. (6) for the solution s_i. Each onlooker bee which selects a solution s_i depending on $prob_i$ also produces a new solution v_i by using Eq. (5). Then, it applies the greedy selection process between v_i and s_i. If the fitness of the new solution v_i is higher than that of the old solution s_i, the solution s_i is replaced by v_i and $trial_i$ is set to 0; otherwise the old solution s_i is kept and $trial_i$ is increased by 1.

The scout bee phase: As described above, if the solution s_i is not improved through the employed phase and the onlooker bee phase, the $trial_i$ value of solution s_i will be increased by 1. If the $trial_i$ of solution s_i is more than the parameter '*limit*', the solution s_i is considered to be an abandoned solution; meanwhile, the employed bee will be changed into a scout. The scout randomly produces the new solution to replace s_i. Then, the value of $trial_i$ is set 0, and this scout is changed into an employed bee.

Check the termination criterion: If the *cycle* is greater than the maximum cycle number (*MCN*), then the ABC algorithm is finished and the best solution found so far is output.

3.3 LBG Algorithm

The steps of the LBG algorithm [1] are described as follows:

Step 1: (Initialization)

The best solution obtained by the ABC algorithm represents a codebook C. This codebook C serves as the initial codebook for the LBG algorithm

Set *iter* to 1

Step 2: (Clustering)

For each vector in the set $X=\{x_0, x_1, x_2, x_3,\ldots, x_{M-1}\}$, search for the closest codeword in the current codebook C, and then add the vector into the corresponding cluster

iter=*iter*+1

Step 3: (Iteration or termination)

Evaluate each centroid of its associated cluster and replace the old codewords of codebook C with these centroids. Repeat Steps 2-3 until the termination criterion *iter* ≥ *LBG_iter* is satisfied.

4. EXPERIMENTAL RESULTS

The purpose of this section is to evaluate the performance of the HT-ABC-LBG algorithm by experiments and to compare it with the performances of the FF-LBG algorithm [5] and the BA-LBG algorithm [6]. Four gray-level images "Lena", "Baboon", "Pepper" and "Goldhill" shown in Fig. 4 are used as the test images, each of which consists of 512×512 pixels. The test image is partitioned into non-overlapping blocks with 4×4 pixels. Every block is treated as a 16-dimensional column vector. Therefore, there exist a total of 16,384 input vectors. That is, the value of M is 16,384.

The program is coded in C language and run on a PC with a Pentium 4 CPU. In the proposed algorithm, the following parameter values are used: *SN*=100, *limit*=50, *MCN*=100, and *LBG_iter*=20. For the comparison of image compression performance, the image quality is evaluated by the peak signal to noise ratio (PSNR), which is defined as

$$PSNR = 10 \times \log_{10}\left(\frac{255^2}{MSE}\right)(\text{dB}), \tag{13}$$

where MSE is the mean square error for a 512×512 grayscale image and is defined as:

$$MSE = \left(\frac{1}{512}\right)^2 \sum_{i=1}^{512}\sum_{j=1}^{512}\left(f_{ij} - \overline{f_{ij}}\right)^2, \tag{14}$$

where f_{ij} and $\overline{f_{ij}}$ denote the pixel values of the original and decompressed images, respectively.

In the experiments, the proposed algorithm is executed 10 times. The six different codebooks with sizes of 32, 64, 128, 256, 512 and 1024 are implemented. Compared with the FF-LBG algorithm, the average PSNR for the different codebook sizes of the two algorithms are depicted in Figs. 5-8. For the image "Lena", Fig. 5 shows that the proposed algorithm outperforms the FF-LBG algorithm by 3.84-4.56 dB. For the image "Baboon", Fig. 6 shows that the proposed algorithm outperforms the FF-LBG algorithm by 0.68-1.16 dB. For the image "Pepper", Fig. 7 shows that the proposed algorithm outperforms the FF-LBG algorithm by 0.68-1.24 dB. For the image "Goldhill", Fig. 8 shows that the proposed algorithm outperforms the FF-LBG algorithm by 1.24-1.78 dB. Figs.

9-12 show the decompressed images obtained by the proposed algorithm for the images "Lena", "Baboon", "Pepper" and "Goldhill", respectively. The results show that the proposed algorithm provides good quality decompressed images.

Table 2 compares the average computation times of the proposed algorithm with the FF-LBG algorithm for the generation of codebooks of different sizes. It is obvious that the computation time of using the proposed algorithm is less than that of using the FF-LBG algorithm. It also indicates that the computation time for generating codebooks needed by the proposed algorithm increases with the codebook size.

In the proposed algorithm, the input vectors are sorted by Hotelling Transform. Because the first principal component is computed by preserving the maximum variability in the original data set, some of the sorted vectors can be selected by the ABC algorithm to represent approximately all of the input vectors. These selected sorted vectors form an initial codebook. This codebook serves as the initial codebook for the LBG algorithm. Obviously, a high-quality codebook of vector quantization is generated. It is the main reason that the proposed algorithm can provide better performance.

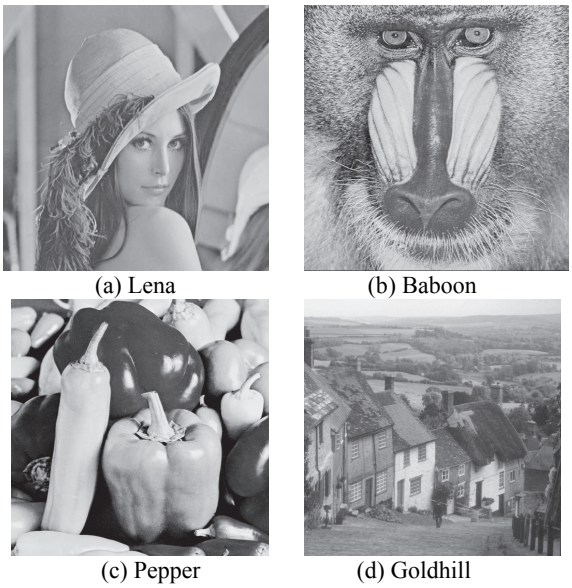
In this study, the codebook size is D+1. If the value of D is 31(63, 127, 255, 511, 1023), the codebook size is 32(64, 128, 256, 512, 1024). In the proposed algorithm, the solution dimension is D for the ABC algorithm. However, the solution dimension is 16×(D+1) in the FF-LBG algorithm. On the other hand, the proposed algorithm can efficiently compute the objective function value $obj(s_i)$ of the solution s_i as described in Subsection 3.2.2. It is the main reason that the computation time of using the proposed algorithm is less than that of using the FF-LBG algorithm.

In addition, two gray-level images "Elaine" and "Truck," shown in Fig. 13, are also used for testing, each of which consists of 512×512 pixels. The average PSNR for the different codebook sizes are depicted in Fig. 14. Figs. 15 and 16 show the decompressed images obtained by the proposed algorithm for the images "Elaine" and "Truck", respectively. The results also show that the proposed algorithm provides good quality decompressed images.

Compared with the BA-LBG algorithm, the average PSNR for the different codebook sizes of the two algorithms are depicted in Figs. 17-19. The proposed algorithm also obtains better PSNRs than the BA-LBG algorithm does. For the image "Lena", Fig. 17 shows that the proposed algorithm outperforms the BA-LBG algorithm by 3.81– 4.55 dB. For the image "Baboon", Fig. 18 shows that the proposed algorithm outperforms the BA-LBG algorithm by 0.01-1.10 dB. For the image "Goldhill", Fig. 19 shows that the proposed algorithm outperforms the BA-LBG algorithm by 0.58-1.23 dB. The BA-LBG algorithm is simulated in MATLAB version 7.9.0. Table 3 compares the average computation times of the proposed algorithm with the BA-LBG algorithm for the generation of codebooks of different sizes.

In this study, a codebook generation method, called the HT-SIMPLE-LBG algorithm, is implemented for comparison. In the HT-SIMPLE-LBG algorithm, a simple method is adopted to generate the initial codebook. First, D random integer numbers $rint(i)$, i = 1, 2, …, D are generated, where $0 < rint(1) < rint(2) < \ldots < rint(D) < 16383$. The set $X=\{sort\_x_0, sort\_x_1, sort\_x_2, \ldots, sort\_x_{16383}\}$ is divided into D+1 parts, where D+1 is the codebook size. That is, the 0th part is the subset $\{sort\_x_i | i=0,1,\ldots rint(1)\}$, the 1-th part is the subset $\{sort\_x_i | i=rint(1)+1, rint(1)+2,\ldots,rint(2)\}$, the Dth part is the subset $\{sort\_x_i | i=rint(D)+1, rint(D)+2, \ldots, 16383\}$, and so on. Then, select a codeword ran-

domly from each part to form a codebook. This codebook serves as the initial codebook for the LBG algorithm. Compared with the HT-SIMPLE-LBG algorithm, the average PSNR for the different codebook sizes of the two algorithms are depicted in Figs. 20-23. For the image "Lena", Fig. 20 shows that the proposed algorithm outperforms the HT-SIMPLE-LBG algorithm by 0.56–1.09 dB. For the image "Baboon", Fig. 21 shows that the proposed algorithm outperforms the HT-SIMPLE-LBG algorithm by 0.43–0.62 dB. For the image "Pepper", Fig. 22 shows that the proposed algorithm outperforms the HT-SIMPLE-LBG algorithm by 0.64–1.07 dB. For the image "Goldhill", Fig. 23 shows that the proposed algorithm outperforms the HT-SIMPLE-LBG algorithm by 0.55–0.73 dB. On average, the PSNRs of decompressed images reconstructed by the proposed algorithm are higher than those obtained by the HT-SIMPLE-LBG algorithm.



(a) Lena (b) Baboon

(c) Pepper (d) Goldhill

Fig. 4. The test images.

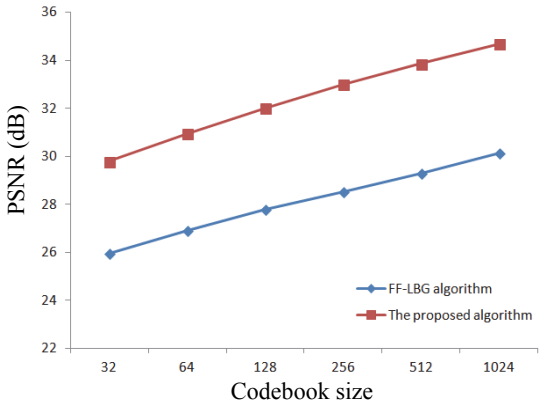


Fig. 5. PSNR comparisons for the image "Lena" using the FF-LBG algorithm and the proposed algorithm.

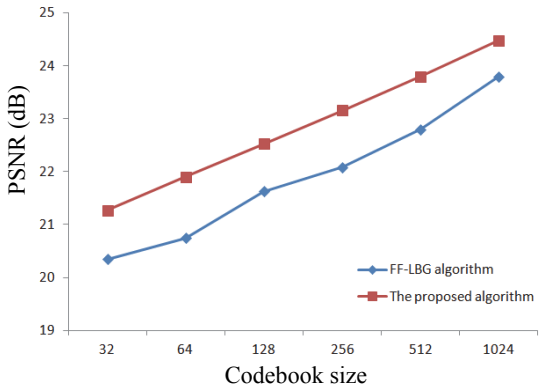


Fig. 6. PSNR comparisons for the image "Baboon" using the FF-LBG algorithm and the proposed algorithm.

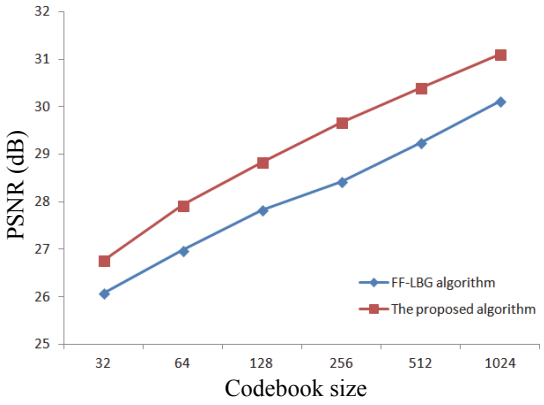


Fig. 7. PSNR comparisons for the image "Pepper" using the FF-LBG algorithm and the proposed algorithm.

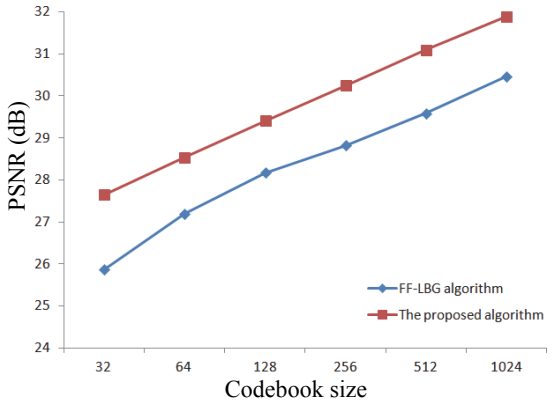
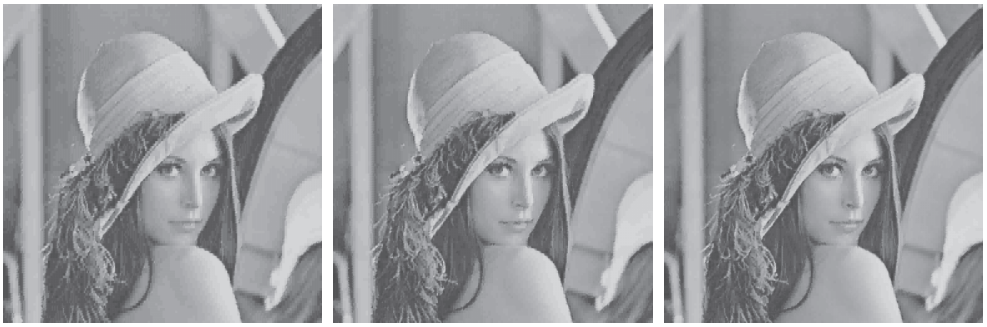
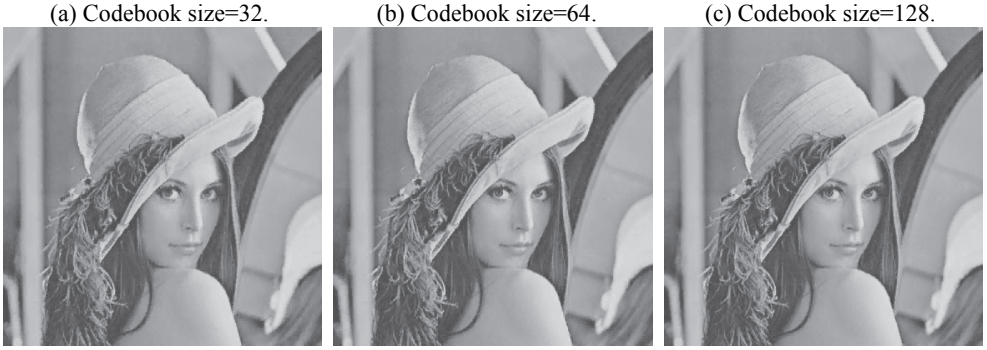


Fig. 8. PSNR comparisons for the image "Goldhill" using the FF-LBG algorithm and the proposed algorithm.

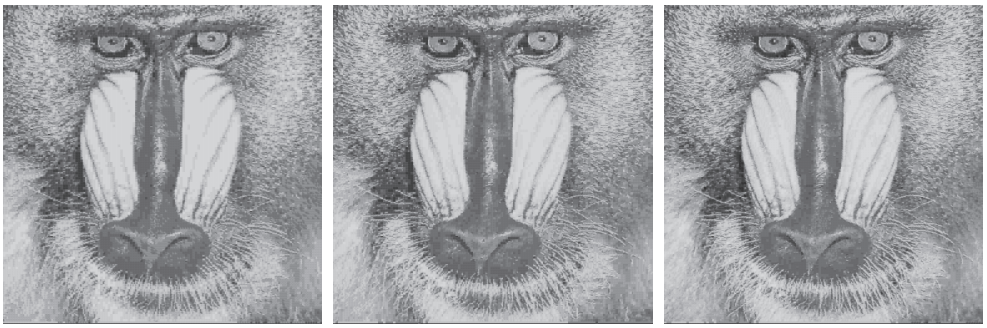


(a) Codebook size=32. (b) Codebook size=64. (c) Codebook size=128.

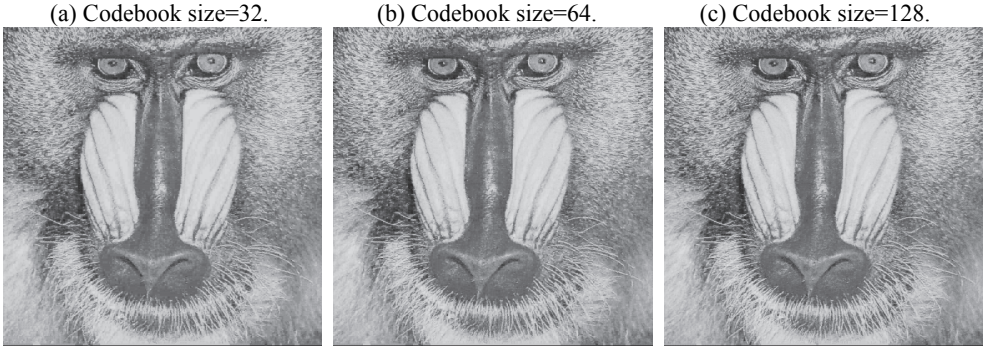


(d) Codebook size=256. (e) Codebook size=512. (f) Codebook size=1024.

Fig. 9. The decompressed image "Lena" by the proposed algorithm.



(a) Codebook size=32. (b) Codebook size=64. (c) Codebook size=128.



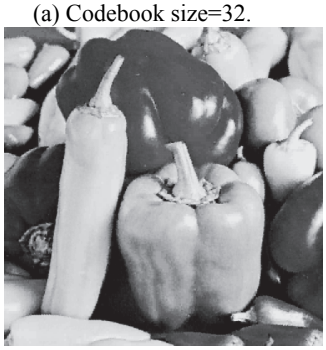
(d) Codebook size=256. (e) Codebook size=512. (f) Codebook size=1024.

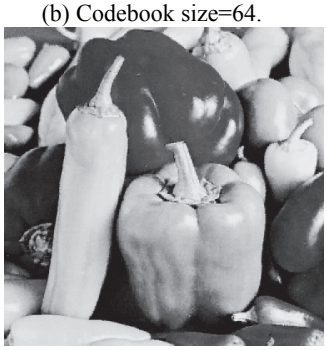
Fig. 10. The decompressed image "Baboon" by the proposed algorithm.


(a) Codebook size=32.


(b) Codebook size=64.


(c) Codebook size=128.


(d) Codebook size=256.


(e) Codebook size=512.

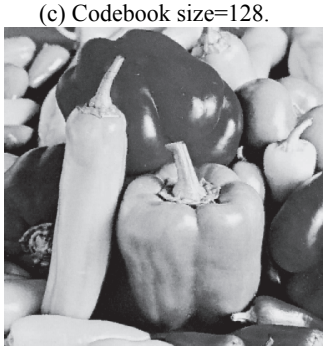
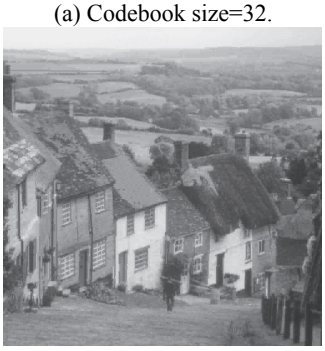

(f) Codebook size=1024.

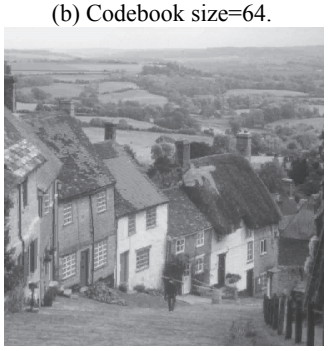
Fig. 11. The decompressed image "Pepper" by the proposed algorithm.


(a) Codebook size=32.


(b) Codebook size=64.


(c) Codebook size=128.


(d) Codebook size=256.


(e) Codebook size=512.

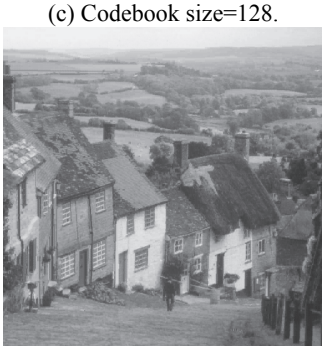
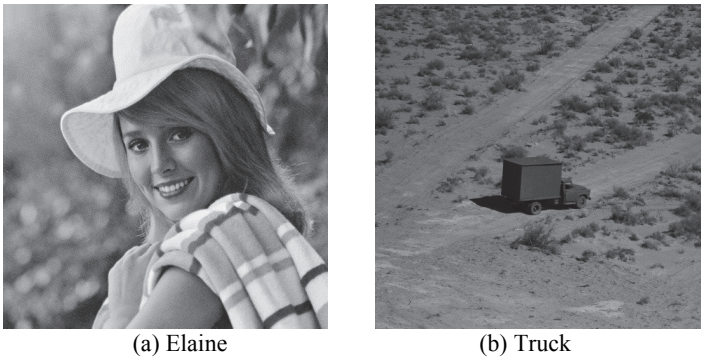

(f) Codebook size=1024.

Fig. 12. The decompressed image "Goldhill" by the proposed algorithm.

Table 2. The average computation time of the test images for codebook generation using the FF-LBG algorithm and the proposed algorithm.

Image(512×512)	Average computation time (sec)	
	FF-LBG algorithm	The proposed algorithm
codebook size=32		
Lena	22.93	1.53
Baboon	24.10	1.53
Pepper	23.82	1.54
Goldhill	22.99	1.54
codebook size=64		
Lena	56.82	2.29
Baboon	54.87	2.30
Pepper	53.55	2.29
Goldhill	56.15	2.29
codebook size=128		
Lena	121.34	3.85
Baboon	111.87	3.77
Pepper	117.54	3.73
Goldhill	125.36	3.79
codebook size=256		
Lena	234.65	6.81
Baboon	228.79	6.69
Pepper	216.52	6.61
Goldhill	239.94	6.75
codebook size=512		
Lena	531.98	12.61
Baboon	492.67	12.41
Pepper	548.34	12.27
Goldhill	572.93	12.52
codebook size=1024		
Lena	906.13	24.01
Baboon	877.97	23.68
Pepper	913.12	23.52
Goldhill	901.64	23.84



(a) Elaine (b) Truck

Fig. 13. Two gray-level images, “Elaine” and “Truck”.

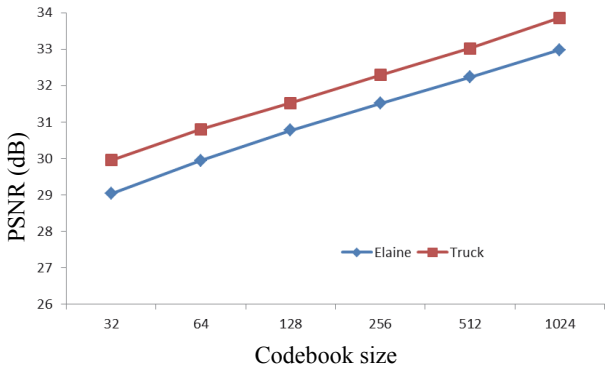
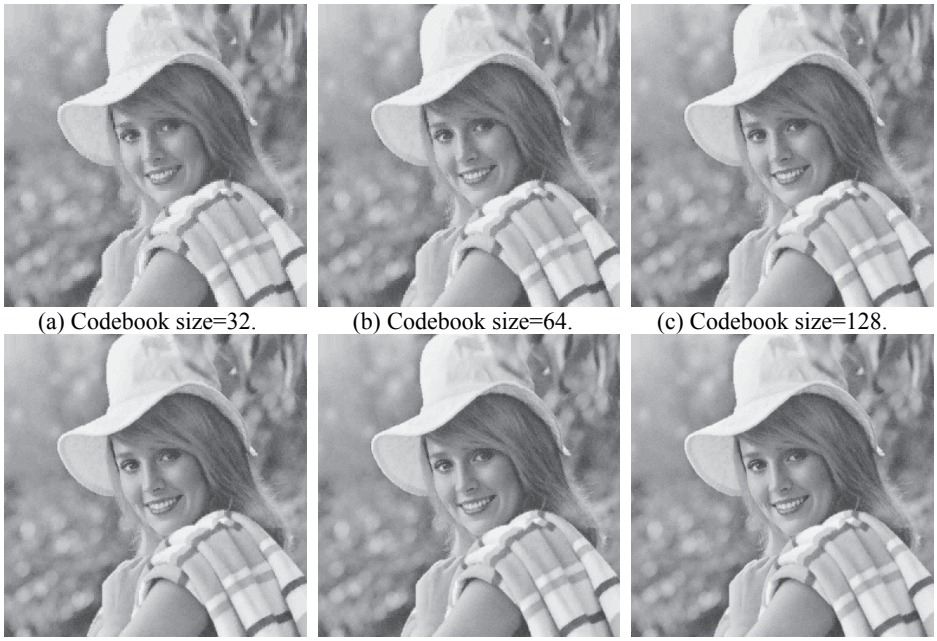


Fig. 14. PSNRs obtained by the proposed algorithm in different size codebooks for the images "Elaine" and "Truck".



(a) Codebook size=32. (b) Codebook size=64. (c) Codebook size=128.

(d) Codebook size=256. (e) Codebook size=512. (f) Codebook size=1024.

Fig. 15. The decompressed image "Elaine" by the proposed algorithm.

(a) Codebook size=32. (b) Codebook size=64. (c) Codebook size=128.

Fig. 16. The decompressed image "Truck" by the proposed algorithm.

(d) Codebook size=256. (e) Codebook size=512. (f) Codebook size=1024.

Fig. 16. (Cont'd) The decompressed image "Truck" by the proposed algorithm.

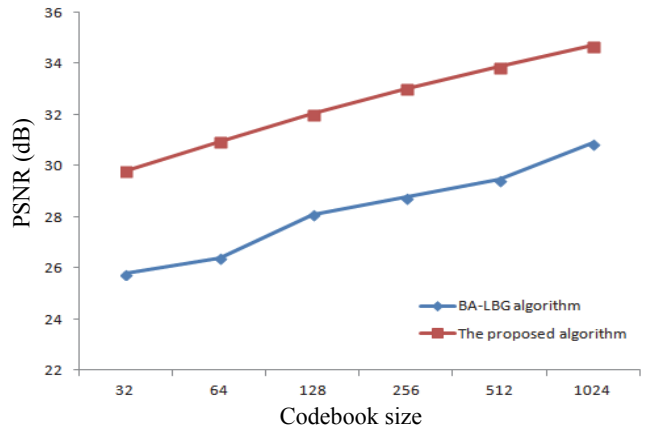


Fig. 17. PSNR comparisons for the image "Lena" using the BA-LBG algorithm and the proposed algorithm.

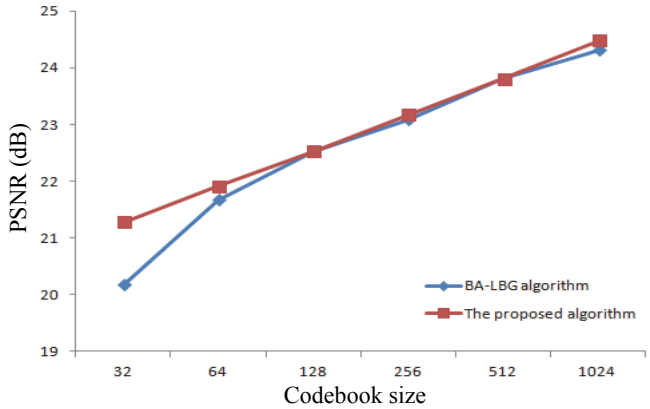


Fig. 18. PSNR comparisons for the image "Baboon" using the BA-LBG algorithm and the proposed algorithm.

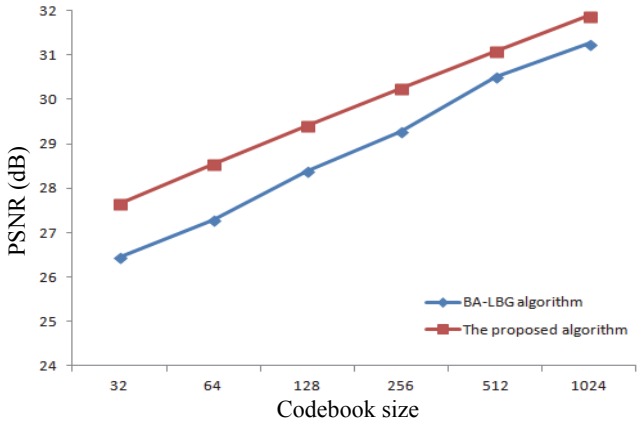


Fig. 19. PSNR comparisons for the image "Goldhill" using the BA-LBG algorithm and the proposed algorithm.

Table 3. The average computation time of the test images for codebook generation using the BA-LBG algorithm and the proposed algorithm.

Image(512×512)	Average computation time (sec)	
	BA-LBG algorithm	The proposed algorithm
codebook size=32		
Lena	343.74	1.53
Baboon	319.59	1.53
Goldhill	279.30	1.54
codebook size=64		
Lena	320.12	2.29
Baboon	395.51	2.30
Goldhill	394.38	2.29
codebook size=128		
Lena	515.17	3.85
Baboon	836.09	3.77
Goldhill	419.51	3.79
codebook size=256		
Lena	665.04	6.81
Baboon	567.64	6.69
Goldhill	974.39	6.75
codebook size=512		
Lena	1342.51	12.61
Baboon	2458.56	12.41
Goldhill	1960.10	12.52
codebook size=1024		
Lena	3511.56	24.01
Baboon	3917.09	23.68
Goldhill	1485.62	23.84

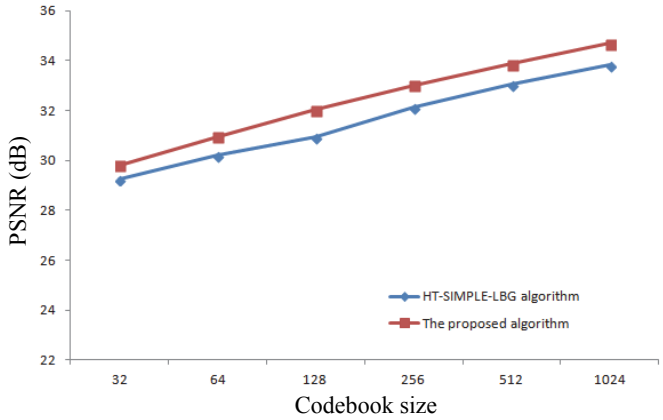


Fig. 20. PSNR comparisons for the image "Lena" using the HT-SIMPLE-LBG algorithm and the proposed algorithm.

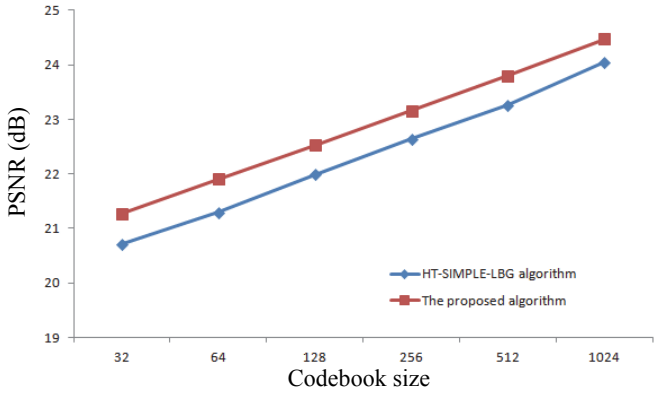


Fig. 21. PSNR comparisons for the image "Baboon" using the HT-SIMPLE-LBG algorithm and the proposed algorithm.

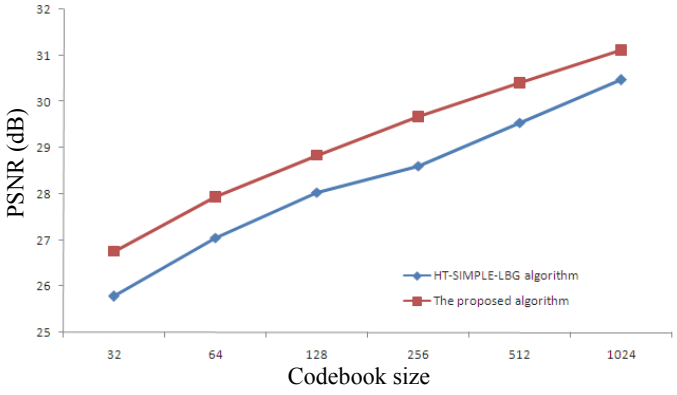


Fig. 22. PSNR comparisons for the image "Pepper" using the HT-SIMPLE-LBG algorithm and the proposed algorithm.

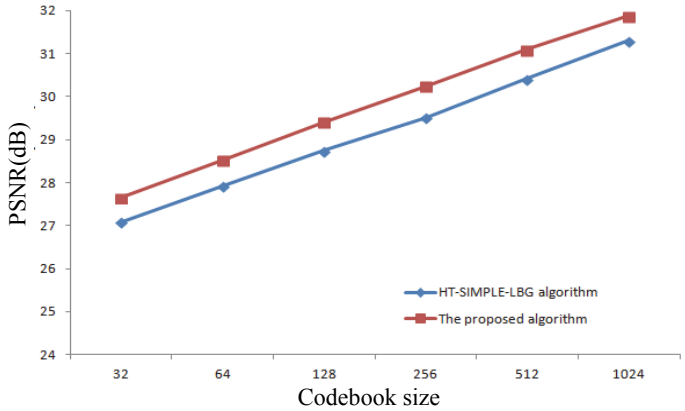


Fig. 23. PSNR comparisons for the image "Goldhill" using the HT-SIMPLE-LBG algorithm and the proposed algorithm.

5. CONCLUSIONS

VQ techniques have been widely used as a powerful approach in image compression. The key to VQ is a good codebook. The major contribution of this study is the finding of search schemes for developing codebook generation. The Hotelling transform is used to sort the input vectors to reduce the computational complexity. The searching ability of the ABC algorithm is used to generate the initial codebook for the LBG algorithm. That is, the search schemes in HT-ABC-LBG first adopt the global search (*i.e.,* HT-ABC) approach and then the local search (*i.e.,* LBG) approach. The experimental results demonstrate that the HT-ABC-LBG algorithm can generate high-quality codebooks. Furthermore, the HT-ABC-LBG algorithm needs much less computation time than the FF-LBG algorithm. With these characteristics, the HT-ABC-LBG algorithm is suitable for image compression.

REFERENCES

1. Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, Vol. 28, 1980, pp. 84-95.
2. J. Z. C. Lai, Y. C. Liaw, and J. Liu, "A fast VQ codebook generation algorithm using codeword displacement," *Pattern Recognition*, Vol. 41, 2008, pp. 315-319.
3. C. W. Tsai, C. Y. Lee, M. C. Chiang, and C. S. Yang, "A fast VQ codebook generation algorithm via pattern reduction," *Pattern Recognition Letters*, Vol. 30, 2009, pp. 653-660.
4. M. H. Horng and T. W. Jiang, "Image vector quantization via honey bee mating optimization," *Expert Systems with Applications*, Vol. 38, 2011, pp. 1382-1392.
5. M. H. Horng, "Vector quantization using the firefly algorithm for image compression," *Expert Systems with Applications*, Vol. 39, 2012, pp. 1078-1091.
6. C. Karri and U. Jena, "Fast vector quantization using a bat algorithm for image compression," *Engineering Science and Technology*, Vol. 19, 2016, pp. 769-781.

7. J. C. Chuang, Y. C. Hu, C. C. Lo, and W. L. Chen, "Grayscale image tamper detection and recovery based on vector quantization," *International Journal of Security and Its Applications*,Vol. 7, 2013, pp. 209-228.
8. A. Swilem, "A fast vector quantization encoding algorithm based on projection pyramid with Hadamard transformation," *Image and Vision Computing*, Vol. 28, 2010, pp. 1637-1644.
9. S. C. Chu, Z. M. Lu, and J. S. Pan, "Hadamard transform based fast codeword search algorithm for high-dimensional VQ encoding," *Information Sciences*, Vol. 177, 2007, pp. 734-746.
10. N. B. Karayiannis and Z. Liu, "Split and merge codebook design algorithms for image compression," *Journal of Electronic Imaging*, Vol. 9, 2000, pp. 509-520.
11. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Addison-Wesley, Reading, MA, 1992.
12. L. Zhang, W. Dong, D. Zhang, and G. Shi, "Two-stage image denoising by principal component analysis with local pixel grouping," *Pattern Recognition*, Vol. 43, 2010, pp. 1531-1549.
13. D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm," *Journal of Global Optimization*, Vol. 39, 2007, pp. 459-471.
14. D. Karaboga and B. Basturk, "On the performance of artificial bee colony (ABC) algorithm," *Applied Soft Computing*, Vol. 8, 2008, pp. 687-697.
15. D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied Mathematics and Computation*, Vol. 214, 2009, pp. 108-132.
16. D. Karaboga and C. Ozturk, "Neural networks training by artificial bee colony algorithm on pattern classification," *Neural Network World*, Vol. 19, 2009, pp. 279-292.
17. C. Ozturk, E. Hancer, and D. Karaboga, "A novel binary artificial bee colony algorithm based on genetic operators," *Information Sciences*, Vol. 297, 2015, pp. 154-170.
18. G. Zhu and S. Kwong, "Gbest-guided artificial bee colony algorithm for numerical function optimization," *Applied Mathematics and Computation*, Vol. 217, 2010, pp. 3166-3173.
19. Q. K. Pan, M. F. Tasgetiren, P. N. Suganthan, and T. J. Chua, "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," *Information Sciences*, Vol. 181, 2011, pp. 2455-2468.
20. M. H. Horng, "Multilevel thresholding selection based on the artificial bee colony algorithm for image segmentation," *Expert Systems with Applications*, Vol. 38, 2011, pp. 13785-13791.
21. W. Gao, S. Liu, and L. Huang, "A global best artificial bee colony algorithm for global optimization," *Journal of Computational and Applied Mathematics*, Vol. 236, 2012, pp. 2741-2753.
22. A. Draa and A. Bouaziz, "An artificial bee colony algorithm for image contrast enhancement," *Swarm and Evolutionary Computation*, Vol. 16, 2014, pp. 69-84.

Shu-Chien Huang (黃樹乾) received the Ph.D. degree in Computer Science and Information Engineering from National Cheng Kung University, Taiwan, in 1999. Currently, he is an Associate Professor in the Department of Computer Science, National Pingtung University, Taiwan. His current research interests include image processing and evolutionary computation.