# Codebook Generation Using Partition and Agglomerative Clustering

Chih-Tang CHANG[1], Jim Z.C. LAI[2], Mu-Der JENG[1]
[1]*Dept. of Electrical Engineering, National Taiwan Ocean University, Keelung, Taiwan 202, R. O. C.*
[2]*Dept. of Computer Science, National Taiwan Ocean University, Keelung, Taiwan 202, R. O. C.*
*D93530005@mail.ntou.edu.tw*

***Abstract*—In this paper, we present a codebook generation algorithm to produce a codebook with lower distortion. Our method combines a fast codebook generation algorithm (CGAUCD) with doubling technique and fast agglomerative clustering algorithm (FACA) to generate a codebook with less computing time and lower distortion. Instead of using FACA directly to divide training vectors into M clusters, our proposed method first generates qM clusters from these training vectors, where q>1 is an integer, and then applies FACA to merge these qM clusters into M cells. This is due to the computational complexity of CGAUCD with doubling technique is less than that of FACA. These M cluster centers are used as the initial codebook for CGAUCD. Using three real images as the training set, our method can reduce the MSE and computing time of FPNN+CGAUCD, which is the available best method to our knowledge, by 0.19 to 0.38 and 74.6% to 84.3%, respectively.**

***Index Terms*—Codebook generation, agglomerative clustering, vector quantization**

## I. INTRODUCTION

Vector quantization (VQ) has been investigated in more than a decade [1-11] for data compression due to its relatively simple structure and computation. Many types of VQ, such as classified VQ [1- 4], finite state VQ [5], and side-match VQ [6], have been implemented for various purposes. VQ has been applied to some other applications including inverse halftoning [7-9]. The operations of VQ consist of separating the signal to be compressed into vectors (or blocks) and finding the closest codeword of an input vector from the codebook. This most similar codeword is called the reproduction vector of the input vector. In the encoding process of VQ, an index, which points to the closest codeword of an input vector, is found. In the decoding process, this index is used to fetch a codeword from the identical codebook.

A good codebook is important for VQ. The method most commonly used is the Linde-Buzo-Gary (LBG) algorithm [10, 11], which is also called the generalized Lloyd algorithm (GLA). GLA performs iteratively the partition step and new codebook generation step until convergence. The main drawback of GLA is that it gets stuck to the local optimal solution. To solve this problem, simulated annealing was presented [10]. However, simulated annealing requires a large amount of computing time and gains a little improvement only [12]. Another approach of obtaining a codebook is agglomerative clustering [13], which can usually obtain a better codebook than GLA [14]. For a data set of N training vectors, the computational complexities, in terms of distance calculations, of GLA and agglomerative clustering are O(NMt) and $O(N^3)$, respectively, where M is the number of codewords in a codebook, and t is the number of iterations. It is noted that M<< N and t<< N, in general.

The generated codebook using GLA depends on the initial codebook. With a good initialization, a good codebook can be obtained by GLA. Using the cluster centers obtained by agglomerative clustering as the initialization for GLA, the corresponding codebook can further be improved. The computational complexities of GLA and agglomerative approach can be reduced by many available methods [14-16]. In [15], an FPNN (fast pairwise nearest neighbor) algorithm is proposed to reduce the computational complexity of agglomerative clustering. The computational complexity, in terms of the number of distance calculations, of FPNN is $O(\tau N^2)$, where $\tau$ is the average number of clusters to be updated for each merging process. In this paper, we will use the approaches presented in [16] to reduce the computational complexity of our proposed method. To use the developed fast agglomerative clustering algorithm, the nearest neighbor for each data points should be determined in the initialization process. To our knowledge, the fast k-nearest-neighbor search algorithm (FKNNSA) [17] is the fastest method and we will use it to determine the nearest neighbor of a data point for agglomerative clustering.

In this paper, we will use a method to divide a set of training vectors into qM clusters, where q>1 is an integer and M is the codebook size. Then the developed fast agglomerative clustering algorithm is used to merge these qM clusters into M ones. Finally, CGAUCD (codebook generation algorithm using codeword displacement) [16] is used to generate the desired codebook of size M. It is noted that CGAUCD is a partitioning clustering algorithm.

This paper is organized as follows. Section II describes the related works. Section III presents the algorithms developed in this paper. Some experimental results are given in Section IV and concluding remarks are presented in Section V.

## II. RELATED WORKS

In this section, we will describe CGAUCD (codebook generation algorithm using codeword displacement) [16], CGAUCD with doubling technique, and fast PNN algorithm [15].

### (A) CGAUCD with Doubling Technique

The most well known algorithm for generating a

Digital Object Identifier 10.4316/AECE.2011.03015

codebook is the generalized Lloyd algorithm (GLA). Denote the set of training vectors as S ={X}. Let d(X,Y) be the distortion between any two vectors X and Y. In this paper, d(X,Y) is defined as the squared Euclidean distance between X and Y. To reduce the computational complexity of GLA, we will use CGAUCD [16] to generate the desired codebook for it is a fast version of GLA and has the least computing time as far as we know.

The squared Euclidean distance between an input vector $X = (x_1, x_2,\ldots, x_d)^t$ and a codeword $C = (c_1, c_2,\ldots, c_d)^t$ is defined by the following equation:

$$d(X,C) = [\sum_{i=1}^{d} |x_i - c_i|^2] \quad (1)$$

Let the center of cluster $R_a$ in the current and previous partitions be $C_a$ and $C'_a$, respectively. Denote the squared displacement between $C_a$ and $C'_a$ as $D_a$. If $D_a = 0$, then the vector $C_a$ is defined as a static cluster center; otherwise it is called an active cluster center. If $C_a$ is active, then $R_a$ is called an active cluster; otherwise $R_a$ is defined as a static cluster. Denote the squared Euclidean distances between a training vector X and the corresponding nearest as well as second nearest codewords in the previous partition as $r'_1$ and $r'_2$, respectively.

Let the subcodebook $CB_{active}$ consist of active codewords. Suppose that the training vector X is in a cluster Ra. Let $r_1 = d(X, C_a)$. In the case that $R_a$ is static and a codeword $C'_b$ is not the nearest codeword of X in the previous partition, then $C'_b$ can't be the nearest codeword of X in the current partition also if $C_b$ is static. If X is in an active cluster $R_a$, we will first calculate $r_1 = d(X,C_a)$. If $r_1 < r'_2$, the nearest codeword of X in the current iteration is impossible from the set of static codewords and we can find the closest codeword of X from $CB_{active}$. In the case of $r_1 \geq r'_2$, we must perform full search to find the closest codeword of X. Now, we would like to present CGAUCD below for the reason of completeness.

CGAUCD

(1) Give an initial codebook $CB_0$ and preprocess all training vectors for fast search. Perform the partition process using codebook $CB_0$, calculate $r'_1$ and $r'_2$ for each training vector **X** and generate a new codebook *CB*.

(2) Generate the subcodebook $CB_{active}$ and search structures for *CB* and $CB_{active}$.

(3) For each training vector **X**, perform the following partition process:

3a. If **X** is in a static cluster, search the codewords from $CB_{active}$ to determine its nearest codeword and update $r'_1$ and $r'_2$.

3b. If **X** is in an active cluster with center $\mathbf{C}_a$, calculate $r_1$ between **X** and $\mathbf{C}_a$. If $r_1 < r'_2$, search subcodebook $CB_{active}$ to find the nearest codeword of **X** and update $r'_1$ and $r'_2$; otherwise determine its nearest codeword of **X** from the codebook *CB* and update $r'_1$ and $r'_2$.

(4) Genarate the new codebook *CB* and subcodebook $CB_{active}$ as well as the search structures for *CB* and $CB_{active}$.

(5) Go to step 3 until the codebook is converged.

CGAUCD with doubling technique, referred to as CGAUCD+DT, generates M cluster centers in s steps, where $s = \log_2 M$. Let $SC(j) = \{C_1, C_2, \ldots, C_{2^j}\}$ consist of $2^j$ cluster centers. To generate $2^{j+1}$ cluster centers using CGAUCD+DT, we first let $SC^*(j+1) = \{C^*_1, C^*_2, \ldots, C^*_{2^{j+1}}\}$, where $C^*_i = C_{i/2} - \varepsilon$ if i is even; $C^*_i = C_{(i-1)/2} + \varepsilon$ if i is odd; and ε is a small random vector. It is noted that each component of ε is generated randomly to be a small positive real number. Using SC*(j+1) as the set of initial cluster centers, we can use CGAUCD to generate $SC(j+1) = \{C_1, C_2, \ldots, C_{2j+1}\}$. CGAUCD with doubling technique is presented as follows:

CGAUCD with Doubling Technique

(1) Compute $SC(0) = \{\mathbf{C}_1\}$ from the training set S, where $\mathbf{C}_1 = (\sum_{i=1}^{N} \mathbf{X}_i)/N$, and *N* is the number of data points. Set $s = \log_2 M$ and j = 1, where M is the number of clusters.

(2) Let $SC^*(j) = \{C^*_1, C^*_2, \ldots, C^*_{2j}\}$, where $C^*_i = C_{i/2} - \varepsilon$ if i is even; $C^*_i = C_{(i-1)/2} + \varepsilon$ if i is odd. Use SC*(j) as the set of initial cluster centers for CGAUCD to generate $SC(j) = \{C_1, C_2, \ldots, C_{2j}\}$.

(3) Set j = j+1. If j > s stop; otherwise go to step (2).

(B)Fast PNN Algorithm

The fast PNN (pairwise nearest neighbor) algorithm, divides a set of N training vectors into M clusters through a sequence of merge operations. The increase of distortion of merging two clusters Ra and Rb into one cluster Rab can be calculated by [18]

$$D_{a,b} = \frac{n_a n_b}{n_a + n_b} d(\mathbf{C}_a, \mathbf{C}_b) \quad (2)$$

where $n_a = |R_a|$; $n_b = |R_b|$; $\mathbf{C}_a$ is the center of $R_a$; and $\mathbf{C}_b$ is the center of $R_b$. $D_{a,b}$ is called the cluster distance of $R_a$ and $R_b$. The cluster center $\mathbf{C}_{ab}$ and cardinality $n_{ab}$ (the number of training vectors) of $R_{ab}$ are updated as follows:

$$\mathbf{C}_{ab} = (n_a \mathbf{C}_a + n_b \mathbf{C}_b)/(n_a + n_b) \quad (3)$$

$$n_{ab} = n_a + n_b \quad (4)$$

At each stage of merge, two clusters which have the least cluster distance are determined and merged. For clarity, clusters $R_a$, $R_b$, and $R_{ab}$ will be abbreviated to clusters *a*, *b*, and *ab*, respectively, in this paper. Fränti et al. [15] used a nearest neighbor table *NNT*, which records the index pointing to a cluster's nearest neighbor and the cluster

distance between the cluster and its nearest neighbor, to reduce the computational complexity of PNN algorithm. It is noted that $NNT[l]$ is the nearest neighbor for cluster $l$. If two clusters $a$ and $b$ are merged, then a set of clusters being updated ($S_u$) is determined, where $S_u = \{l: NNT[l] = a \text{ or } NNT[l] = b\} \setminus \{a, b\}$. Now, we would like to present the FPNN (fast PNN) algorithm.

FPNN Algorithm

(1) For a cluster $l$ ($l = 1, 2, ..., N$), find the corresponding nearest neighbor $NNT[l]$ and generate the nearest neighbor table $NNT$. Set $N_t = N$.
(2) Find two clusters $b$ and $a$, which have the minimum cluster distance. Merge clusters $a$ and $b$ into cluster $ab$.
(3) Find the set $S_u = \{l: NNT[l] = a \text{ or } NNT[l] = b\} \setminus \{a, b\}$.
(4) Update $\mathbf{C}_{ab}$ and $n_{ab}$. Set $n_a = n_{ab}$, $\mathbf{C}_a = \mathbf{C}_{ab}$ and delete cluster $b$. For each cluster in $S_u$, find the corresponding nearest neighbor and update table $NNT$.
(5) Set $N_t = N_t - 1$. If $N_t > M$, go to step (2).

## III. Proposed Algorithm

To generate a codebook with lower distortion, we can use FPNN to generate a set of M cluster centers first. Then, these M cluster centers are used as the initial codebook for CGAUCD. To reduce the computational complexity of FPNN, we will present a fast agglomerative clustering algorithm in this section.

Let $NN_l$ be the nearest neighbor of cluster $l$. For each cluster, we will also maintain a set of clusters, which have cluster $l$ as their nearest neighbor. Denote $INNS_l$ as the set containing clusters, which have cluster $l$ as their nearest neighbor. That is, $INNS_l = \{j: NN_j = l\}$. Denote cluster a as the nearest neighbor of cluster $l$. Here, we set $D_{min}(l) = D_{l,a}$. Let the cluster table CT record the cluster index and nearest distance table NDT store the corresponding nearest distance. It is noted that CT and NDT are arranged in the descending order of $D_{min}(l)$. From CT, we can easily find the clusters being merged. Denote these two clusters being merged as a and b. From $INNS_a$ and $INNS_b$, we can find the corresponding set of clusters whose nearest neighbors need to be updated. That is, the set of candidates being updated is $S_u = INNS_a \cup INNS_b \setminus \{a, b\}$. At this stage, we can calculate the nearest neighbor of cluster $ab$ ($NN_{ab}$) and replace all the information of cluster a by the one of $ab$, where $ab$ is the merged cluster.

In the beginning of determining the nearest neighbor for a cluster $l$, we first select a cluster and calculate its cluster distance to $l$. Let $NN_l$ be the current nearest neighbor of $l$ and denote $r$ as the cluster distance between cluster $l$ and $NN_l$. If another cluster $j$ satisfies the follow expression:

$$D_{\min}(j) \geq r \tag{5}$$

then cluster $j$ can be rejected directly in the process of finding the closest neighbor for $l$. This is due to the cluster distance between $l$ and $j$ should be greater than or equal to the nearest distance $r$. Since the cluster table CT is arranged in the descending order of cluster distance, we will have $NDT[j] \leq NDT[m]$ for $m<j$. That is, if expression (5) is satisfied by a cluster $j$ and $CT[i] = j$, then $CT[t]$ won't be the nearest neighbor of $l$, where $t < i$. Using expression (5) to reject impossible candidates for a cluster's nearest neighbor is a novel approach.

We may expect that expression (5) can have a good performance of rejecting unlikely candidates for a cluster's nearest neighbor, if a good initial cluster is found. If clusters b and a have the minimum cluster distance we will merge them into cluster $ab$. To determine the nearest neighbor for cluster $ab$, we determine the initial candidate through finding the nearest neighbor $NN_{ab}$ from the set $S_u = INNS_a \cup INNS_b \setminus \{a, b\}$. That is, $NN_{ab}$ is used as the initial candidate for the nearest neighbor of cluster ab. In the case of $S_u = \varnothing$, if $CT[N_t-1] \neq b$ we set the initial nearest neighbor $NN_{ab} = CT[N_t-1]$; otherwise we let $NN_{ab} = CT[N_t-2]$, where $\varnothing$ is the empty set and $N_t$ is the number of clusters. To determine the initial nearest neighbor $NN_l$ for a cluster $l \in S_u$, we can find it through finding the nearest neighbor of $l$ from the set $INNS_l \setminus \{a, b\}$. In the case of $INNS_l \setminus \{a,b\} = \varnothing$, if $CT[N_t-1] \neq b$ we set the initial nearest candidate $NN_l = CT[N_t-1]$; otherwise let $NN_l = CT[N_t-2]$.

In the initialization step of our proposed fast agglomerative clustering algorithm, we need to determine the nearest neighbor for each data point. As far as we know, FKNNSA [17] is the fastest algorithm of determining a query point's nearest neighbor. Therefore, we will use it to determine NNl for each cluster $l$ in initialization step. FKNNSA uses a set of inequalities to reject impossible candidates for a query point during the process of finding its nearest neighbors. These inequalities are developed using the geometrical information of the query point and candidates. Now we would like to present our fast agglomerative clustering algorithm (FACA).

Fast Agglomerative Clustering Algorithm

(1) Initially, allocate $N$ data points to $N$ clusters with each cluster having one data point.
  (1a) For each cluster $l$ ($l = 1, 2, ..., N$), use FKNNSA to determine the corresponding nearest neighbor $NN_l$.
  (1b) Generate the cluster table $CT$, nearest distance table $NDT$, and $INNS_l$.
  (1c) Use the nearest distance table $NDT$ to sort the cluster table $CT$ by the descending order of cluster distance. Set $N_t = N$.
(2) Find two clusters $a$ and $b$, which have the minimum cluster distance from the cluster table $CT$, where $CT[N_t] = a$ and $NN_a = b$.
  (2a) Merge clusters $a$ and $b$ into cluster $ab$ and generate $S_u = INNS_a \cup INNS_b \setminus \{a, b\}$. Use equation (3) and (4) to update $\mathbf{C}_{ab}$ and $n_{ab}$. Set $n_a = n_{ab}$ and $\mathbf{C}_a = \mathbf{C}_{ab}$. Remove cluster $b$.
  (2b) If $S_u \neq \varnothing$, determine $NN_{ab}$ from the set $S_u$. In the case of $S_u = \varnothing$, if $CT[N_t-1] \neq b$ set $NN_{ab} = CT[N_t-$

1]; otherwise set $NN_{ab} = CT[N_t\text{-}2]$. Let $r = D_{ab,c}$, where $c = NN_{ab}$. Let $id_u = id = N_t - 1$.

(2c) If $CT[id] = b$, set $id_b = id$ and go to step (2d). Calculate $D_{ab,c}$, where $c = CT[id]$. If $D_{ab,c} < r$, let $r = D_{ab,c}$ and set $NN_{ab} = CT[id]$. If $r < NDT[id]$, set $id_u = id$ and go to step (2e).

(2d) Set $id = id - 1$ and go to step (2c).

(2e) For $j = (id_u + 2)$ to $id_b$, set $CT[j] = CT[j\text{-}1]$ and $NDT[j] = NDT[j\text{-}1]$. Set $NN_a = NN_{ab}$, $CT[id_u+1] = a$, and $NDT[id_u+1] = r$. Let $m = NN_a$ and update $INNS_m = INNS_m \cup \{a\}$.

(3) If $S_u$ is an empty set, go to step (5); otherwise, for each cluster $l \in S_u$:

(3a) If $INNS_l \backslash \{a,b\} \neq \varnothing$, set $NN_l$ as the nearest neighbor of $l$ determined from the set $INNS_l \setminus \{a,b\}$. In the case of $INNS_l \backslash \{a,b\} = \varnothing$, if $CT[N_t\text{-}1] \neq b$ set $NN_l = CT[N_t\text{-}1]$; otherwise let $NN_l = CT[N_t\text{-}2]$. Let $r = D_{l,c}$, where $c = NN_l$. Let $id = id_u = N_t - 1$.

(3b) If $CT[id] = l$, set $id_l = id$ and go to step (3c). Calculate $D_{ab,c}$, where $c = CT[id]$. If $D_{ab,c} < r$, let $r = D_{ab,c}$ and set $NNl = CT[id]$. If $r < NDT[id]$, set $id_u = id$ and go to step (3d).

(3c) Set $id = id - 1$ and go to step (3b).

(3d) For $j = (id_u+2)$ to $id_l$, set $CT[j] = CT[j\text{-}1]$ and $NDT[j] = NDT[j\text{-}1]$. Set $CT[id_u + 1] = l$ and $NDT[id_u + 1] = r$. Let $m = NN_l$ and update $INNS_m = INNS_m \cup \{l\}$.

(3e) If cluster $b$ is in $INNS_l$, update $INNS_l$ by removing cluster b from $INNS_l$.

(4) Update $INNS_a$ from $S_u$. For each cluster $c$ in $S_u$, insert cluster $c$ into $INNS_a$ if $NN_c$ is cluster $a$.

(5) Set $N_t = N_t - 1$. If $N_t > M$, go to step (2).

The flow chart of FACA is given in figure 1. The computational complexity, in terms of the number of distance calculations, of FACA is $O(\tau\gamma N)$, where $\tau$ is the average number of clusters to be updated for each stage of cluster merge and $\gamma$ is the average number of clusters to be visited in the process of determining a cluster's nearest neighbor. From our experiments, we find that the value of $\tau$ is about 4 and $\tau < N$.

The computational complexity of CGAUCD is $O(utN)$ [16], where $N$ is the number of data points, $t<<N$ is the number of iterations, and $u<M$ is the average number of clusters to be searched to determine a data point's nearest neighbor. CGAUCD with doubling technique (referred to as CGAUCD+DT) performs CGAUCD $\log_2 M$ times to generate a codebook of size $M$. Therefore the computational complexity of CGAUCD+DT is $O(utN\log_2 M)$. That is, the computational complexity of CGAUCD+DT is less than that of FACA. Instead of using FACA to merge $N$ data points into $M$ clusters directly, we can use CGAUCD+DT to generate a set of $qM$ cluster centers first, where $q > 1$, and then apply FACA to merge these $qM$ clusters into $M$ cells to reduce computing time. Finally, these $M$ cluster centers will be used as the initial codebook for CGAUCD. This approach is referred to as algorithm 1 here. Now, we would like to present algorithm 1.

Algorithm 1

(1) Use CGAUCD+DT to generate the set $SC=\{\mathbf{C}_1, \mathbf{C}_2,.., \mathbf{C}_{qM}\}$, which consists of $qM$ cluster centers.

(2) Apply FACA to determine the initial codebook $CB=\{\mathbf{C}_1, \mathbf{C}_2,.., \mathbf{C}_M\}$ with $M$ codewords.

(3) Use CGAUCD to generate the desired codebook.

## IV. Experimental Results

To evaluate the performances of the proposed algorithms, three real data sets and several synthetic data sets have been used as our training sets to generate codebooks. The first real data set having 16,384 data points is obtained from a real image: "Lena;" while the second real data set with 49,152 data points is generated from three real images: "Lena," "Baboon," and "Peppers." The third real data set with size of 98,304 is obtained from six real images: "Tiffany," "Peppers," "Baboon," "Airplane," "Bike" and "Girl." It is noted here that the characteristics of these images are different. The images "Baboon" and "Bike" possess many texture regions; whereas the pictures "Peppers" and "Tiffany" have a lot of smooth areas. "Girl" and "Airplane" have a small fraction of texture and edge areas. The synthetic data sets with size 10,000 and dimensions from 8 to 40 are obtained from the Gauss Markov sequence [10] with $\sigma$=10, $\mu$=0, and $a$=0.9, where $\sigma$ is the standard deviation, $\mu$ is the mean value of the sequence and $a$ is the correlation coefficient. All computing is performed on a Pentium IV 3.2 GHz PC with 512 MB of memory.

Five codebook generation algorithms: GLA, FPNN, FPNN+CGAUCD, FACA+CGAUCD, and our method (algorithm 1) are implemented, where FPNN+CGAUCD and FACA+CGAUCD use the cluster centers generated by FPNN and FACA, respectively, as the initial codebooks for CGAUCD. For the GLA, the initial codewords are selected randomly from the training set. These methods are compared in terms of computing time and mean square error (MSE). The mean square error (MSE) for a set of N training vectors {Xi} is defined as follows:

$$\mathrm{MSE} = \left(\sum_{i=1}^{N} \|\mathbf{X}_i\text{-}\mathbf{Q}(\mathbf{X}_i)\|^2\right)/(Nd) \quad (6)$$

where $\mathbf{Q}(\mathbf{X}_i)$ is the closest codeword (reproduction vector) of $\mathbf{X}$ and $d$ is the vector dimension. MSE is usually used to measure the quality of a picture [10]. The lower MSE implies the better image quality. That is, a codebook with lower MSE implies that it is a better codebook. Therefore MSE is used here as a metric to measure the quality of a codebook.

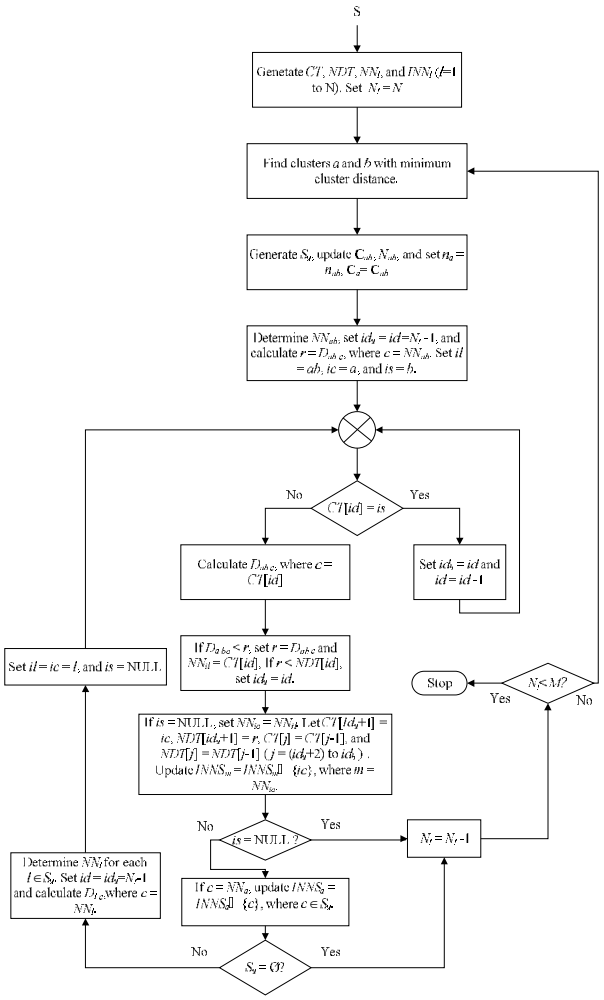



Figure 1. The flow chart of FACA Algorithm 1

TABLE 1: THE COMPUTING TIME OF FPNN AND FACA USING THE FIRST AND SECOND REAL DATA SETS, RESPECTIVELY, TO GENERATE CODEBOOKS OF SIZE 256

| Method | The first data set | The second data set |
|---|---|---|
| FPNN | 83.02 | 1143.28 |
| FACA | 59.89 | 763.52 |

TABLE 2: THE LEAST MEAN SQUARE ERROR OF GLA WITH 100 RUNS AS WELL AS MEAN SQUARE ERRORS OF PNN, PNN+CGAUCD, FACA+CGAUCD AND ALGORITHM 1 USING TRAINING VECTORS FROM THE FIRST REAL DATA SET

| Method | *M* | | |
|---|---|---|---|
| | 128 | 256 | 512 |
| GLA with 100 runs | 53.786457 | 43.434393 | 35.469834 |
| FPNN | 54.121899 | 41.138968 | 30.639848 |
| FPNN+CGAUCD | 52.703916 | 40.108582 | 30.005831 |
| FACA+CGAUCD | 52.705974 | 40.111941 | 29.974793 |
| Algorithm 1, $q$=2 | 52.352139 | 40.315626 | 30.536785 |
| Algorithm 1, $q$=4 | 52.385429 | 40.039429 | 30.066588 |

TABLE 3: THE COMPUTING TIME OF GLA WITH 100 RUNS, PNN, PNN+CGAUCD, FACA+CGAUCD AND ALGORITHM 1 USING TRAINING VECTORS FROM THE FIRST REAL DATA SET

| Method | *M* | | |
|---|---|---|---|
| | 128 | 256 | 512 |
| GLA with 100 runs | 15.83 | 32.19 | 55.03 |
| FPNN | 146.00 | 146.13 | 145.92 |
| FPNN+CGAUCD | 86.69 | 91.55 | 93.72 |
| FACA+CGAUCD | 61.78 | 61.92 | 62.66 |
| Algorithm 1, $q$=2 | 18.60 | 25.89 | 39.86 |
| Algorithm 1, $q$=4 | 26.02 | 38.07 | 57.25 |

TABLE 4: THE LEAST MEAN SQUARE ERROR OF GLA WITH 100 RUNS AS WELL AS MEAN SQUARE ERRORS OF PNN, PNN+CGAUCD, FACA+CGAUCD AND ALGORITHM 1 USING TRAINING VECTORS FROM THE SECOND REAL DATA SET

| Method | *M* | | |
|---|---|---|---|
| | 128 | 256 | 512 |
| GLA with 100 runs | 144.015528 | 122.699041 | 105.406689 |
| FPNN | 147.710213 | 125.642842 | 105.700978 |
| FPNN+CGAUCD | 143.607546 | 121.411422 | 102.054237 |
| FACA+CGAUCD | 143.374579 | 121.545285 | 102.120814 |
| Algorithm 1, $q$=2 | 143.303827 | 121.274645 | 102.202804 |
| Algorithm 1, $q$=4 | 143.178823 | 121.167374 | 101.937034 |

TABLE 5: THE COMPUTING TIME OF GLA WITH 100 RUNS, PNN, PNN+CGAUCD, FACA+CGAUCD AND ALGORITHM 1 USING TRAINING VECTORS FROM THE SECOND REAL DATA SET

| Method | *M* | | |
|---|---|---|---|
| | 128 | 256 | 512 |
| GLA with 100 runs | 107.05 | 207.81 | 421.03 |
| FPNN | 1157.625 | 1143.28 | 1183.18 |
| FPNN+CGAUCD | 1173.535 | 1162.69 | 1202.79 |
| FACA+CGAUCD | 787.66 | 780.93 | 793.08 |
| Algorithm 1, $q$=2 | 143.13 | 191.34 | 250.61 |
| Algorithm 1, $q$=4 | 188.41 | 237.11 | 305.84 |

(6)

Table 1 gives the computing time of FPNN and FACA, which use the first and second real data sets to generate codebooks of size 256. From table 1, we can find that FACA can reduce the computing time of FPNN significantly. FACA can reduce the computing time of FPNN by a factor of about 1.40. Table 2 gives the least mean square error (MSE) of GLA with 100 randomly selected initial codebooks and MSEs of FPNN, FPNN+CGAUCD, FACA+CGAUCD and algorithm 1 with various values of q for the first real data set. Table 3 presents the computing time of GLA with 100 runs, FPNN, FPNN+CGAUCD, FACA+CGAUCD and our method with various values of q for the first real data set. Table 4 lists the least MSE of GLA with 100 runs as well as MSEs of FPNN, FPNN+CGAUCD, FACA+CGAUCD and algorithm 1 for the second data set; while table 5 presents the computing time of GLA with 100

runs, FPNN, FPNN+CGAUCD, FACA+CGAUCD and algorithm 1 for the second data set.

TABLE 6: THE LEAST MEAN SQUARE ERROR OF GLA WITH 100 RUNS AS WELL AS MEAN SQUARE ERRORS OF PNN, PNN+CGAUCD, FACA+CGAUCD AND ALGORITHM 1 USING TRAINING VECTORS FROM THE THIRD REAL DATA SET

| Method | *M* | | |
|---|---|---|---|
| | 128 | 256 | 512 |
| GLA with 100 runs | 153.946964 | 129.462589 | 109.456045 |
| FPNN | 158.015196 | 132.853107 | 112.043991 |
| FPNN+CGAUCD | 153.419479 | 128.515097 | 108.244302 |
| FACA+CGAUCD | 153.033692 | 128.805776 | 108.185836 |
| Algorithm 1, $q$=2 | 153.737790 | 128.575970 | 108.116461 |
| Algorithm 1, $q$=4 | 153.252590 | 128.612478 | 108.057259 |

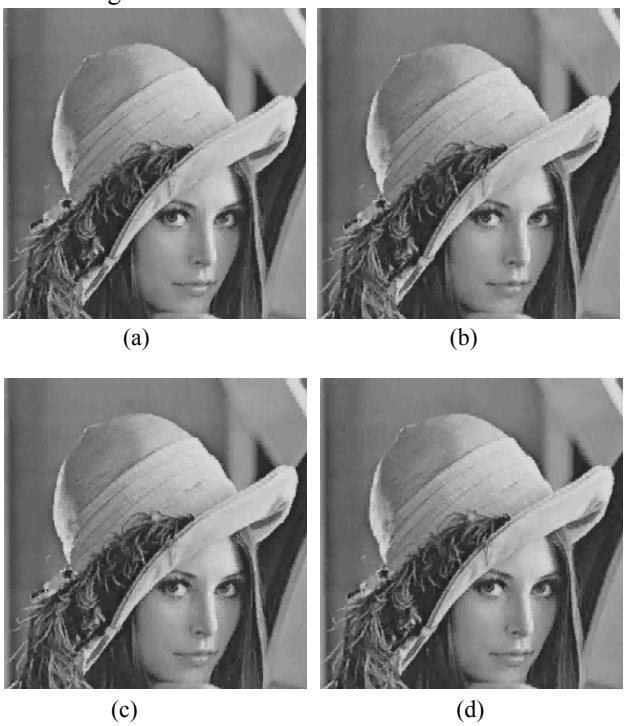
TABLE 7: THE COMPUTING TIME OF GLA WITH 100 RUNS, PNN, PNN+CGAUCD, FACA+CGAUCD AND ALGORITHM 1 USING TRAINING VECTORS FROM THE THIRD REAL DATA SET

| Method | *M* | | |
|---|---|---|---|
| | 128 | 256 | 512 |
| GLA with 100 runs | 273.53 | 436.03 | 909.18 |
| FPNN | 6817.52 | 6779.39 | 6737.73 |
| FPNN+CGAUCD | 6882.86 | 6863.53 | 6840.41 |
| FACA+CGAUCD | 2018.39 | 1685.56 | 1636.61 |
| Algorithm 1, $q$=2 | 321.47 | 452.91 | 583.51 |
| Algorithm 1, $q$=4 | 433.97 | 564.84 | 777.80 |

From table 2 and table 4, we will find that FPNN gives highest mean square error in average and our method (algorithm 1 with q = 4) gives the least mean square errors in almost all cases. Compared to GLA, algorithm 1 with q = 4 can reduce the mean square error by 0.69 to 5.40. Tables 2 and 4 show that both FPNN+CGAUCD and FACA+CGAUCD can obtain about the same mean square error for a codebook. It is noted that there is a little difference between the means square errors of the codebooks generated by FPNN+CGAUCD and FACA+CGAUCD, respectively. In the early stage of merging clusters, there are many cluster pairs with the same cluster distance. Merging different pairs with the same cluster distance in the early stage will result in a little difference for the clustering results. That is, there will be a little difference between the means square errors of the codebooks generated by FPNN+CGAUCD and FACA+CGAUCD due to FPNN and FACA may select different cluster pairs with the same cluster distance. Our method FACA+CGAUCD can reduce the computing time of FPNN+CGAUCD by about 32.9% for the second real data set. From table 3, we will also find that the computing time of algorithm1 with $q$ = 2 or 4 is less than that of FACA+CGAUCD. This is due to the computational complexity of CGAUCD+DT is less than that of FACA. It is noted that to our knowledge, FPNN+CGAUCD is the available algorithm of generating a codebook with the lowest mean square error. Compared to FPNN+CGAUCD, algorithm 1 with $q$ = 2 can generate a codebook with the much less computing time and little higher MSE. Compared with FPNN+CGAUCD, algorithm 1 with $q$ = 4 generates a codebook with the much less computing time and lower MSE. Compared to FPNN+CGACD, algorithm 1 with $q$ = 4 can reduce the average mean square error and computing time by 0.165 and 79.4%, respectively. From tables 3 and 4, we can find that the MSE of algorithms 1 is lower in general when a larger $q$ is used. However, it should be noted that the corresponding computing time is also increased when a larger $q$ is adopted. From table 2, we can conclude that algorithm 1 with $q$ = 4 is not global optimal. Therefore algorithm 1 with $q$ = 2 may occasionally obtain lower MSE than algorithm 1 with $q$ = 4 (for example, $M$ = 128), although their difference is not significant any more.

Figures 2 to 4 presents the compressed images of "Lena," "Baboon," and "Peppers," using codebooks of size 256 generated by GLA with 100 runs, FPNN, FPNN+CGAUCD, and algorithm 1 with q=2, respectively. From figures 2(a), 2(b), 2(c), and 2(d), we can find that GLA and FPNN have more blocking effects in the hat; while FPNN+CGAUCD and algorithm 1 provide less blocking effects and better image quality. For figures 3(a) to 4(d), these four algorithms have almost the same visual quality.

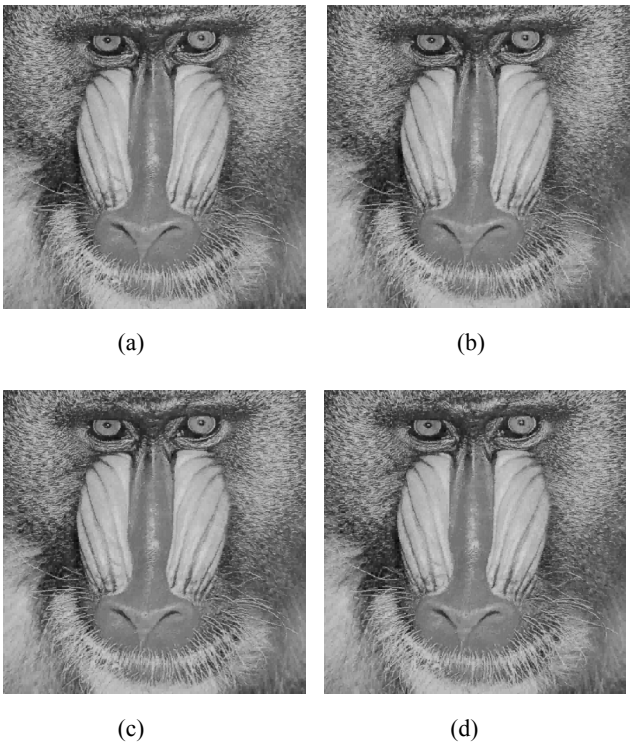
To understand the effect of data size on our proposed methods, a large data set is used here. Table 6 presents the least MSE of GLA with 100 runs as well as MSEs of FPNN, FPNN+CGAUCD, FACA+CGAUCD and algorithm 1 for the third real data set; while table 7 shows the computing time of GLA with 100 runs, FPNN, FPNN+CGAUCD, FACA+CGAUCD and algorithm 1 for the same data set. From table 7, we can find that our method algorithm 1 with q = 2 has the least computing time in average. Compared to FPNN+CGAUCD, algorithm 1 with q = 2 can reduce the computing time significantly and obtain a little higher MSE. From table 2 to table 7, we can conclude that the performances of our proposed methods are more remarkable when a larger data set is used.



(a) (b)

(c) (d)

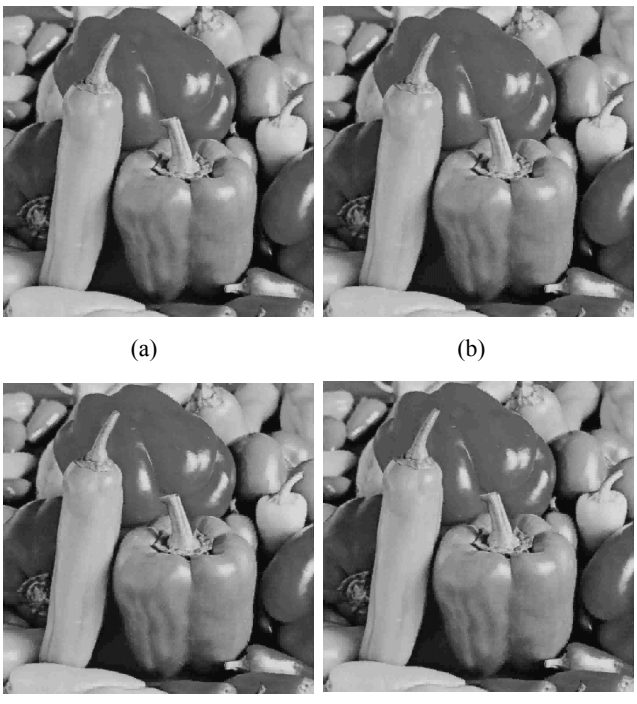
Figure 2. The compressed images of "Lena" using codebooks generated by (a) GLA with 100 runs; (b) FPNN, (c) FPNN+CGAUCD, and (d) algorithm 1 with $q$=2.



(a) (b)

(c) (d)

Figure 3. The compressed images of "Baboon" using codebooks generated by (a) GLA with 100 runs; (b) FPNN, (c) FPNN+CGAUCD, and (d) algorithm 1 with $q$=2.



(a) (b)

(c) (d)

Figure 4. The compressed images of "Peppers" using codebooks generated by (a) GLA with 100 runs; (b) FPNN, (c) FPNN+CGAUCD, and (d) algorithm 1 with $q$=2.

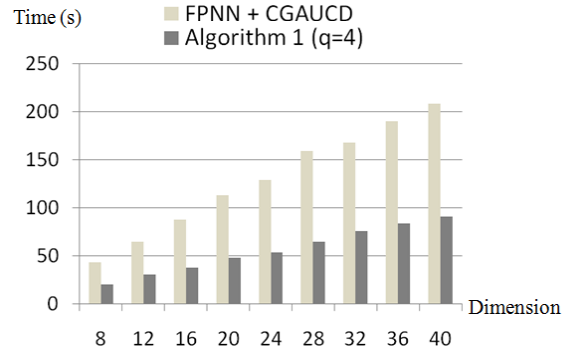



Figure 5. The computing time (in seconds) of generating codebooks of size 256 using synthetic data sets with sizes = 10,000 and dimensions from 8 to 40.

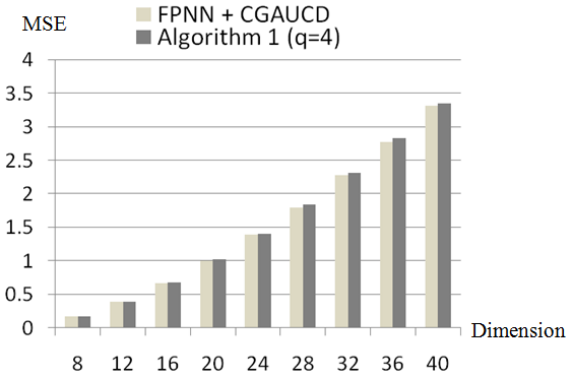



Figure 6. The mean square errors for synthetic data sets of generating codebooks of size 256 using synthetic data sets with sizes = 10,000 and dimensions from 8 to 40.

To study the effect of data dimension on the performances of our proposed methods, figures 5 and 6 present the mean square errors and computing time, respectively, of FPNN+CGACD and algorithm 1 with q = 4 to generate codebooks of size 256 using synthetic data sets with size = 10,000 and dimension ranging from 8 to 40. From figures 4 and 5, we can find that compared to FPNN+CGAUCD, our method algorithm 1 with q = 4 can reduce the average computing time of 60.8% with about the same MSE. The performance of algorithm 1 is better when a data set with higher dimension is used. It is recommended that algorithm 1 with q = 4 can be used to generate the desired codebook for it gives the least mean square error.

## V. Conclusion

In this paper, we develop a method to generate a codebook from a set of training vectors. A fast agglomerative clustering algorithm FACA is also developed to reduce the computing time of FPNN. FACA uses an inequality to speed up the process of finding a cluster's nearest neighbor. Compared to the available best method as far as we know, our method algorithm 1 with q = 4 can reduce the average mean square error by 0.165 with the reduction of computing time by 82.3% to 89.3% for the second real data set. Compared to GLA, our method algorithm 1 with q = 4 can decrease the mean square error of a codebook by 0.69 to 5.40. Compared to FPNN+CGAUCD, which is the available best method to our knowledge, our proposed method FACA+CGAUCD can decrease the computing time by about 32.9% with the same mean square error for the second real data set.

## REFERENCES


[1] Y. C. Liaw, J. Z. C. Lai, and Winston Lo," Image restoration of compressed image using classified vector quantization," Pattern Recognition, vol. 35, no. 2, pp. 181-192, February 2002.

[2] M. Bi , S. H. Ong, and Y. H. Ang , "Wavelet-based image compression using classified interpolative vector quantization," Optical engineering, vol. 47, no. 2, pp. 1528-1535, June 2002.

[3] J. Z. C. Lai, Y. C. Liaw, and Winston Lo, "Artifact reduction of JPEG coded images using mean-removed classified vector quantization," Signal Processing, vol. 82, no. 10, pp. 1375-1388, October 2002.

[4] S. H Hong, R. H Park, S. Yang, and J. Y. Kim, "Image interpolation using interpolative classified vector quantization," Image and Vision Computing, vol. 26, no. 2, pp. 228-239, February 2008, Available: http://dx.doi.org/10.1016/j.imavis.2007.05.002.

[5] Y. L. Huang and R. F. Chang, "A new side-match finite-state vector quantization for image coding," Journal of Visual Communication and Image Representation, vol. 13, no. 3, pp. 335-347, September 2002.

[6] S. B. Yang and L. Y. Tseng, "Smooth side-match classified vector quantizer with variable block size," IEEE Transactions on Image Processing, vol. 10, no. 5, pp. 677-685, May 2001, Available: http://dx.doi.org/10.1109/83.918561.

[7] J. Z. C. Lai and Chen C. C. Chen, "Algorithms of halftoning color images with edge enhancement," Journal of Visual Communication and Image Representation, vol. 14, no. 4, December 2003, pp.389-404..

[8] J. Z. C. Lai and J. Y. Yen, "Inverse error-diffusion using classified vector quantization," IEEE Trans. on Image Processing, vol. 7, no. 12, pp. 1753-1758, December 1998, Available: http://dx.doi.org/10.1109/83.730390.

[9] P. C. Chang, C. S. Yu, and T. H. Lee, "Hybrid LMS-MMSE inverse halftoning technique," IEEE Trans. on Image Processing, vol. 10, no. 1, pp. 95-103, January 2001, Available: http://dx.doi.org/10.1109/83.892446.

[10] *Gersho and R. M. Gray,* Vector Quantization and Signal Compression. *Kluwer Academic Publishers, Boston MA., 1991.*

[11] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," IEEE Trans. on Communications, vol. 28, no. 1, pp. 84-95, January 1980, Available: http://dx.doi.org/10.1109/TCOM.1980.1094577.

[12] J. Z. C. Lai and C. C. Lue, "Fast search algorithms for VQ codebook generation," Journal of Visual Communication and Image Representation, vol. 7, no. 2, pp. 163-168, June 1996.

[13] J. Shanbehzadeh and P. O. Ogunbona, "On the computational complexity of the LBG and PNN algorithm," IEEE Trans. on Image Processing, vol. 6, no. 4, pp. 614-616, April 1997, Available: http://dx.doi.org/10.1109/83.563327.

[14] P. Fränti, O. Virmajoki, and Ville Hautamäki, "Fast agglomerative clustering using a k-nearest neighbor graph," IEEE Trans. on PAMI, vol. 26, no. 11, pp1875-1881, November 2006, Available: http://dx.doi.org/10.1109/TPAMI.2006.227.

[15] T. Kaukoranta, P. Fränti, and O. Nevalainen, "A fast Exact GLA based code vector activity detection," IEEE Trans. on Image Processing, vol. 9, no. 8, pp. 1337-1342, August 2000, Available: http://dx.doi.org/10.1109/83.855429.

[16] Jim Z. C. Lai, Y. C. Liaw, and Julie Liu, "A fast VQ codebook generation using codeword displacement," Pattern Recognition, vol. 41, no. 1, pp. 315-319, January 2008.

[17] Jim Z. C. Lai, Y. C. Liaw, and Julie Liu, "Fast k-nearest-neighbor search based on projection and triangular inequality," Pattern Recognition, vol. 40, no. 2, pp. 351-359, February 2007.

[18] W. H. Equitz, "A new vector quantization clustering algorithm," IEEE Trans. on Acoustics, Speech, and Signal Processing, vol. 37, no. 10, pp.1568-1575, October 1989, Available: http://dx.doi.org/10.1109/29.35395.