Available online at www.sciencedirect.com
ScienceDirect
Pattern Recognition 41 (2008) 689–700
PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY
www.elsevier.com/locate/pr

Constrained-storage multistage vector quantization based on genetic algorithms

Shiueng-Bien Yang*

Department of Computer Science and Information Engineering, Leader University, No. 188 Sec. 5 An-chung Rd., Tainan City, Taiwan

Received 14 April 2006; received in revised form 2 May 2007; accepted 18 May 2007

Abstract

Multistage vector quantization (MSVQ) and their variants have been recently proposed. Before MSVQ is designed, the user must artificially determine the number of codewords in each VQ stage. However, the users usually have no idea regarding the number of codewords in each VQ stage, and thus doubt whether the resulting MSVQ is optimal. This paper proposes the genetic design (GD) algorithm to design the MSVQ. The GD algorithm can automatically find the number of codewords to optimize each VQ stage according to the rate–distortion performance. Thus, the MSVQ based on the GD algorithm, namely MSVQ(GD), is proposed here. Furthermore, using a sharing codebook (SC) can further reduce the storage size of MSVQ. Combining numerous similar codewords in the VQ stages of MSVQ produces the codewords of the sharing codebook. This paper proposes the genetic merge (GM) algorithm to design the SC of MSVQ. Therefore, the constrained-storage MSVQ using a SC, namely CSMSVQ, is proposed and outperforms other MSVQs in the experiments presented here.
© 2007 Pattern Recognition Society. Published by Elsevier Ltd. All rights reserved.

Keywords: Multistage vector quantizer; Genetic algorithm

1. Introduction

Vector quantization (VQ) [1,2] is an effective technique for data compression and coding, especially image and speech coding. The codebook design is the key issue for VQ. Different VQ schemes are compared based on rate–distortion performance. Residual vector quantization (RVQ) is a simple constrained-storage vector quantizer (CSVQ) and a type of multiple stage vector quantization (MSVQ). In Refs. [3–13], MSVQ and its variants were proposed. Fig. 1 shows the basic structure of MSVQ, which consists of p VQ stages. Each VQ stage is regarded as a codebook, which quantizes the input vector. The input vector x is encoded in MSVQ. Initially, $x^{(0)}$ is set to x. $x^{(0)}$ then is first quantized to generate the approximation $C^{(1)}$ by the first VQ stage. The residual vector, $x^{(1)}$, is then calculated by $x^{(1)} = x^{(0)} - C^{(1)}$. $x^{(1)}$ serves as the input to the next VQ stage. Let MSVQ consist of z_i codewords in the ith VQ stage. After x is encoded in MSVQ, the reconstructed vector, y, of x is $y = \sum_{i=1}^{p} C^{(i)}$ and the code length required for representing x is $\sum_{i=1}^{p} \log z_i$. The advantage of the p-stages MSVQ is that it is capable of uniquely representing $\prod_{i=1}^{p} z_i$ vectors with only $\sum_{i=1}^{p} z_i$ codewords required for storage.

The users must determine the number of codewords or the rate for each VQ stage before MSVQ is designed. However, the users usually have no idea of the number of codewords for each VQ stage to achieve the optimal coding of MSVQ under a total rate constraint. In Ref. [12], the pairwise nearest-neighbor (PNN) design algorithm was proposed to design the residual vector quantizers by merging the pair of stage clusters that minimizes the increase in overall distortion resulting from a given decrease in entropy. Thus, the number of codewords in each VQ stage can be automatically determined according to the rate–distortion performance. However, the PNN design algorithm only merges similar codewords from the same VQ stage, and does not measure the similarity of codewords from different VQ stages. To improve further the coding performance of MSVQ, MSVQ allows each VQ stage to comprise any desired subset of codewords in a universal codebook, namely the CSVQ with a universal codebook, which was introduced

* Tel./fax: +886 6 2957911.
E-mail address: ysb@mail.leader.edu.tw.

0031-3203/$30.00 © 2007 Pattern Recognition Society. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2007.05.011

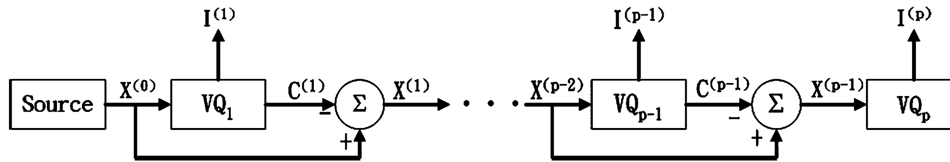

Fig. 1. The structure of the traditional MSVQ and MSVQ(GD).

in Ref. [13]. That is, if the probability density functions are sufficiently similar, then the coding performance with shared codebooks rival that using separate codebooks. In Ref. [13], the Linde–Buzo–Gray (LBG) algorithm [14] is first applied to the training data set to obtain the initial universal codebook. Starting with an initial universal codebook and an initial selector function, an iterative algorithm closely related to the LBG algorithm was used to design the universal codebook of CSVQ. In Ref. [15], MSVQ was used for the application of partial image retrieval from a huge number of images. The LBG algorithm was also applied to design the codebooks in Ref. [15]. In Ref. [16], the channel-optimized MSVQ (CO-MSVQ) was proposed to reconstruct the speech signal. They employed the modified LBG algorithm to design the CO-MSVQ encoder, and found that the perceptual quality of speech reconstructed using CO-MSVQ is better than that obtained using the channel-matched MSVQ (CM-MSVQ) proposed in Ref. [17]. In Ref. [18], a novel multipurpose image watermarking algorithm based on MSVQ. Their proposed algorithm can be applied to image authentication and copyright protection. The LBG algorithm was also used to design the VQ encoder in Ref. [18]. Lahouti [19] explored redundancy in the output of MSVQ, and presented an approximate minimum mean squared error (MMSE) technique for reconstructing MSVQ-encoded sources transmitted over a noisy channel. Similarly, the LBG algorithm was applied to design the encoders in MSVQ. However, Selim [20] showed that the LBG algorithm fails to converge to a local minimum under certain conditions. The primary drawback in CSVQ is described as follows. The users must input the rate for each source to the LBG algorithm before designing CSVQ. However, the users cannot easily determine the appropriate rate (or number of codewords) for each source when the total rate is given as a constraint in the system of Ramakrishnan [13]. Pan [21] proposed the genetic algorithms employed in designing the codebook. These genetic algorithms perform significantly better than the LBG algorithm. Begum [22] proposed a new wavelet-domain codebook design algorithm, which uses the genetic algorithm to determine the goodness of the clusters to design the codebook. This approach is similar to that of the LBG algorithm. However, Begum's algorithm [22] differs from the LBG in cluster formation. Sun [23] combined the principal component analysis and genetic algorithm to design the codebook. Experimental results show that the proposed method outperforms the LBG algorithm. Some genetic algorithms [21–23] outperform the LBG algorithm in the design of the codebook. However, the common disadvantage of these genetic algorithms is that the users must provide the codebook size before the genetic algorithms are used to design the codebook. Users generally have no idea about the optimal codebook size. A large codebook size results in a high coding rate is increased. Otherwise, the coding quality is reduced when a small codebook size is set. Thus, the codebooks designed in Refs. [21–23] are not optimal.

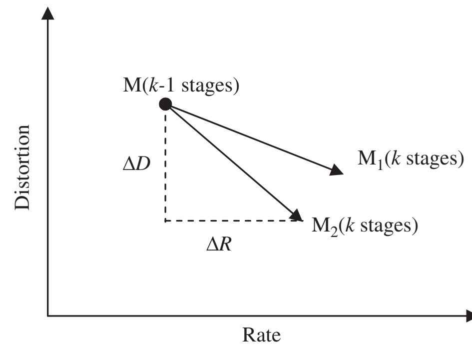

Fig. 2. The rate–distortion performance of MSVQ.

This paper makes two main contributions. First, this paper proposes MSVQ based on the genetic design (GD) algorithm, namely MSVQ(GD). The structure of the proposed MSVQ(GD) is the same as that of the traditional MSVQ. The GD algorithm is a genetic algorithm that automatically searches for the optimal number of codewords at each VQ stage according to the rate–distortion performance. Fig. 2 shows an example to illustrate the rate–distortion performance of MSVQ. Let the M be the MSVQ with $(k-1)$ stages, and the codebook at the kth VQ stage begins to be designed to produce the MSVQ with k-stages. Let M_1 and M_2 be two MSVQs with k- stages, which have different numbers of codewords at the kth VQ stage. Fig. 2 indicates that the MSVQ M_2 outperforms the MSVQ M_1 in. The reason is that the quality of M_2, has a lower distortion than that of M_1, at the same coding rate. Both the rate and distortion of MSVQ are depended on the design of codebook at each VQ stage. The users should determine the codebook size (the number of codewords) when the codebook is designed. However, the users usually cannot easily determine the number of codewords at each VQ stage of MSVQ, and then they doubt that an MSVQ model better than the M_2. In this paper, the GD algorithm maximizes that the value of $\lambda = |\Delta D/\Delta R|$ when designing the codebook at kth VQ stage. That is, the GD algorithm searches for a near-optimal codebook by maximizing the value of λ. The users need not determine the rate (or the number of codewords) for each VQ stage before the MSVQ(GD) is designed. Thus, the codebook for each VQ stage can be optimized, making it better than the codebooks

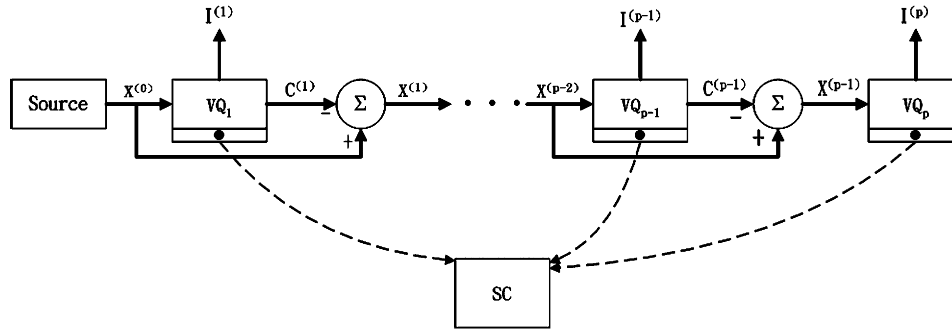

Fig. 3. The structure of the CSMSVQ.

designed in Refs. [21–23]. Additionally, the users can set the value of the parameter w to control the structure of MSVQ(GD) in the GD algorithm. If the value of parameter w is small, then MSVQ(GD) generally consists of more codewords for each VQ stage; otherwise, fewer codewords are produced in each VQ stage. Experimental results indicate that MSVQ(GD) outperforms the other traditional MSVQs. The second contribution of this paper is to develop the constrained-storage MSVQ, namely CSMSVQ, using a sharing codebook. Fig. 3 illustrates the structure of CSMSVQ, in which all VQ stages share the codewords contained in the sharing codebook. The genetic merge (GM) algorithm is a genetic algorithm that automatically generates the sharing codebook under the storage constraint, in which case the users need not define the sharing codebook size in CSMSVQ. That is, the GM algorithm can optimize the sharing codebook.

The remainder of this paper is organized as follows. Section 2 presents the design of the MSVQ(GD). Section 3 then describes the design of the CSMSVQ using the GM algorithm. Subsequently, experiments are given in Section 4. Conclusions are finally made in Section 5.

2. Design of MSVQ based on the GD algorithm

Section 2.1 describes the GD algorithm, while Section 2.2 presents the design of the MSVQ(GD).

2.1. GD algorithm

In the proposed MSVQ(GD), the GD algorithm replaces the traditional LBG algorithm to design the codebook for each VQ stage. The GD algorithm can automatically search the proper number of codewords as the codebook in each VQ stage according to the rate–distortion performance of MSVQ(GD). Also, the number of codewords in each VQ stage of MSVQ(GD) is different, and the code of each VQ stage relies on the Huffman coding [24].

Let $L=\{x_1, x_2, \ldots, x_n\}$ denote the set of training data objects for designing the MSVQ(GD). Suppose that L is a large data set. To perform GD for processing the large data set, the PNN algorithm [25] is first applied to the set L. In PNN, the two closed objects can be merged at a time to form a new cluster, and this merge processing is continued until the desired number of clusters is obtained. Let the set $L' = \{B_1, B_2, \ldots, B_m\}$ with m clusters, be obtained after the PNN is applied to L, and let V_i indicate the center of cluster B_i, for $1 \leqslant i \leqslant m$. Each cluster B_i is considered a component and is not divided during the GD algorithm. That is, only m ($m \ll n$) components must be further processed in GD. The PNN algorithm is used to reduce the computation time in GD. Therefore, the GD algorithm can efficiently process the large data set.

Before describing the design of the GD algorithm for the p-stages MSVQ(GD), this study first defines the distortion of k-stages MSVQ(GD), D(k-stages MSVQ(GD)), and the rate, R(k-stages MSVQ(GD)), for $1 \leqslant k \leqslant p$ as follows. Let $B_i^{(k-1)}$ denote the residual vector obtained after the component B_i is quantized by $k-1$ VQ stages in MSVQ(GD). $B_i^{(k-1)}$ thus is considered the input vector to the kth VQ stage in MSVQ(GD). Moreover, let the codebook in the kth VQ stage contain z_k codewords, $C_1^{(k)}, C_2^{(k)}, \ldots, C_{z_k}^{(k)}$. Each codeword $C_j^{(k)}$ is regarded as the center of cluster C_j, for $1 \leqslant j \leqslant z_k$. Then, $B_i^{(k-1)}$ searches the closest codeword, $C_q^{(k)}$, as the coding result in the kth VQ stage, where $q = \arg\min_{1 \leqslant j \leqslant z_k} \|B_i^{(k-1)} - C_j^{(k)}\|$. Initially, we set $B_i^{(0)} = V_i$, for $1 \leqslant i \leqslant m$. Then,

$$B_i^{(k)} = B_i^{(k-1)} - C_q^{(k)} \quad \text{for } 1 \leqslant k \leqslant p \text{ and } 1 \leqslant i \leqslant m. \tag{1}$$

The residual vector, $B_i^{(k)}$, serves as the input to the $(k+1)$th VQ stage of MSVQ(GD). $code(B_i^{(k-1)})$ then is defined for $1 \leqslant k \leqslant p$, as follows:

$$code(B_i^{(k-1)}) = \text{The code of } C_q^{(k)} \text{in the codebook of the } k\text{th VQ stage.} \tag{2}$$

Therefore, D(k-stages MSVQ(GD)) and R(k-stages MSVQ(GD)) are defined as follows:

$$D(k\text{-stages MSVQ(GD)}) = \sum_{j=1}^{z_k} \sum_{B_i \in C_j} \|V_i^- C_j^{(k)}\| |B_i|, \tag{3}$$

where $|B_i|$ indicates the number of objects contained in the component B_i.

$$R(k\text{-stages MSVQ(GD)}) = \sum_{i=1}^{m} |code(B_i^{(k-1)})| |B_i|, \tag{4}$$

where $|code(B_i^{(k-1)})|$ is denoted as the length of $code(B_i^{(k-1)})$ defined as Eq. (2). Under the conditions of quasi-convexity, the slope of the rate–distortion function for rates between that of $(k-1)$-stages MSVQ(GD) and k-stages MSVQ(GD), λ_k, is

$$\lambda_k = \frac{\Delta D}{\Delta R} = \frac{D((k\text{-}1)\text{-stages MSVQ(GD)}) - D(k\text{-stages MSVQ(GD)})}{R(k\text{-stages MSVQ(GD)}) - R((k-1)\text{-stages MSVQ(GD)})} \quad \text{for } 1 \leqslant k \leqslant p. \tag{5}$$

This paper emphasizes that the kth VQ stage of MSVQ(GD) is designed to maximize the value of the rate–distortion performance, λ_k. Furthermore, users can control the size of the codebook in the kth VQ stage of MSVQ(GD). If ΔR in Eq. (5) is maximized, the training data set at any one time is classified into many codewords. That is, the MSVQ(GD) with fewer stages can be designed at the certain bit rate. Also, if ΔR is minimized, then the training data set is classified into few codewords, and the GD algorithm tends to produce a MSVQ(GD) with more stages at the same bit rate. Therefore, λ_k in Eq. (5) can be rewritten as

$$\lambda'_k = \frac{\Delta D}{(\Delta R)^w} = \frac{D((k-1)\text{-stages MSVQ(GD)}) - D(k\text{-stages MSVQ(GD)})}{[R(k\text{-stages MSVQ(GD)}) - R((k-1)\text{-stages MSVQ(GD)})]^w}, \tag{6}$$

where w denotes a weighting factor. Notably, w is positive in Eq. (16). If w is larger than 1, then the codebook containing few codewords can be obtained. Moreover, if w is within (0, 1), the codebook containing many codewords is produced.

Let MSVQ(GD) with $(k-1)$-stages have been designed. The following only describes how to design the kth stage of MSVQ(GD) because the GD algorithm designs one VQ stage at a time in MSVQ(GD). The main goal of the GD algorithm is to search for the proper number of codewords as the codebook, which enhances the value of λ'_k when the kth VQ stage of MSVQ(GD) is designed by the GD algorithm. The GD algorithm is designed using the genetic approach, which consists of an initialization step and iterations with three phases in each generation. Fig. 4 shows the basic structure of the genetic approach. The basic structure is described in the following.

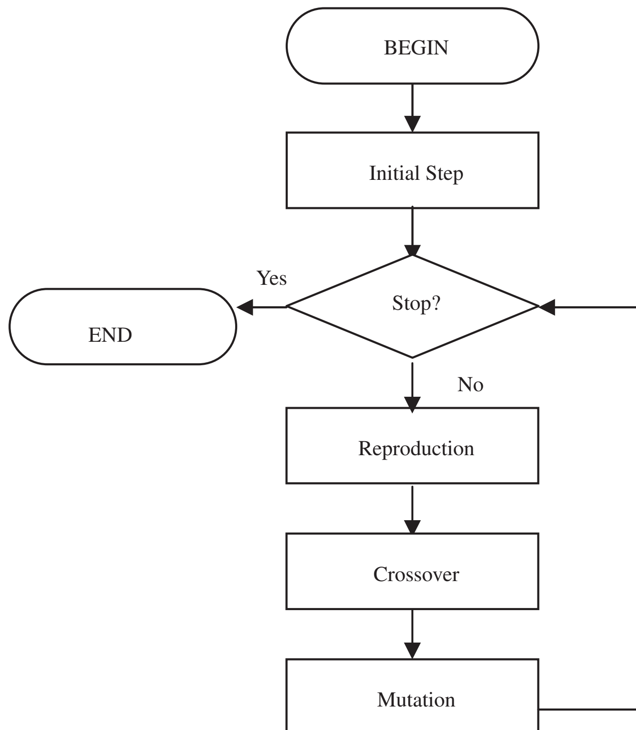

Fig. 4. The basic processing of the genetic algorithm.

2.1.1. Initialization step

A population of N strings is randomly generated in the initialization step of GD. The length of each string is m, which is the number of components obtained in the PNN algorithm. Moreover, N strings are generated such that the 1s in the strings are uniformly distributed within $[m, 1]$. Each string represents a subset of $\{B_1, B_2, \ldots, B_m\}$. If this subset contains B_i, then the ith position of the string will be 1; otherwise, it will be 0. Each B_i in the subset is a seed for generating a cluster.

The method for generating a clustering from the seeds is described before elucidating the three phrases. Let $R = (b_1, b_2, \ldots, b_m)$ be a bit string in the population. Each bit b_i indicates the corresponding component B_i. The string R then includes two sets of components, L_1 and L_2, which are defined as

$$L_1 = \{B_i | b_i = 1, 1 \leqslant i \leqslant n_1\}, \tag{7}$$

$$L_2 = \{B'_j | b_j = 0, 1 \leqslant j \leqslant n_2\}, \tag{8}$$

where $n_1 + n_2 = m$ and $L' = L_1 \cup L_2$. In L_1, n_1 components, B_i for $1 \leqslant i \leqslant n_1$, serve as the seeds to generate n_1 clusters. Initially, each cluster C_i contains just one component, B_i, and then the center S_i of cluster C_i is set to V_i. The components in L_2 subsequently are considered individually and the Euclidean distances between each component and the centers S_i, for $1 \leqslant i \leqslant n_1$, are calculated. Then,

$$B'_j \subset C_i \quad \text{if } \|V'_j - S_i\| \leqslant \|V'_j - S_l\| \quad \text{for } 1 \leqslant l \leqslant n_1. \tag{9}$$

If B'_j is classified into the cluster C_i to form the new cluster $\hat{C}_i$, then the new center $\hat{S}_i$ and the size of cluster $\hat{C}_i$ are updated as

$$\hat{S}_i = \frac{S_i|C_i| + V'_j|B'_j|}{|C_i| + |B'_j|}, \quad |\hat{C}_i| = |C_i| + |B'_j|, \tag{10}$$

where $|C_i|$ and $|B'_j|$ indicate the number of objects contained in the cluster C_i and the component B'_j, respectively. After

considering all of the components in L_2, n_1 clusters, C_i for $1 \leqslant i \leqslant n_1$, are obtained from the string R.

2.1.2. Reproduction phase

The main issue of the reproduction phase is to design the fitness function of the string R. Let the string R generate n_1 clusters, C_i for $1 \leqslant i \leqslant n_1$. The center S_i of each cluster C_i is considered the codeword in the codebook. That is, n_1 codewords are generated to produce the codebook in the kth VQ stage of MSVQ(GD). Thus, $C_i^{(k)} = S_i$ is set for $1 \leqslant i \leqslant z_k$, where $z_k = n_1$. These z_k codewords are coded by the Huffman codes based on the number of objects contained in each cluster C_i. λ'_k is calculated when the codebook in the kth VQ stage is generated by the string R. If λ'_k is large, then the fitness of string R also tends to be large; otherwise, the fitness of string R is small. Thus, the fitness function of string R is defined as

$$Fitness(R) = \lambda'_k. \tag{11}$$

After calculating the fitness of each string in the population, the reproduction operator is implemented using a roulette wheel with slots sized according to fitness.

2.1.3. Crossover phase

If the crossover operator is applied to a selected pair of strings R and Q, then two random numbers e and f in $[1, m]$ are generated to determine which pieces of the strings are to be interchanged. After the crossover phase, two new strings, R' and Q', replace the strings, R and Q, in the population. The significance of the crossover phase is that it exchanges seeds between the different strings, to yield the various clusterings.

2.1.4. Mutation phase

During the mutation phase, the bits of the strings in the population are chosen from $[1, m]$ with probability P_m. Each chosen bit is then changed from 0 to 1 or from 1 to 0. That is, if one bit is chosen, then a selected cluster is discarded or produced in a string. After the mutation phase, the new string R' can be obtained and the original string R can be replaced.

The user may specify the number of generations over which they want the GD algorithm to run. The GD algorithm runs for this number of generations and retains the best fitting string.

The following analyzes the time complexity of GD when the kth VQ stage of MSVQ(GD) is designed. The GD algorithm consists of an initialization step and iterations with three phases in each generation. The number of training components is set to m, and N denotes the population size. Moreover, z denotes the average codebook size of these $(k-1)$ VQ stages in MSVQ(GD). The time complexity of the GD algorithm is dominated by the calculation of the fitness function. N strings take $\mathbf{O}(N(m^2 + mkz))$ time to design the kth VQ stage of MSVQ(GD). Suppose the GD algorithm is asked to run G generations, the time complexity of GD is $\mathbf{O}(GN(m^2 + mkz)) \fallingdotseq \mathbf{O}(GNm^2)$.

2.2. Design of the proposed MSVQ(GD)

The following describes the design algorithm for MSVQ(GD), as follows.

Algorithm: Design_MSVQ(GD)

Input: The distortion threshold d and the set of training objects, $L = \{x_1, x_2, \ldots, x_n\}$.

Output: MSVQ(GD) under the distortion constraint.

Step 1. The PNN algorithm is applied to the set L, and then the set L' with m components, $\{B_1, B_2, \ldots, B_m\}$, is obtained. Let $B_i^{(k-1)}$ represent the residual vector for the input to the kth VQ stage. Moreover, set $B_i^{(0)} = V_i$, for $1 \leqslant i \leqslant m$, where V_i is the center of B_i. Set $k = 1$.

Step2. **While** L' is not an empty set.

Step 2.1. The GD algorithm is applied to the set L', and then z_k clusters, $C_1, C_2, \ldots, C_{z_k}$, are obtained. Let S_j be the center of cluster C_j, and let $C_j^{(k)}$ denote the codeword in the codebook of the kth VQ stage. Then, set $C_j^{(k)} = S_j$, for $1 \leqslant j \leqslant z_k$.

Step 2.2. Let H represent the collection of the codewords, in which the distortion of each codeword exceeds d. Set $H = \{C_j^{(k)} | \sum_{i=1, B_i \in C_j}^{m} \|V_i - S_j\| |B_i| / |C_j| > d, 1 \leqslant j \leqslant z_k\}$. Set $L'' = \phi$.

Step 2.3 For each vector, $B_i^{(k-1)}$, $1 \leqslant i \leqslant m$, do the following.
If $C_q^{(k)} \in H$, where $q = \arg\min_{1 \leqslant j \leqslant z_k} \|B_i^{(k-1)} - C_j^{(k)}\|$, then calculate $B_i^{(k)} = B_i^{(k-1)} - C_q^{(k)}$ and set $L'' = L'' \cup \{B_i^{(k)}\}$.

Step 2.4 Set $L' = L''$ and $k = k + 1$.

Step 3. Output the k-stages MSVQ(GD). Stop.

Two advantages of the proposed MSVQ(GD) compared to the traditional MSVQs are as follows. First, the GD algorithm can automatically search for the proper number of codewords which are regarded as the codebook in each VQ stage according to the rate–distortion performance of MSVQ(GD). Users need not artificially determine the codebook size in each VQ stage when the MSVQ(GD) is designed. Second, the weight, w, in the GD algorithm is given to the users to design the MSVQ(GD). If w is larger than 1, then the codebook containing a small number of codewords can be obtained in the VQ stage. Restated, the MSVQ(GD) with many VQ stages is produced. If w is within (0, 1), the codebook containing numerous codewords can be produced in the VQ stage. That is, the MSVQ(GD) with few VQ stages is produced. The experiments in Section 4 have shown that the MSVQ(GD) outperforms other traditional MSVQs based on the LBG algorithm given the same bit rate.

3. Design of CSMSVQ using a sharing codebook

This section proposes CSMSVQ using a sharing codebook to reduce the storage size. The codewords in the sharing

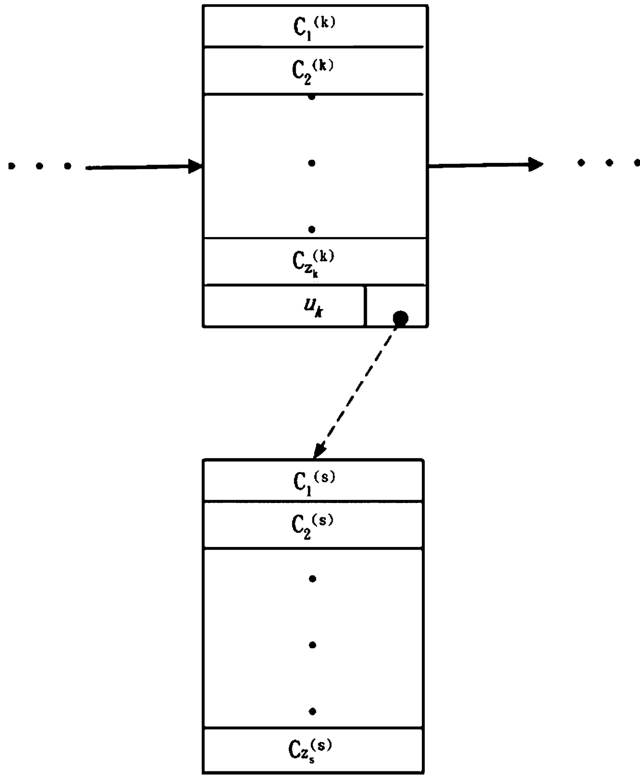

Fig. 5. The sharing codebook and the kth VQ stage in CSMSVQ.

codebook are obtained by merging similar codewords in the VQ stages. All VQ stages then can share the codewords in the sharing codebook. The structure of CSMSVQ is described in Section 3.1. Section 3.2 then indicates the design of the sharing codebook by the GM algorithm, and Section 3.3 presents the design of the CSMSVQ.

3.1. Structure of CSMSVQ

Fig. 5 details the p-stages CSMSVQ, which is described as follows. Let the codebook in the kth VQ stage of CSMSVQ contain z_k codewords, $C_1^{(k)}, C_2^{(k)}, \ldots, C_{z_k}^{(k)}$, for $1 \leqslant k \leqslant p$, and let the sharing codebook contain z_s codewords, $C_1^{(s)}, C_2^{(s)}, \ldots, C_{z_s}^{(s)}$. The codebook of each VQ stage contains a pointer indicating the sharing codebook. In the sharing codebook, the selector function, u_k, specifies a specific subset of sharing codewords for the kth VQ stage. If the sharing codeword $C_j^{(s)}$ is contained in the subset for the kth VQ stage, $u_k(j)$ is set to 1; otherwise $u_k(j)$ is set to 0. That is, when aninput vector is quantized by the kth VQ stage of CSMSVQ, the input vector compares with the codewords in the set, $S^{(k)} = \{C_i^{(k)} | 1 \leqslant i \leqslant z_k\} \cup \{C_j^{(s)} | u_k(j) = 1, 1 \leqslant j \leqslant z_s\}$.

The following describes how to define the code of each codeword in CSMSVQ. Let $N(C_i^{(k)})$ indicate the number of training vectors, which are the closest to the codeword, $C_i^{(k)}$, for $1 \leqslant i \leqslant z_k$. Let $N(C_j^{(s)})$ indicate the number of training vectors, which are the closest to the sharing codeword, $C_j^{(s)}$, for $1 \leqslant j \leqslant z_s$. Thus, $N(C_j^{(s)}) = \sum_{k=1, u_k(j)=1}^{p} N_k(C_j^{(s)})$, where $N_k(C_j^{(s)})$ indicate the number of training vectors, which are the closest to the sharing codeword, $C_j^{(s)}$, when the training vectors are quantized by the kth VQ stage. Then, these codewords in $S^{(k)}$are coded by the Huffman codes for the kth VQ stage based on these numbers, $N(C_i^{(k)})$ for $1 \leqslant i \leqslant z_k$ and $N_k(C_j^{(s)})$ for $1 \leqslant j \leqslant z_s$.

3.2. GM algorithm

MSVQ(GD) described in Section 2 can be produced before designing the CSMSVQ. The GM algorithm then searches the similar codewords contained in these VQ stages of MSVQ(GD) that need to be merged into the codewords in the sharing codebook. That is, these similar codewords in the VQ stages can then be discarded after being merged into a new codeword in the sharing codebook. The merging process is continued until the desired storage of CSMSVQ is reached. Although merging many codewords into one reduces the storage size, it also increases the distortion of CSMSVQ. The main design issue of the GM algorithm is to automatically determine the VQ stages where the codewords can be merged into a codeword in the sharing codebook, thus maximizing the storage–distortion performance of CSMSVQ. Before describing the design of the GM algorithm, the storage of p-stages CSMSVQ with the sharing codebook size z_s, S(p-stages CSMSVQ with the sharing codebook size z_s), and the distortion, D(p-stages CSMSVQ with the sharing codebook size z_s), are defined as follows. Let $B_i^{(k-1)}$ denote the residual vector to the kth VQ stage of CSMSVQ. Then, $B_i^{(k-1)}$ searches the closest codeword $\hat{C}$ in $S^{(k)}$. We calculate

$$B_i^{(k)} = B_i^{(k-1)} - \hat{C} \quad \text{for } 1 \leqslant i \leqslant m \text{ and } 1 \leqslant k \leqslant p. \tag{12}$$

Therefore, D(p-stages CSMSVQ with the sharing codebook size z_s) and S(p-stages CSMSVQ with the sharing codebook size z_s) are defined as follows:

$$D(p\text{-stages CSMSVQ with the sharing codebook size } z_s) = \sum_{k=1}^{p} \sum_{j=1}^{z_k} \sum_{B_i \in C_j} \| V_i^{-} C_j^{(k)} \| |B_i| + \sum_{j=1}^{z_s} \sum_{B_i \in C'_j} \| V_i^{-} C_j^{(s)} \| |B_i|, \tag{13}$$

where $C_j^{(k)}$ and $C_j^{(s)}$ are denoted as the center of clusters C_j and C'_j, respectively.

$$S(p\text{-stages CSMSVQ with the sharing codebook size } z_s) = z_s + \sum_{i=1}^{p} z_i. \tag{14}$$

Let MSVQ(GD) consist of p VQ stages, and let the codebook in the kth VQ stage of MSVQ(GD) contain z_k codewords, $C_1^{(k)}, C_2^{(k)}, \ldots, C_{z_k}^{(k)}$, for $1 \leqslant k \leqslant p$. The GM algorithm using the genetic approach to design the sharing codebook of CSMSVQ is described as follows.

3.2.1. Initialization step

A population of N strings is randomly generated during the initialization step of GM. The length of each string is set to $e=\sum_{i=1}^{p} z_k$, where e denotes the total number of codewords contained in all VQ stages of MSVQ(GD). N strings are generated such that the 1s in the strings are uniformly distributed within $[1,e]$. Each string represents a subset of $H=\{C_1^{(1)},C_2^{(1)},\ldots,C_{z_k}^{(1)},\ldots,C_1^{(p)},C_2^{(p)},\ldots,C_{z_k}^{(p)}\}$. If C_i^k is in this subset, then the corresponding position of the string will be 1; otherwise, it will be 0. All C_i^k in the subset can be merged into a codeword in the sharing codebook. For example, the string $R=(1,0,1,0,\ldots,0)$ indicates that both codewords, $C_1^{(1)}$ and $C_3^{(1)}$, can be merged into a codeword in the sharing codebook.

3.2.2. Reproduction phase

The following describes how to define the fitness function of the string R. Let $T=\{C_j^{(k)}|$the corresponding bit in R is 1, $1\leqslant j\leqslant z_k, 1\leqslant k\leqslant p\}$ be the collection of codewords whose corresponding bits equal 1 in R. These codewords in T then are merged into a new codeword, $C^{(s)}$, for the string R. Then,

$$C^{(s)}=\frac{\sum_{C_j^{(k)}\in T} C_j^{(k)} N(C_j^{(k)})}{\sum_{C_j^{(k)}\in T} N(C_j^{(k)})}. \quad (15)$$

That is, the string R produces a codeword in the sharing codebook, and then CSMSVQ with the sharing codebook size 1 is obtained. Thus, the fitness function for the string R is defined as

$$Fitness(R)=\frac{\Delta S}{\Delta D} = \frac{S(p\text{-stages MSVQ(GD)})-S(p\text{-stages CSMSVQ with the sharing codebook size 1})}{D(p\text{-stages CSMSVQ with sharing codebook size 1})-D(p\text{-stages MSVQ(GD)})}, \quad (16)$$

where $D(p$-stages MSVQ(GD)) and $S(p$-stages MSVQ(GD)) indicate the distortion and storage of MSVQ(GD), respectively. According to the storage–distortion performance, the fitness of the string is maximized in the GM algorithm. This phenomenon occurs because the string with high fitness indicates that the codewords represented by the string are appropriate for merging into a codeword in the sharing codebook.

3.2.3. Crossover and mutation phases

The crossover and mutation phases in GM are similar to that in GD. If a pair of strings R and Q is chosen for applying the crossover operator, two random numbers in $[1,e]$ are generated to determine which pieces of the strings are to be interchanged. Also, the crossover operator is done with probability P_c. Furthermore, in the mutation phase, the bits of each string R in the population are chosen from $[1,e]$ with probability P_m. Each chosen bit is then changed from 0 to 1 or from 1 to 0. Following the mutation phase, the new string R' can be obtained and the original string R can be replaced.

The objective of the GM algorithm is not to find the string with the best fitness, but rather a set of strings with good fitness. Each string represents that some of $C_j^{(k)}$, for $1\leqslant j\leqslant z_k, 1\leqslant k\leqslant p$, can be merged into a sharing codeword. Thus, the set of strings with good fitness produces a good sharing codebook. The next section describes how to design the codewords in the sharing codebook under the storage constraint given by the users after the completion of the GM algorithm.

The following analyzes the time complexity of the GM algorithm when the sharing codebook is designed in the p-stages CSMSVQ. The number of training components is m, and N denotes the population size. Moreover, z denotes the average codebook size of these p VQ stages of CSMSVQ. The time complexity of the GM algorithm then is dominated by the calculation of the fitness function. Each string takes $\mathbf{O}(mpz)$ to calculate the fitness function in the reproduction phase. If the GM algorithm is asked to run G generations, then the time complexity of GM is $\mathbf{O}(GNmpz)$.

3.3. Design of the CSMSVQ

Completing the GM algorithm obtains the N bit strings with fitness. Each bit string represents a subset of H. That is, the codewords represented by a bit string can be merged into a new codeword in the sharing codebook. The problem is how to find the subset of N bit strings to generate the sharing codebook of CSMSVQ under the storage constraint. The Design_SMSVQ algorithm is described as follows.

Algorithm: Design_CSMSVQ

Input: p-stages MSVQ(GD) and the storage threshold M.
Output: p-stages CSMSVQ with the sharing codebook size z_s.

Step 1. Let the codebook in the kth VQ stage of MSVQ(GD) contain z_k codewords for $1\leqslant k\leqslant p$. Set the structure of CSMSVQ to be the same as that of MSVQ, and set $z_s=0$.

Step 2. The GM algorithm is applied to all codewords contained in these p VQ stages of MSVQ(GD). Then, N bit strings with the fitness can be obtained. Sort the fitness of the strings in non-increasing order. For brevity, assume $Fitness(R_1)\geqslant Fitness(R_2)\geqslant\cdots\geqslant Fitness(R_N)$. Let $T_i=\{C_j^{(k)}|$the corresponding bit in R_i is 1, $1\leqslant j\leqslant z_k, 1\leqslant k\leqslant p\}$ be the collection of codewords, the corresponding bits of which equal 1 in R_i. Let the set U contain the codewords in the sharing codebook. Set $i=1$, $U=\phi$ and $T'=\phi$.

Step 3. **While** (The storage size of CSMSVQ with the sharing codebook size z_s.) $>M$

Step 3.1. Choose R_i. The codewords in T_i then are merged into a new codeword $C^{(s)}$, as shown in Eq. (15).

Step 3.2. Set $U=U\cup C^{(s)}$, $T'=T'\cup T_i$ and $z_s=z_s+1$.

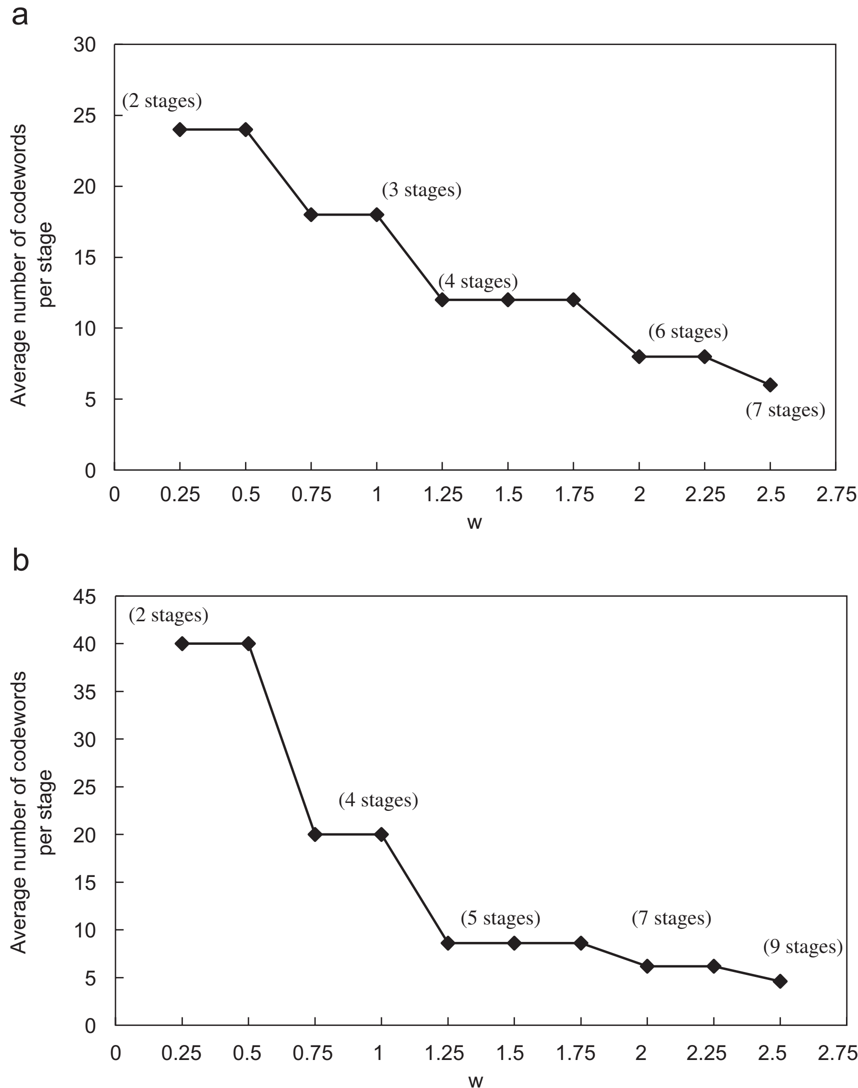

Fig. 6. The relation between the values of w and the average number of codewords for each VQ stage in MSVQ(GD). (a) Image data. (b) Speech data.

Step 3.3. For each codeword $C_j^{(k)}$ in T_i do the following:
Discard the codeword $C_j^{(k)}$ in the kth VQ stage of CSMSVQ. Set $z_k = z_k - 1$.

Step 3.4. Choose the smallest $b > i$ such that $T_b \cap T' = \phi$. Then, set $i = b$.

Step 4. Output the p-stages CSMSVQ with the sharing codebook size z_s. End.

An example to illustrate the Design_CSMSVQ algorithm is given as follows. Suppose $Fitness(R_1)Fitness(R_2) \geqslant \cdots \geqslant Fitness(R_N)$, where R_1 represents subset $\{C_1, C_2, C_3\}$, R_2 represents subset $\{C_3, C_4, C_6\}$, R_3 represents subset $\{C_4, C_5\}$, and each of R_4 to R_N represents a subset containing at least one of C_1, C_2, C_3, C_4 and C_5. In this algorithm, by first choosing R_1, codewords C_1, C_2, C_3, are merged into the first codeword of the sharing codebook. Since the subset represented by R_2 contains C_3 that is already in the subset represented by R_1, R_2 is discarded. After that R_3 is considered, the codewords, C_4 and C_5, in this subset are merged, and then the second codeword is obtained in the sharing codebook. In Design_CSMSVQ, the merging processing is continued until the desired storage of CSMSVQ is achieved.

The Design_CSMSVQ algorithm employs the greedy strategy to search the set of bit strings to design the sharing codebook. That is, the strings are selected based on order of fitness values. However, the dynamic programming strategy can also be applied to these N bit strings to identify the set of bit strings under the storage constraint. Although the sharing codebook produced by the dynamic programming strategy is better than that produced by the greedy strategy, the time complexity of the dynamic programming strategy is high. In the present experiments, the coding quality obtained using the dynamic programming strategy compared to the greedy strategy is well below 0.005 dB in PSNR. Thus, the greedy strategy is sufficient in the Design_CSMSVQ algorithm.

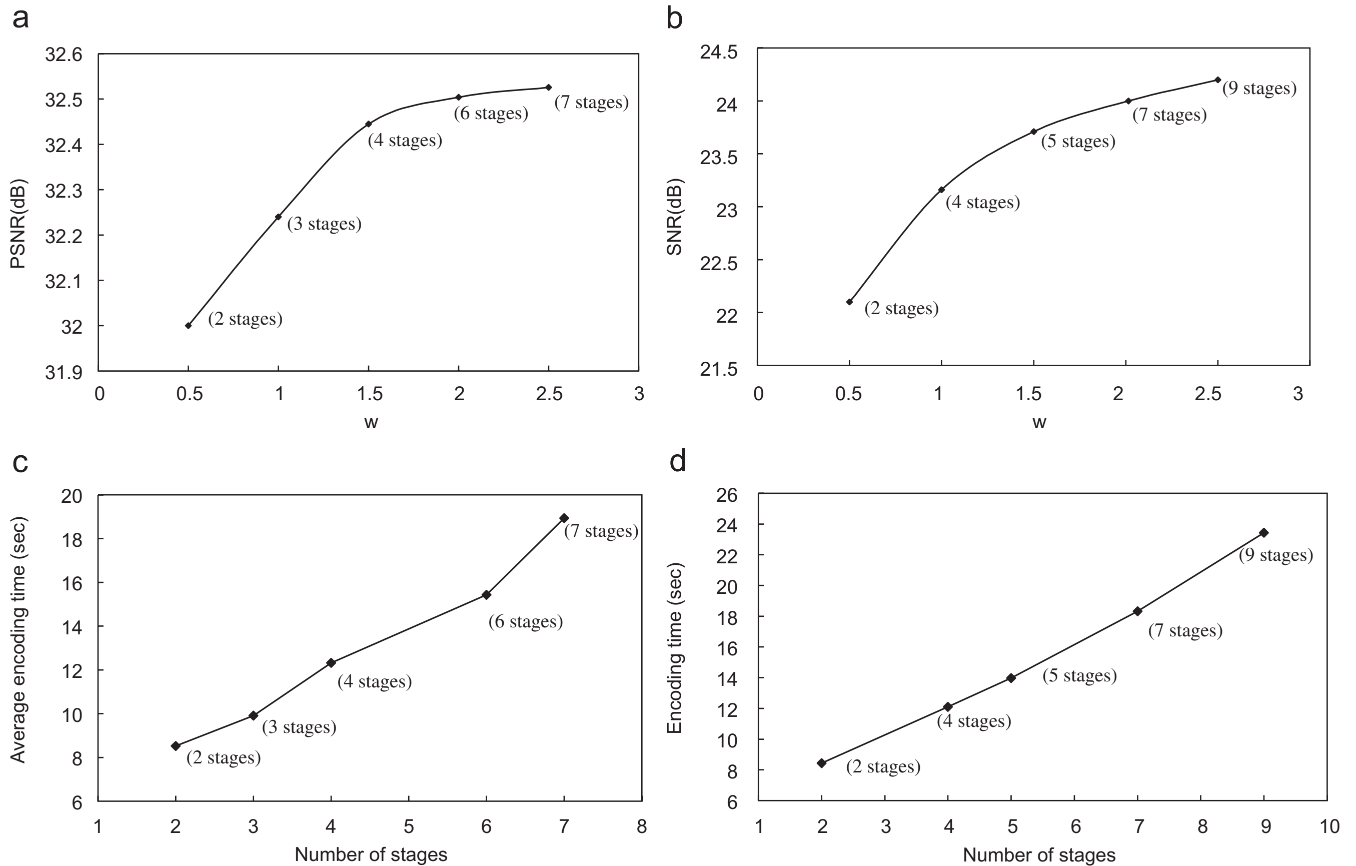

Fig. 7. The relation between the values of w and the performance of MSVQ(GD). (a) The coding quality for image data set. (b) The coding quality for speech data set. (c) Average encoding time for image data set. (d) Average encoding time for speech data set.

4. Experiments

This section compares the performance of MSVQ(GD) with that of the other MSVQs. Furthermore, the proposed CSMSVQ and CSVQ are compared below. Section 4.1 describes the data sets used in the experiments. Moreover, Sections 4.2 and 4.3 show the performance of MSVQ(GD) and CSMSVQ, respectively.

4.1. Data sets

The experiments use two data sets, namely speech and image data sets, to test the performance of MSVQ(GD) and CSMSVQ, respectively. In the image data set, five 512×512 (pixels) images with 256 gray levels were employed as the training images, and were divided into 4×4 blocks to design the MSVQ(GD) and the other MSVQs. The "Lena", "Boat" and "F-16" images were not used in training to test the performance of these methods. In the speech data set, a total of 100 000 spectral feature vectors of speech taken from the TIMIT [26] database were used as training objects for designing the MSVQ(GD) and the other methods. These spectral feature vectors were taken from 1000 continuous speeches, given by five males and five females. Each spectral feature vector contained 64 sample points. Furthermore, 50 000 spectral feature vectors were extracted as testing objects that were not used in training.

4.2. Performance of MSVQ(GD)

The parameters used in GD were population size 300, crossover rate 80% and mutation rate 5%. Five hundred generations were run, and the best solution was retained in each generation. Before GD was applied to the training data set, the PNN algorithm was applied to all training objects to obtain a small set of components. In the experiments, the size of the component set was set to a fifth of the total number of training objects. In GD, the weight, w, is used to control the number of codewords for each codebook in MSVQ(GD). Fig. 6 shows the average number of codewords in the codebook of each VQ stage in MSVQ(GD) under the distortion threshold ($d = 40$ for image data, $d = 60$ for speech data) when various values of w are used in Eq. (6). In Fig. 6, the value contained in the bracket indicates the number of VQ stages in MSVQ(GD). If w is large, then the MSVQ(GD) with many stages is obtained; otherwise, the MSVQ(GD) with few stages is obtained. A simple heuristic method is given below for users to determine the number of stages of MSVQ(GD). In Fig. 6(a), MSVQ(GD)

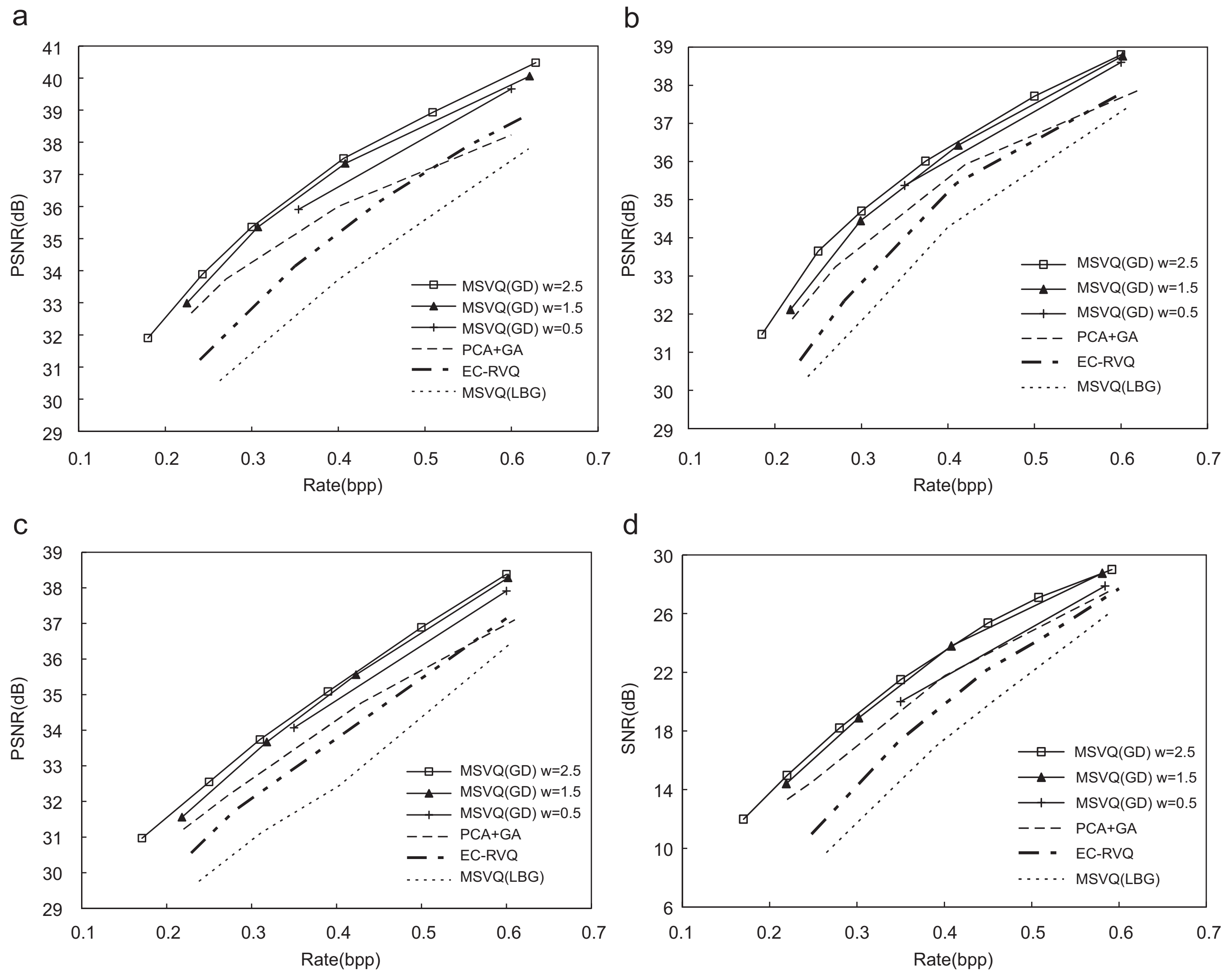

Fig. 8. Comparison of the performance of MSVQ(GD), MSVQ(LBG), EC-RVQ and PCA + GA. (a) "Lena" image. (b) "Boat" image. (c) "F-16" image. (d) Speech data.

with four stages is better than the other MSVQ(GD)s with different number of stages, because the MSVQ(GD) with four stages is more stable when w is within a range, [1.25, 1.75], that is larger than the other ranges, [0.25, 0.5], [0.75, 1] and [2, 2.25]. Furthermore, MSVQ(GD) with five stages is chosen by the heuristic method in Fig. 6(b). Figs. 7(a)and (b) show that increasing w improves the coding quality. This occurs because the GD algorithm tends to grow into an MSVQ(GD) containing numerous VQ stages with increasing w, and because the number of vectors represented by the codewords in the MSVQ(GD) with numerous VQ stages is larger than that represented by the codewords in the MSVQ(GD) with few VQ stages. Figs. 7(c) and (d) also show that the encoding time of MSVQ(GD) increases with increasing numbers of stages. However, increasing the number of VQ stages also enhances the coding quality of MSVQ(GD).

In the present experiments, the values of w in the GD algorithm were set to 0.5, 1.5 and 2.5 to test the performance of MSVQ(GD), respectively. Fig. 8 compares MSVQ(GD) based on the GD algorithm, MSVQ(LBG) based on the LBG algorithm, EC-RVQ [10] and Sun's method [23]. In Fig. 8, "PCA + GA" denotes Sun's method [23]. The GD algorithm was first used to design the MSVQ(GD), and MSVQ(LBG), EC-RVQ and PCA + GA were then applied to obtain equal-sized codebooks for each VQ stage. That is, MSVQ(GD), MSVQ(LBG), EC-RVQ and PCA + GA have the same number of VQ stages and the same number of codewords in each VQ stage. Fig. 8 demonstrates MSVQ(GD) outperforms MSVQ(LBG), EC-RVQ and PCA + GA. Additionally, Fig. 8 reveals two phenomena. First, the GD algorithm designs the codewords in each codebook of the VQ stage according to the rate–distortion performance of MSVQ(GD), while the PCA + GA does not. Thus, the MSVQ designed by the GD algorithm has a better rate–distortion performance than that designed by PCA+GA. Additionally, the GD algorithm can automatically search the proper number of codewords for each VQ stage in MSVQ(GD). Thus, the coding quality of MSVQ(GD) tends to achieve the near-optimal coding. Furthermore,

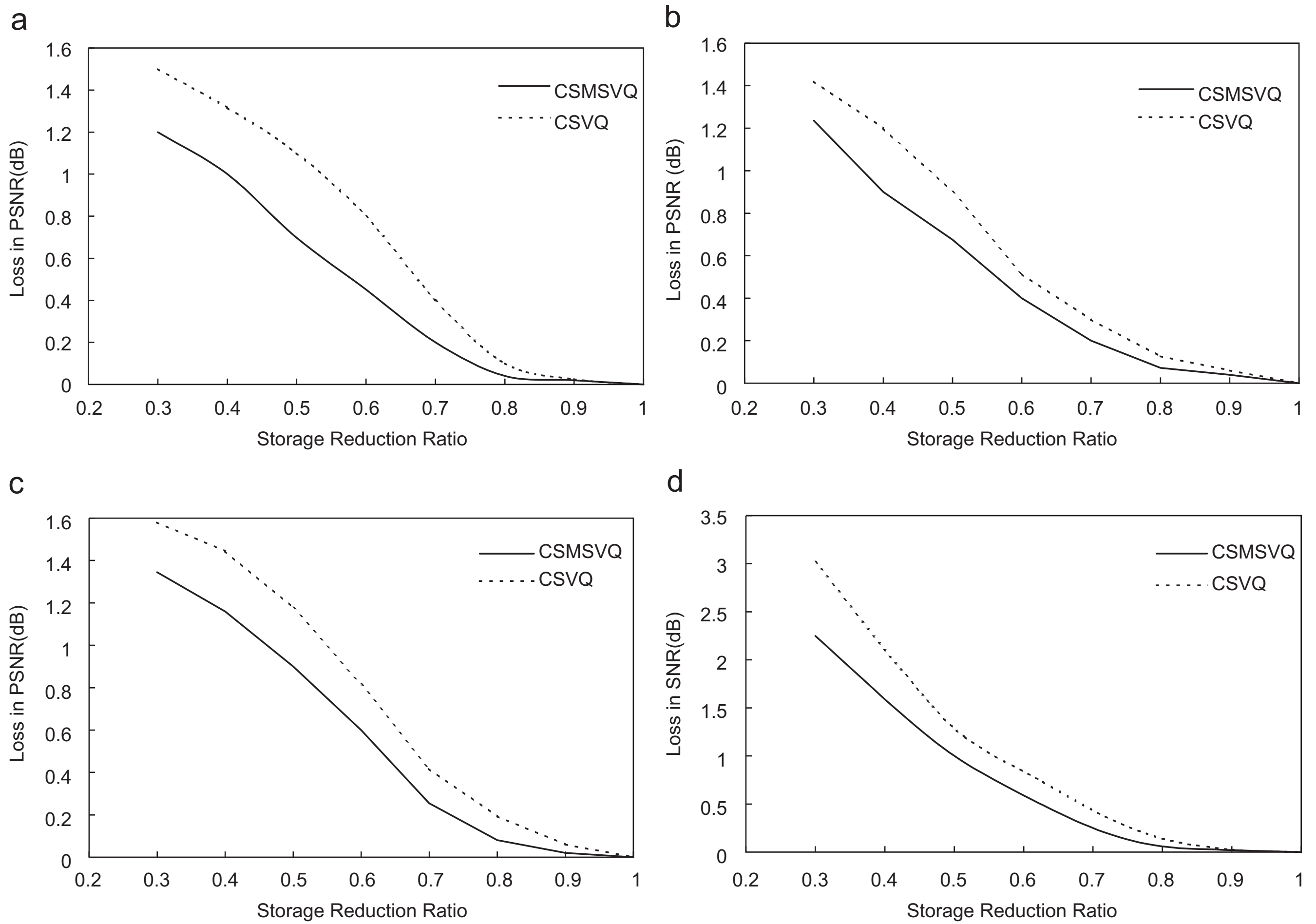

Fig. 9. Comparison of the performance of CSMSVQ and CSVQ. (a) "Lena" image. (b) "Boat" image. (c) "F-16" image. (d) Speech data.

the GD algorithm based on the genetic approach finds the near-optimal solution, while the LBG and the EC-RVQ design algorithms usually only finds a locally optimal solution.

4.3. Performance of CSMSVQ

The MSVQ(GD) can be produced before designing the CSMSVQ. The GM algorithm searches the similar codewords in the codebooks of MSVQ(GD) to design a sharing codebook, and then the Design_CSMSVQ algorithm produces the CSMSVQ under the storage constraint. Fig. 9 compares CSMSVQ and CSVQ proposed in Ref. [13]. To fairly compare CSMSVQ and CSVQ, p codebooks in MSVQ(GD) are used as p sources to design the universal codebook for CSVQ . That is, the GM algorithm first was used to design the CSMSVQ, and then the CSVQ was applied to obtain the universal codebook with equal rate of each VQ stage. In Fig. 9, the storage reduction ratio (SRR) is defined as

$$\mathrm{SRR} = \frac{\text{Memory required by the constrained-storage approach}}{\text{Memory required by MSVQ(GD)}} = \frac{M}{\sum_{i=1}^{p} z_i}. \quad (17)$$

The solid and dotted lines indicate the loss of coding quality when CSMSVQ and CSVQ are used to encode the image, respectively. From Fig. 9, the loss of coding quality by using CSMSVQ is less than that by using CSVQ. This phenomenon occurs for a reason. The GM algorithm outperforms the LBG algorithm when the sharing codebook is designed. The GM algorithm designs the sharing codebook according to the storage–distortion performance of CSMSVQ. The users need not to determine the sharing codebook size before the CSMSVQ is designed. The Design-CSMSVQ algorithm can automatically generate the sharing codebook in CSMSVQ based on the storage constraint. In the experiments conducted here, the LBG algorithm often searches for the local optimal solution when the universal codebook of CSVQ is designed.

5. Conclusions

MSVQ(GD) designed by the GD algorithm has the advantage of better coding performance than the traditional MSVQ designed by the LBG algorithm. It seems sometimes inappropriate for the users to determine the codebook size for each VQ stage. Specifically, the inappropriate division generally increases either the bit rate or the average distortion, or both.

This paper thus proposed MSVQ(GD). The GD algorithm is proposed to automatically search for the appropriate number of codewords in each VQ stage to maximize the rate–distortion function. Moreover, the threshold w is provided to control the structure of MSVQ(GD). MSVQ(GD) outperforms the traditional MSVQ, as shown in the present experiments. Furthermore, this paper also proposes the GM algorithm to search for the similar codewords in the codebooks of MSVQ(GD), and then the MSVQ(GD) using the sharing codebook, namely CSMSVQ, can be produced by the Design_CSMSVQ algorithm. In the experiments conducted here, although the coding quality obtained by CSMSVQ is smaller than that obtained using MSVQ(GD), the storage required by CSMSVQ decreases more than that required by MSVQ(GD).

References

[1] Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantizer design, IEEE Trans. Commun. 28 (1980) 84–95.

[2] A. Gersho, R.M. Gray, Vector Quantization and Signal Compression, Kluwer, Boston, 1992.

[3] B.H. Juang, A.H. Gray, Multiple stage vector quantization for speech coding, in: Proceedings of the IEEE International Conference on Acoustics, Speech, Signal Processing, 1982, pp. 597–600.

[4] C.F. Barnes, R.L. Frost, Vector quantizers with direct sum codebooks, IEEE Trans. Inf. Theory 39 (1993) 565–580.

[5] A.U. Khan, A.H. Mousa, Image coding using entropy-constrained reflected residual vector quantization, in: Proceedings of the IEEE International Conference on Image Processing, 2002, pp. 253–256.

[6] W.J. Hwang, B.Y. Ye, Storage- and entropy-constrained multi-stage vector quantization and its application to progressive image transmission, IEEE Trans. Consum. Electron. 43 (1) (1997) 17–23.

[7] W.Y. Chan, A. Gersho, Enhanced multistage vector quantization with constrained storage, in: Proceedings of the 24th Asilomar Conference on Circuits, Systems, and Computers, 1990, pp. 659–663.

[8] W.Y. Chan, A. Gersho, High fidelity audio transform coding with vector quantization, in: Proceedings of the IEEE International Conference on Acoustics, Speech, Signal Processing, 1990, pp. 1109–1112.

[9] W.Y. Chan, A. Gersho, Constrained-storage quantization of multiple vector sources by codebook sharing, IEEE Trans. Commun. 39 (1) (1991) 11–13.

[10] F. Kossentini, M.J.T. Smith, C.F. Barnes, Image coding using entropy-constrained residual vector quantization, IEEE Trans. Image Process. 4 (10) (1995) 1349–1357.

[11] F. Kossentini, W.C. Chung, M.J.T. Smith, Conditional entropy-constrained residual VQ with application to image coding, IEEE Trans. Image Process. 5 (2) (1996) 311–320.

[12] F. Kossentini, J.T. Smith, A fast PNN design algorithm for entropy-constraint residual vector quantization, IEEE Trans. Image Process. 7 (7) (1998) 1045–1050.

[13] S. Ramakrishnan, K. Rose, A. Gersho, Constraint-storage vector quantization with a universal codebook, IEEE Trans. Image Process. 7 (6) (1998) 785–793.

[14] Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantizer design, IEEE Trans. Commun. 28 (1) (1980) 84–95.

[15] A. Kimura, T. Kawanishi, K. Kashino, Acceleration of similarity-based partial image retrieval using multistage vector quantization, in: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04), 2004, pp. 993–996.

[16] V. Krishnan, D.V. Anderson, K.K. Truong, Optimal multistage vector quantization of LPC parameters over noisy channels, IEEE Trans. Speech Audio Process. 12 (1) (2004) 1–8.

[17] N. Phamdo, N. Farvardin, T. Moriya, A unified approach to tree structured and multistage vector quantization for noisy channels, IEEE Trans. Inf. Theory 39 (1993) 835–850.

[18] Z.M. Lu, D.G. Xu, S.H. Sun, Multipurpose image watermarking algorithm based on multistage vector quantization, IEEE Trans. Image Process. 14 (6) (2005) 822–831.

[19] F. Lahouti, A.K. Khandani, A. Saleh, Robust transmission of multistage vector quantized sources over noisy communication channels—applications to MELP speech codec, IEEE Trans. Veh. Technol. 55 (6) (2006) 1805–1811.

[20] S.Z. Selim, M.A. Ismail, K-means-type algorithm: generalized convergence theorem and characterization of local optimality, IEEE Trans. Pattern Anal. March. Intell. 6 (1984) 81–87.

[21] J.S. Pan, F.R. McInnes, M.A. Jack, VQ codebook design using genetic algorithms, Electron. Lett. 31 (1995) 1418–1419.

[22] M. Begum, N. Nahar, K. Fatimah, M.K. Hasan, M.A. Rahman, An efficient algorithm for codebook design in transform vector quantization, in: Proceedings of the WSCG, 2003.

[23] H. Sun, K.Y. Lam, A.L. Chung, W. Dong, M. Gu, J. Sun, Efficient vector quantization using genetic algorithm, Neural Comput. Appl. 14 (2005) 203–211.

[24] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, The MIT Press, Cambridge, MA, 1990.

[25] W.H. Equitz, A new vector quantization clustering algorithm, IEEE Trans. Acoust. Speech Signal Process. (1989) 1568–1575.

[26] J.S. Garofolo, L.F. Lamel, W.M. Fisher, J.G. Fiscus, D.S. Pallett, N.L. Dahlgren, The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus, 1993 [CD-ROM] NTIS PB91-100 354.

About the Author—SHIUENG-BIEN YANG received the B.S. degree in 1993 and the Ph.D. degree in 1999 both from the Department of Applied Mathematics, National Chung Hsing University, Taichung, Taiwan. He is currently an associate professor of Department of Computer Science and Information Engineering, Leader University, Tainan City, Taiwan. His research interests include pattern recognition, speech coding, image processing, and neural network.