Available online at www.sciencedirect.com

SCIENCE DIRECT®

Real-Time Imaging 11 (2005) 270–281

www.elsevier.com/locate/rti

Improved adaptive vector quantization algorithm using hybrid codebook data structure

Hsiu-Niang Chen[a,1], Kuo-Liang Chung[b,*,2]

[a]*Department of Information Management, National Taiwan University of Science and Technology, No. 43, Section 4, Keelung Road, Taipei 10672, Taiwan, ROC*

[b]*Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, No. 43, Section 4, Keelung Road, Taipei 10672, Taiwan, ROC*

Available online 9 June 2005

Abstract

Recently, Shen et al. [IEEE Transactions on Image Processing 2003;12:283–95] presented an efficient adaptive vector quantization (AVQ) algorithm and their proposed AVQ algorithm has a better peak signal-to-noise ratio (PSNR) than that of the previous benchmark AVQ algorithm. This paper presents an improved AVQ algorithm based on the proposed hybrid codebook data structure which consists of three codebooks—the locality codebook, the static codebook, and the history codebook. Due to easy maintenance advantage, the proposed AVQ algorithm leads to a considerable computation-saving effect while preserving the similar PSNR performance as in the previous AVQ algorithm by Shen et al. [IEEE Transactions on Image Processing 2003;12:283–95]. Experimental results show that the proposed AVQ algorithm over the previous AVQ algorithm has about 75% encoding time improvement ratio while both algorithms have the similar PSNR performance.

© 2005 Elsevier Ltd. All rights reserved.

1. Introduction

Vector quantization (VQ) has been successfully developed for speech, image, and video coding [1–4]. Many types of VQ such as classified VQ [5], index compressed VQ [6], finite state VQ [7–10], and some fast searching algorithms for VQ [11–15] have been proposed. Shannon's rate-distortion theory tells us that the VQ is asymptotically optimal for coding a stationary source [16]. However, since the image source is rarely stationary in practice, there still exists a gap between the theoretical performance and the real performance.

To improve this gap, many adaptive VQ (AVQ) algorithms [17–26] have been developed. These developed AVQ algorithms can adapt to the changing input source when the coding process progresses. In [24], Fowler proposed a new AVQ algorithm called the generalized threshold replenishment (GTR) algorithm and experimental results included the comparisons among the GTR, nonadaptive VQ, other AVQ algorithms, and theoretic bounds. As demonstrated in [24], the GTR algorithm is the best when compared to the others by using the rate-distortion criterion as the cost function. Recently, Shen et al. [26] presented a novel and efficient AVQ algorithm called the SZL algorithm for convenience. In the SZL algorithm, two new techniques, the locality consideration and the history aid, are adopted to improve the benchmark GTR algorithm. Using the locality consideration technique, the encoder and decoder update the operational codebook (OC) by inserting not only the input vector but also the selected neighboring locality vectors. Using the history aid technique, the encoder (decoder) also uses

*Corresponding author. Tel.: +886 2 273 76771; fax: +886 2 273 01081.
E-mail address: klchung@csie.ntust.edu.tw (K.-L. Chung).
[1]H.-N. Chen is also with Department of Information Management, Vanung University of Science and Technology, No. 1, Vannung Road, Shuiwei, Chungli 320, Taiwan, ROC.
[2]K.-L. Chung is supported in part by the National Science Council of ROC under Contract NSC92-2213-E011-079.

1077-2014/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rti.2005.04.004

the information of previously encoded vectors which are stored in the history codebook (HC) to quantize (decode) the input vector. Experimental results reveal that the SZL algorithm has a better peak signal-to-noise ratio (PSNR) performance when compared to the GTR algorithm although the encoding time is around three times that of the GTR algorithm when the codebook size is set to 32 and the range of the history vectors to be searched is set to 256. The motivation of this research is to present an improved AVQ algorithm to speed up the encoding time significantly while preserving the similar PSNR performance as in the SZL algorithm.

In this paper, we present an improved AVQ algorithm based on the proposed hybrid codebook data structure (HCDS). The proposed HCDS consists of three codebooks, namely the locality codebook (LC), the static codebook (SC), and the HC. The LC only has the locality vectors of the next input vector. The SC is constructed from the training images. The HC is obtained from the previously encoded vectors. Due to easy maintenance advantage by using the HCDS, the proposed AVQ algorithm leads to a considerable computation-saving effect while preserving the similar PSNR performance as in the SZL algorithm. Experimental results show that the proposed AVQ algorithm over the previous SZL algorithm has about 75% encoding time improvement ratio while preserving the similar PSNR performance.

Although, the performance of the proposed AVQ algorithm maybe inferior to that of the image coding with a transform coding stage, in [21,25], the authors have shown that AVQ algorithm can be combined with wavelet transform and leads to even better performance. In [25], a new algorithm using zerotrees of vectors of wavelet coefficients and the GTR algorithm for AVQ is described; this newly published algorithm is quite competitive to the wavelet-based algorithms employing nonadaptive scalar quantizers. We believe the proposed AVQ algorithm can be combined with a transform coding stage such as wavelet transform to obtain the good performance.

The paper is organized as follows. In Section 2, the AVQ algorithm by Shen et al. is revisited. Section 3 presents the proposed AVQ algorithm. Some experimental results are demonstrated in Section 4. Some concluding remarks are addressed in Section 5.

2. Past work: The SZL algorithm

In this section, the basic concept used in the AVQ algorithm is introduced first, then the past work by Shen et al. is revisited.

An AVQ algorithm usually updates the codebook dynamically according to the changing input source. An AVQ can be expressed as the following mapping function:

$$A_t : \Re^K \to C_t, \tag{1}$$

where K is the input vector's dimension and $C_t \subset \Re^K$ is the time-varying codebook. The time subscript t indicates that the mapping function A_t and the codebook C_t may change with time. The codebook C_t may contain more than one small codebook. Let $V_t \in \Re^K$ be a K-dimensional input vector which is usually transformed from the $\sqrt{K} \times \sqrt{K}$ input subimage, then the output of AVQ is a K-dimensional vector $\bar{V}_t$ such that

$$\bar{V}_t = A_t(V_t), \tag{2}$$

where the mapping function A_t maps V_t to $\bar{V}_t$ based on some specified minimal cost criterion. In [21–26], the rate-distortion criterion is used as the cost function. Given an input vector V_t and a rate-distortion parameter λ, if V_t is mapped to X, the cost of X is expressed by

$$J(X) = D(V_t, X) + \lambda \cdot l(X), \tag{3}$$

where $D(V_t, X)$ and $l(X)$ represent the distortion and the required code length, respectively. In Eq. (3), the Euclidean 2-norm is often used to measure the distortion $D(V_t, X)$ and is defined by

$$D(V_t, X) = ||V_t - X||^2 = \sum_{j=1}^{K} (V_{tj} - X_j)^2. \tag{4}$$

By Eqs. (3) and (4), the encoder usually finds the winning codeword X^* such that

$$X^* = \arg \min_{X \in C_t \cup \{V_t\}} J(X). \tag{5}$$

Eq. (5) indicates that the winning vector X^* as the output vector $\bar{V}_t$ can reach the minimal cost.

In [26], Shen et al. presented the SZL algorithm with codebook updating based on the locality consideration and the history aid. Since the neighboring vectors of input vector are often similar to the input vector, the encoder updates the OC by inserting not only the input vector but also the selected neighboring locality vectors. Let the distance-d locality vectors of a vector Z be denoted as $L_d(Z)$ which is defined as the previously encoded input vectors whose block distance from Z is not greater than d. For example, in Fig. 1, we have $L_1(Z) = \{a, b\}$ and $L_2(Z) = L_1(Z) \cup \{c, d, e, f\} = \{a, b, c, d, e, f\}$ where the locality vectors are ordered by the block distance. Following these ordered vectors and given an input vector V_t, the encoder first encodes the input vector V_t as $\bar{V}_t$ $(= a)$ and inserts a into (or moves a to) the first position of the OC. Then, the encoder inserts the other locality vectors (excluding a) of the next input vector Z into the OC and inserts them after the first position of the OC according to the block distance. In order to avoid redundantly inserting those locality

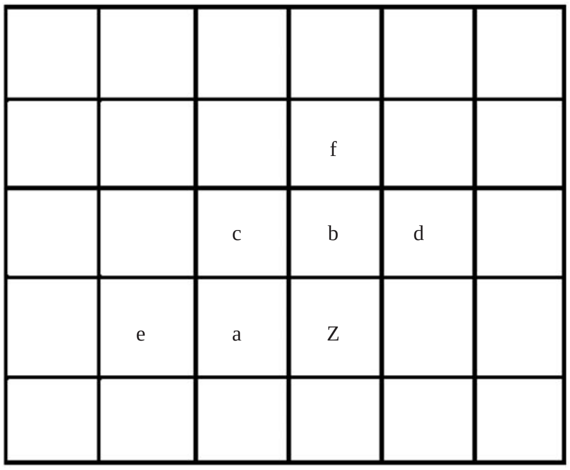

Fig. 1. Locality vectors of Z, $L_2(Z)$.

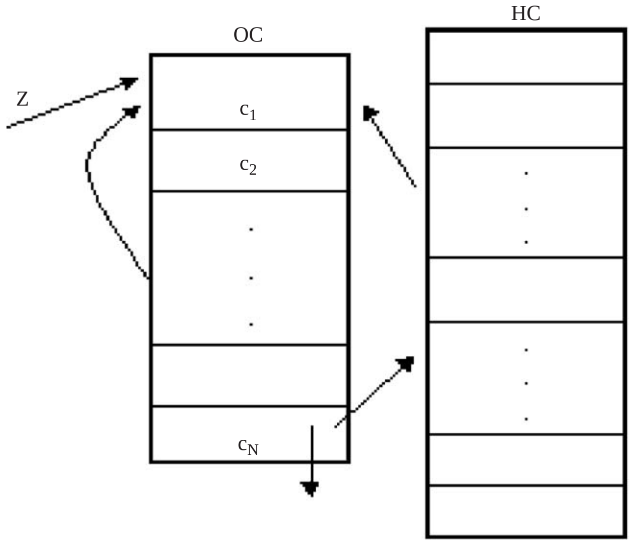

Fig. 2. Two codebooks OC and HC used in the SZL algorithm.

vectors that have been saved in the OC, an identifying process is needed. When the encoder has identified one locality vector being already in the OC, it needs to move this found codeword to the head of the OC.

After describing the locality consideration technique used in the SZL algorithm, we next introduce the history aid technique used in the SZL algorithm. As shown in Fig. 2, when a vector Z is inserted into the OC, the last codeword in the OC will be deleted and moved to the HC if the deleted codeword is not from the HC. Otherwise, the deleted codeword coming from the HC will not be moved back to the HC. In the SZL algorithm, a flag is used to indicate whether the deleted codeword will be moved to the HC or not. Therefore, besides the codewords stored in the OC, the encoder also uses the previously encoded vectors which are stored in HC to quantize the input vector.

Using the two codebooks OC and HC, in the SZL algorithm, the encoder evaluates the costs among the one to encode the new input vector directly, the one to encode the input vector by using the best codeword from the HC, and the one to encode the input vector by using the best codeword from the OC, then it selects the best encoding way with the minimum cost. If the winning case is the one to encode the new input vector directly, the encoder usually sends V_t to the decoder and inserts the input vector into the first position of the OC. If the winning case is the one to encode the input vector by using the best codeword from the HC (OC), the encoder sends the index of the winning codeword to the decoder and inserts the winning codeword into (moves the winning codeword to) the first position of the OC.

After encoding the winning case, the encoder inserts the other locality vectors of the next input vector V_{t+1} into the OC based on the selected $L_d(V_{t+1})$ for all cases. In order to improve the searching time, the index of each codeword in the HC is encoded by the fixed length coding and the HC is ordered according to the potentials of all codewords [13] to accelerate the searching time where the potential of a codeword $X = (X_1, X_2, \ldots, X_K)$ is defined as $P(X) = \sum_{i=1}^{K} X_i^2$. In addition, the indices of the codewords in the OC are encoded by the variable length coding and the OC is organized in a cache manner. Furthermore, the initial OC is usually constructed from the training images.

Using the locality consideration and the history aid techniques, the SZL algorithm has a better PSNR when compared to the GTR algorithm. But, to maintain the OC is a time-consuming work and the encoding time is much more than that of the GTR algorithm.

3. The proposed improved algorithm

In this section, we first present the proposed hybrid codebook data structure called the HCDS. Based on the proposed HCDS, we next present the proposed CCAVQ algorithm, where CC denotes the first letters of the authors' last names, to improve the time performance significantly while preserving the similar PSNR performance as in the SZL algorithm.

Before describing the proposed HCDS, let us examine the previous two codebooks OC and HC used in the SZL algorithm. Three observations on the OC are given as follows.

Observation 1. The OC contains two types of codewords, one from locality vectors and the other constructed from the training images.

Observation 2. In the OC, repeated dictionary operations, such as inserting the input vector or the locality vectors into the OC, inserting the codeword from the HC into the OC, moving the codeword from the OC to itself, or deleting the last codeword of the OC, make the maintenance of the OC some complicated.

Observation 3. For finding the winning codeword, it must search the whole OC. Thus, the searching time is proportional to the size of the OC.

From Observation 1, we suggest splitting the OC into two codebooks, namely the LC and the SC which is built up from the training images. The main reason is that the codebook constructed from the locality vectors will be changed dynamically due to the changing input vector. However, the codebook constructed from the training images is static. Fig. 3 depicts the proposed HCDS where the distance-2 locality vectors are used in the LC and V_t denotes the coming input vector. In Fig. 3, the LC only contains the locality vectors of the input vector V_t and the indices of the locality vectors are encoded by the variable length coding. The indices of the codewords in the SC and the HC are encoded via fixed length coding. In addition, the codewords in the SC and the HC are ordered by some specific property such as sums or potentials in order to speed up the searching time. In order to avoid a large amount of memory movements, we suggest using auxiliary index arrays in the SC and the HC.

From Observation 2, it is clear that updating the small LC is easier than updating the original whole OC in the SZL algorithm. Since we apply the fixed length coding to encode the codeword in the SC, one of the existing fast VQ searching algorithms, such as any one algorithm in [11–15], can be used in the SC. Using the VQ searching algorithm, it is not necessary to search the whole SC for finding the winning codeword and it leads to a computation-saving effect. From Observation 3, it is clear that finding the winning codeword from the SC is faster than finding the winning codeword from the whole OC in the SZL algorithm. Thus, from above discussion, the proposed CCAVQ algorithm leads to a considerable computation-saving effect. That is why we say we have the easy maintenance advantage when compared to the SZL algorithm. Since the LC and HC can be updated dynamically according to the changing input source, the proposed CCAVQ algorithm can preserve the similar PSNR performance as in the SZL algorithm.

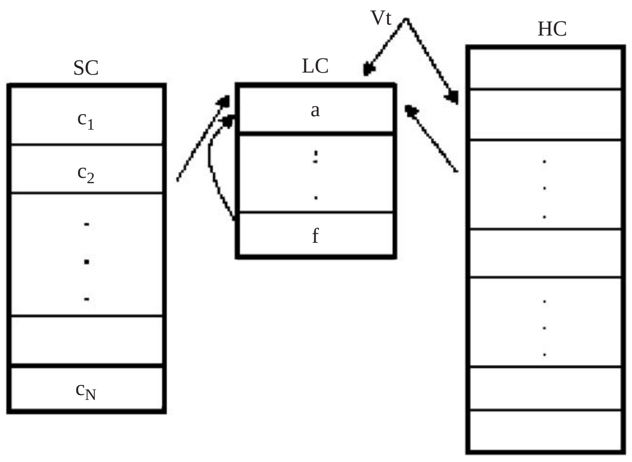

Fig. 3. The proposed hybrid codebook data structure.

According to Fig. 3, when the vector V_t is fed as the new input vector, the encoder evaluates the costs among the one to encode the new input vector directly, and the ones to encode the input vector by using the best codeword from the LC, the SC, and the HC, then it selects the best encoding way with the minimum cost. If the winning case is the one to encode the new input vector directly, the encoder sends V_t to the decoder, updates the LC and inserts the input vector into the HC. Since we do not need a flag array to indicate whether the deleted codeword will be inserted into the HC or not, the maintenance of the proposed CCAVQ algorithm is much easier than that of the SZL algorithm. If the winning case is the one to encode the input vector by using the best codeword from the LC, the SC, or the HC, the encoder sends the index of the winning codeword to the decoder and updates the LC. Given a rate-distortion parameter λ, the proposed five-step CCAVQ algorithm is listed as follows. The flowchart of the following proposed algorithm is supplemented in Fig. 4.

Step 1 (*Initialization*): From the training images, we build up the SC with size $N = |SC|$. Initially, we assign zeros to the LC with size $|L_d|$. The maximum size of the HC is set to $H_{\max}$ and the initial size of the HC is set to $M = 0$. Set the time counter to $t = 1$. Input the current input vector V_t.

Step 2 (*Computing four possible costs*).

Step 2.1 (*Computing the cost of the winning codeword from the LC*): Use full search to find the winning codeword e^* from the LC. Assume the index of the winning codeword e^* is j^*. By Eq. (3), the cost function of the winning codeword e^* is given by

$$J(e^*) = D(V_t, e^*) + \lambda \cdot l(e^*), \tag{6}$$

where $l(e^*)$ is the required code length if the input vector V_t is encoded as e^*.

Step 2.2 (*Computing the cost of the winning codeword from the SC*): Apply one of the existing fast searching algorithms, such as any one algorithm in [11–15], to find the winning codeword from the SC. Assume the index of the winning codeword c^* is i^*. The cost function of c^* is given by

$$J(c^*) = D(V_t, c^*) + \lambda \cdot l(c^*), \tag{7}$$

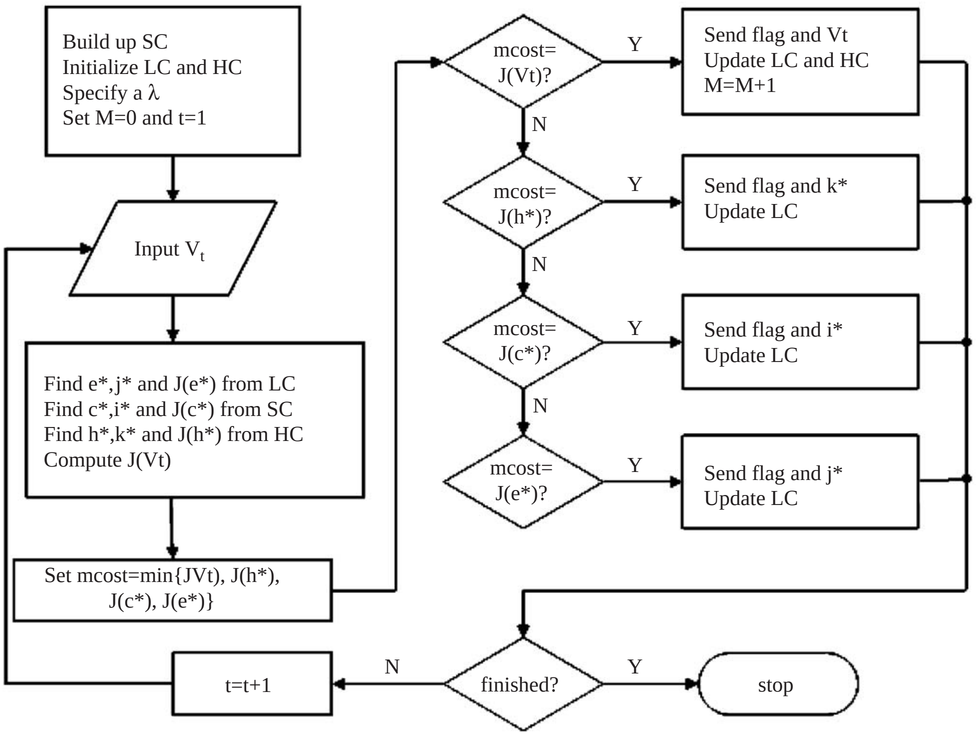

Fig. 4. Flowchart of the proposed algorithm.

where $l(c^*) = \log_2 N$ is the required code length if the input vector V_t is encoded as c^*.

Step 2.3 (*Computing the cost of the winning codeword from the HC*): If $M > 0$, apply one of the existing fast searching algorithms, such as any one algorithm in [11–15], to find the winning codeword from the HC. Assume the index of the winning codeword h^* is k^*. The cost function of h^* is given by

$$J(h^*) = D(V_t, h^*) + \lambda \cdot l(h^*), \quad (8)$$

where $l(h^*) = \log_2 M$ is the required code length if the input vector V_t is encoded as h^*. Otherwise, i.e. $M = 0$, we set the cost to ∞.

Step 2.4 (*Computing the cost to encode using the input vector directly*): Calculate the cost if the input vector V_t is encoded as itself. The cost function of V_t is given by

$$J(V_t) = D(V_t, V_t) + \lambda \cdot l(V_t) = \lambda \cdot l(V_t), \quad (9)$$

where $l(V_t)$ is the required code length if the input vector V_t is encoded as itself.

Step 3 (*Selecting the minimum one among the four possible costs*): Select the minimum cost among $J(V_t)$, $J(h^*)$, $J(c^*)$, and $J(e^*)$. According to the selected minimum cost, perform one of the following four actions.

Case $J(V_t)$ (V_t *is the winning codeword*): Send the flag to indicate that the input vector V_t is encoded as itself and send V_t to the decoder. Next, update the LC by inserting the input vector into the first position of the LC and update the HC by inserting the input vector into the proper position of the HC. Set $M = M + 1$.

Case $J(h^*)$ (h^* *is the winning codeword*): Send the flag to indicate that the input vector V_t is encoded as h^* and send the index k^* to the decoder. Then, update the LC by inserting the winning codeword h^* into the first position of the LC.

Case $J(c^*)$ (c^* *is the winning codeword*): Send the flag to indicate that the input vector V_t is encoded as c^* and send the index i^* to the decoder. Next, update the LC by inserting the winning codeword c^* into the first position of the LC.

Case $J(e^*)$ (e^* *is the winning codeword*): Send the flag to indicate that the input vector V_t is encoded as e^* and send the index j^* to the decoder. Next, update the LC by moving the winning codeword e^* to the first position of the LC if the winning codeword e^* is not in the first position.

Step 4 (*Inserting the other locality vectors into the LC*): Update the LC by inserting the other locality vectors of the next input vector into their destined positions of the LC.

Step 5: If t is less than the number of image blocks, set $t = t + 1$, read the next input vector, and go to Step 2. Otherwise, stop the process.

In Case $J(V_t)$ of Step 3, the encoder sends 8K bits representing V_t to the decoder in our algorithm. If the encoder quantizes the input vector V_t and the length of encoding bits is less than 8K, Eq. (9) can be modified easily. As described before, the LC only contains the locality vectors of the next input vector and the indices of the locality vectors are encoded by the variable length coding. In Step 2.1, different variable length coding scheme can be used to obtain the required code length. In the implementation of our proposed CCAVQ algorithm, the beforehand constructed Huffman codes are used to get the required code length.

In addition, the SC is usually available for the encoder and the decoder. Initially, both LC and HC are empty. The output of the encoder includes either the flag and index or the flag and current input vector. The flag indicates which case happened. Using the received data, the decoder can decode the current input vector and perform the same codebook updating as the encoder does. First, the decoder judges which case happened. If Case $J(V_t)$ in Step 3 happens, the decoder uses the received 8K bits to represent the current input vector. Then, it inserts the decoded vector into the HC and updates the LC. If the other cases in Step 3 happen, the decoder uses the received index to find the corresponding codeword in the specific codebook. Further, it updates the LC. After decoding all the received data, the image or video sequence can be decompressed. Since no searching algorithm is used in the decoding process, the decoding time is much faster than the encoding time.

In next section, some experiments are carried out to demonstrate the encoding time improvement of the proposed CCAVQ algorithm while preserving the similar PSNR performance when compared to the previous SZL algorithm.

4. Experimental results

To assure a fair comparison between the SZL algorithm and the proposed CCAVQ algorithm, following the same experiments as in the SZL algorithm, the similar types of images and video sequences are used to evaluate the performance comparison. The machine used in the experiments is Pentium III PC with 866 MHz and the used language is C language. The dimension of the input vector is 16 and the distance-2 locality vectors are used in the LC.

The LBG algorithm [27] is adopted to generate the initial codebook which is used in the OC of the SZL algorithm and used in the SC of the proposed CCAVQ algorithm simultaneously. The eight images in Fig. 5(a) are used as the training images to generate the initial codebook of the OC and the SC. Fig. 5(b) depicts the four testing images, namely Lena, Barbara, House, and Scene, each with size 512×512, used to evaluate the time and PSNR performance comparison between the SZL algorithm and the proposed CCAVQ algorithm. Figs. 6(a) and (b) show two images scanned from the 2003 book catalog of Open-Tech Pub., where the image in Fig. 6(a) is used as the training image and the image in Fig. 6(b) is used as the testing image, namely the Catalog image. In addition, Figs. 6(c) and (d) show two images captured from the web pages of the website www.kenphoto.com where the image in Fig. 6(c) is still used as the training image to generate the initial codebook of the OC and the SC. The Photo image as shown in Fig. 6(d) is used as the testing image. The pictures in Fig. 6 contain images and some Chinese and English characters, each with size 512×512. The testing video sequences shown in Fig. 7 consist of four frames of the Akiyo video sequence and four frames of the Garden video sequence, each with size 352×256. In Figs. 7(a) and (b), eight frames from two sequences are concatenated and interleaved so as to form the slow-changing patterns and the fast-changing patterns, respectively. Here, we take another frame from the Akiyo video sequence as the training image.

In the experiments, the codewords of the SC in the proposed CCAVQ algorithm and the codewords of the HCs in both concerning algorithms are sorted by the sum of each codeword in an increasing order where the sum of a K-dimensional codeword X is defined as $S(X) = \sum_{i=1}^{K} X_i$. Suppose the current minimum distortion is $D_{\min}$. Given an input vector V_t and a codeword X in the SC or in the HC,

$$\text{if } (S(V_t) - S(X))^2 \geqslant K \cdot D_{\min}, \quad \text{then the condition } D(V_t, X) \geqslant D_{\min} \text{ holds.} \tag{10}$$

If Eq. (10) holds, the codeword X will not be a possible winning codeword [11] and X can be discarded (the condition in Eq. (10) is also surveyed in [14]). This strategy can speed up the searching time in the SC and the HC. As described before, any existing fast searching algorithms, such as any one algorithm in [11–15], can be used to speed up the searching time of the SC in the proposed CCAVQ algorithm and the searching time of the HCs in both concerning algorithms. The faster the searching technique is, the less the time of our CCAVQ algorithm needs. Consequently, the important property in Eq. (10) is used to implement both algorithms. Since the OC in the SZL algorithm is not static, it is hard to employ any existing fast searching algorithm. Therefore, the full search algorithm is still used in the OC.

In Fig. 8, we plot the bit rate–PSNR performance for the testing image "Lena" using the codebook sizes, 256, 128, 64, and 32, where the codebook denotes the OC in the SZL algorithm and the SC in the proposed CCAVQ algorithm. Fig. 8 illustrates the similar PSNR performance for both concerning algorithms. In fact, all the testing images and video sequences used in the experiments have the similar PSNR performance for both algorithms. Therefore, we only plot the bit rate–PSNR performance for the testing image "Lena."

Since all the testing images and video sequences used in the experiments have the similar PSNR performance

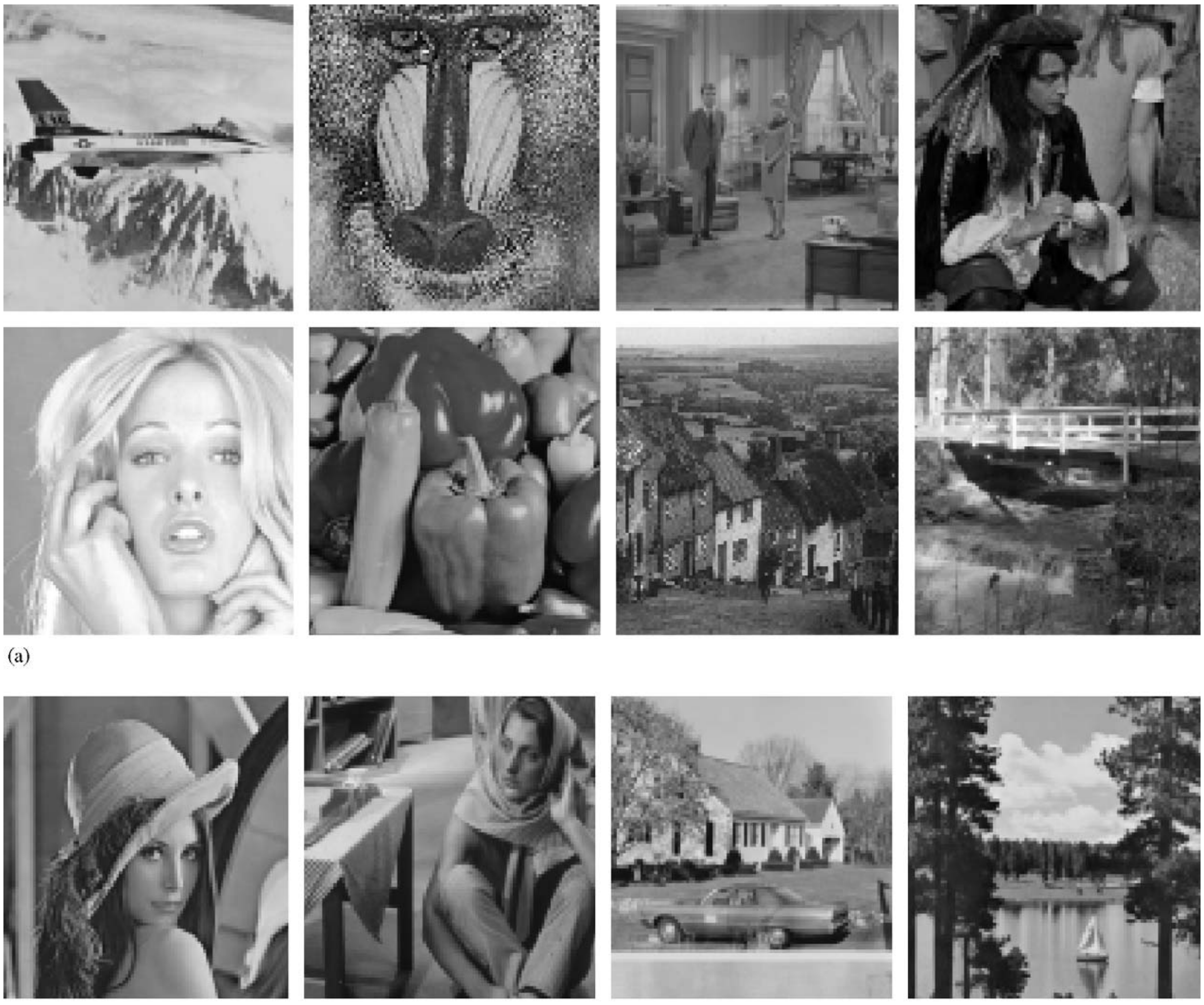
Fig. 5. (a) Eight training images and (b) four testing images.

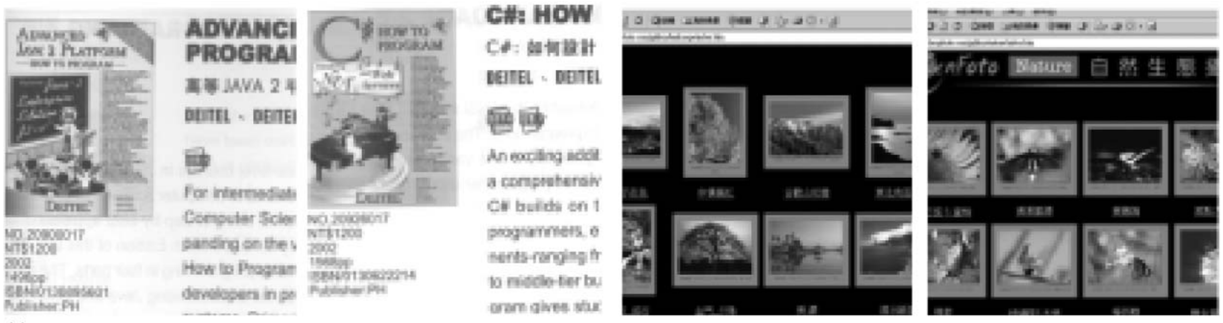
Fig. 6. Scanned images and captured web pages: (a) scanned training image, (b) scanned testing image, (c) captured web page for training and (d) captured web page for testing.

for both concerning algorithms, the bit rate–PSNR performance plot as shown in Fig. 8 can hardly distinguish one from the other. Therefore, the part of the experimental results for the testing image "Lena" with codebook size 256 is shown in Table 1 in order to show that the proposed CCAVQ algorithm is quite competitive to the previous SZL algorithm. We also depict some of the decoded images for "Lena" with codebook size 256 in Fig. 9 to illustrate the visual comparison. In the experiments, the total size of the

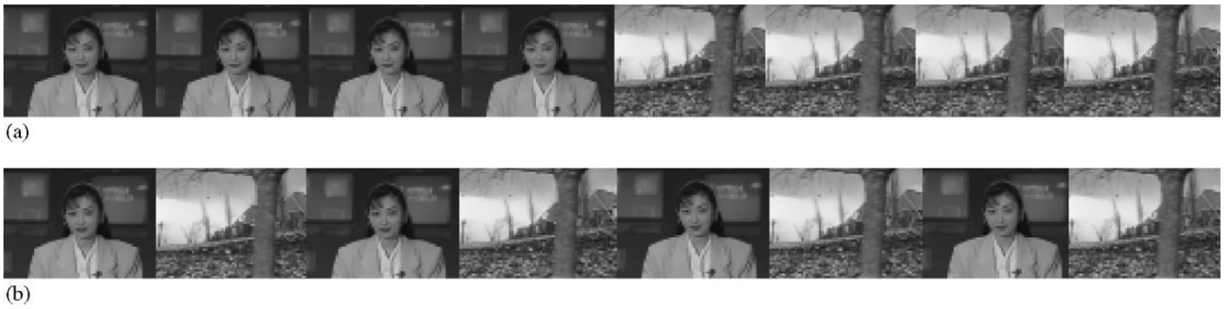

Fig. 7. Two video sequences: (a) concatenating sequence and (b) interleaving sequence.

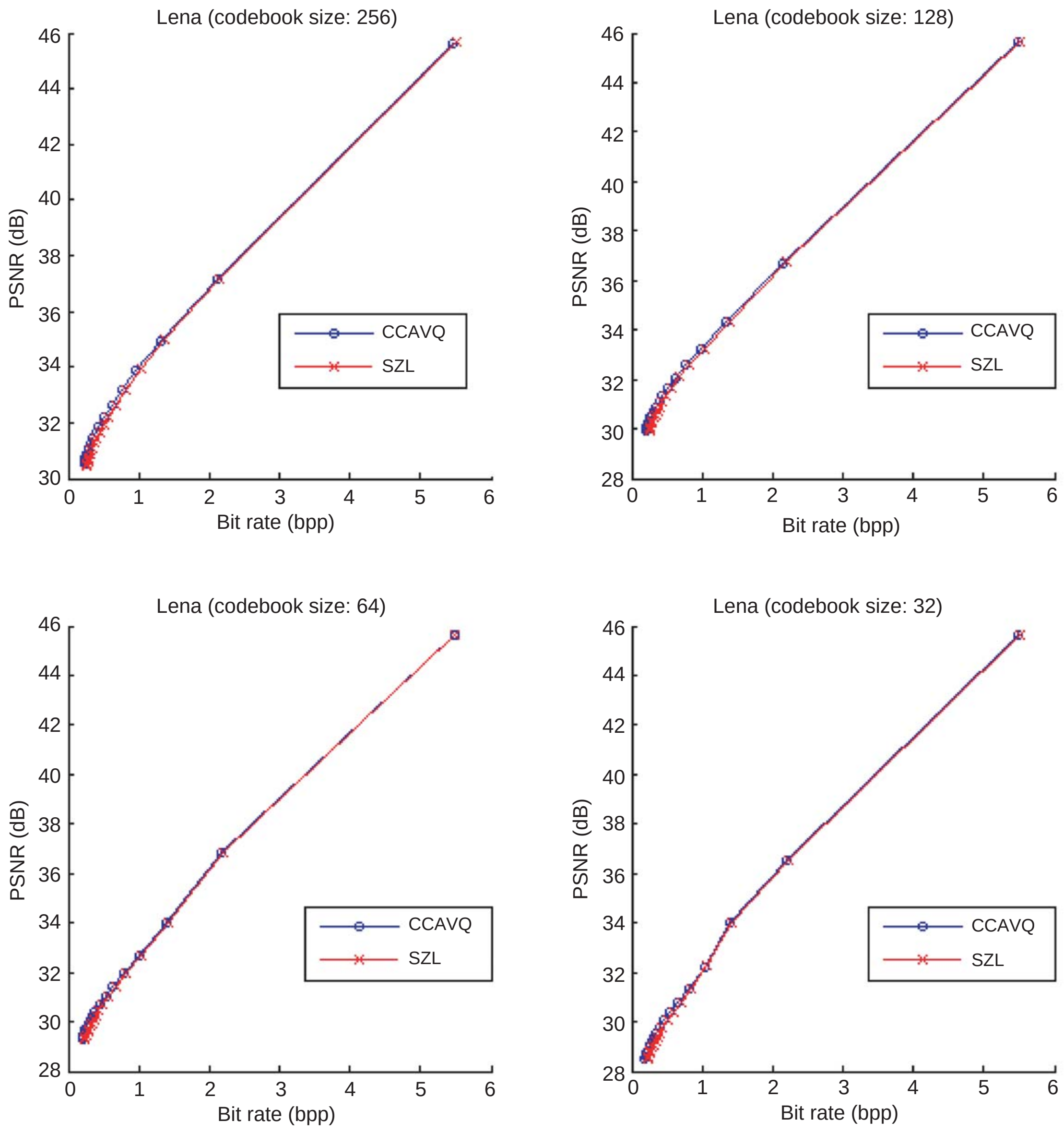

Fig. 8. Bit rate-PSNR performance for the testing image "Lena."

three codebooks used in the proposed CCAVQ algorithm is less than the total size of the two codebooks used in the SZL algorithm. The main reason is that the HC size used in the SZL algorithm is usually larger than the HC size used in the proposed CCAVQ algorithm (see Table 1). Note that the way to obtain the vectors of the HC in the CCAVQ algorithm is different from that in the SZL algorithm.

Table 1
Bit rate–PSNR performance for the testing image "Lena" with codebook size 256

λ	CCAVQ			SZL		
	Bit rate	PSNR	HC size	Bit rate	PSNR	HC size
1	5.464214	45.615870	10 839	5.503166	45.676255	11 087
5	2.107521	37.127776	3926	2.154171	37.155922	4069
10	1.323814	34.953046	2324	1.373631	34.978552	2461
15	0.963856	33.869679	1579	1.024998	33.914545	1751
20	0.750439	33.149259	1132	0.804737	33.178928	1291
25	0.615303	32.624969	849	0.669147	32.644947	1030
30	0.506340	32.198476	618	0.562592	32.213744	816
35	0.436176	31.867055	474	0.498840	31.905020	663
40	0.389252	31.626053	378	0.448753	31.652607	569
45	0.346931	31.407691	291	0.404202	31.418072	501
50	0.318840	31.245189	233	0.377380	31.264205	448
55	0.296375	31.113115	189	0.351460	31.104855	399
60	0.280807	31.011115	158	0.330036	31.003578	375
65	0.266342	30.897606	130	0.313515	30.882574	346
70	0.250977	30.779250	100	0.297588	30.761661	320
75	0.242813	30.718936	85	0.286930	30.676678	303
80	0.233894	30.642103	69	0.279015	30.584166	285
85	0.228207	30.583524	59	0.271717	30.520418	273
89	0.224285	30.544837	52	0.267334	30.474720	274

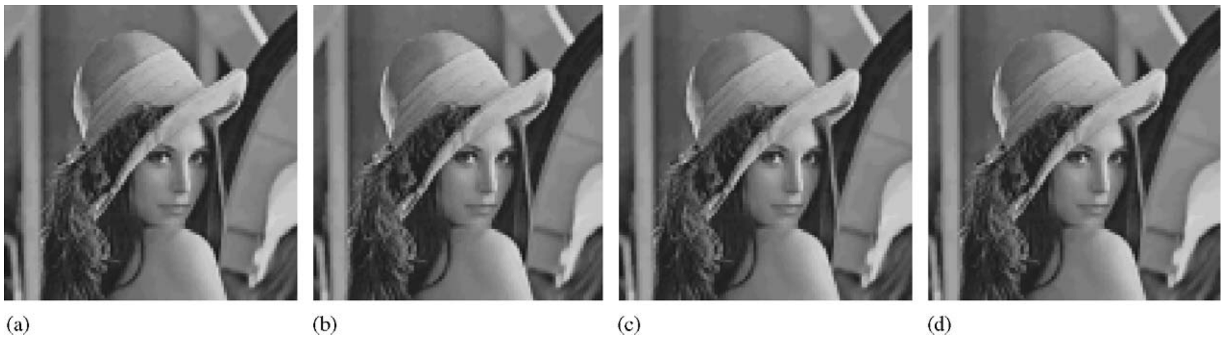
Fig. 9. Decoded image for the testing image "Lena" with codebook sizes 256: image generated by (a) CCAVQ for $\lambda = 30$, rate = 0.50634 bpp, and PSNR = 32.198476, (b) SZL for $\lambda = 34$, rate = 0.506958 bpp, and PSNR = 31.940309, (c) CCAVQ for $\lambda = 54$, rate = 0.300117 bpp, and PSNR = 31.135015 and (d) SZL for $\lambda = 69$, rate = 0.30035 bpp, and PSNR = 30.788631.

Table 2
Time comparison for testing images in Fig. 5(b)

Codebook size	Lena		Barbara		House		Scene	
	SZL	CCAVQ	SZL	CCAVQ	SZL	CCAVQ	SZL	CCAVQ
32	79	37	116	74	115	73	114	72
64	121	37	157	74	154	71	154	71
128	204	37	241	75	235	70	238	71
256	371	38	422	84	400	72	403	75

Tables 2–4 demonstrate the time requirement generated by the SZL algorithm and the proposed CCAVQ algorithm with λ ranging from 1 to 89 where the time unit is second. The codebook size vs. time curves for six testing images and two testing video sequences are shown in Fig. 10. Specifically, the required encoding time in the SZL algorithm is growing very quickly when the OC is growing. The reason is that the previous SZL algorithm must search the whole OC for finding the winning codeword and perform dictionary operations in

Table 3
Time comparison for Catalog and Photo testing images

Codebook size	Catalog		Photo	
	SZL	CCAVQ	SZL	CCAVQ
32	113	72	103	63
64	157	76	143	62
128	241	77	227	65
256	406	82	385	68

Table 4
Time comparison for two testing video sequences in Fig. 7

Codebook size	Concatenating		Interleaving	
	SZL	CCAVQ	SZL	CCAVQ
32	358	236	405	282
64	480	236	529	278
128	736	236	782	280
256	1250	243	1296	287

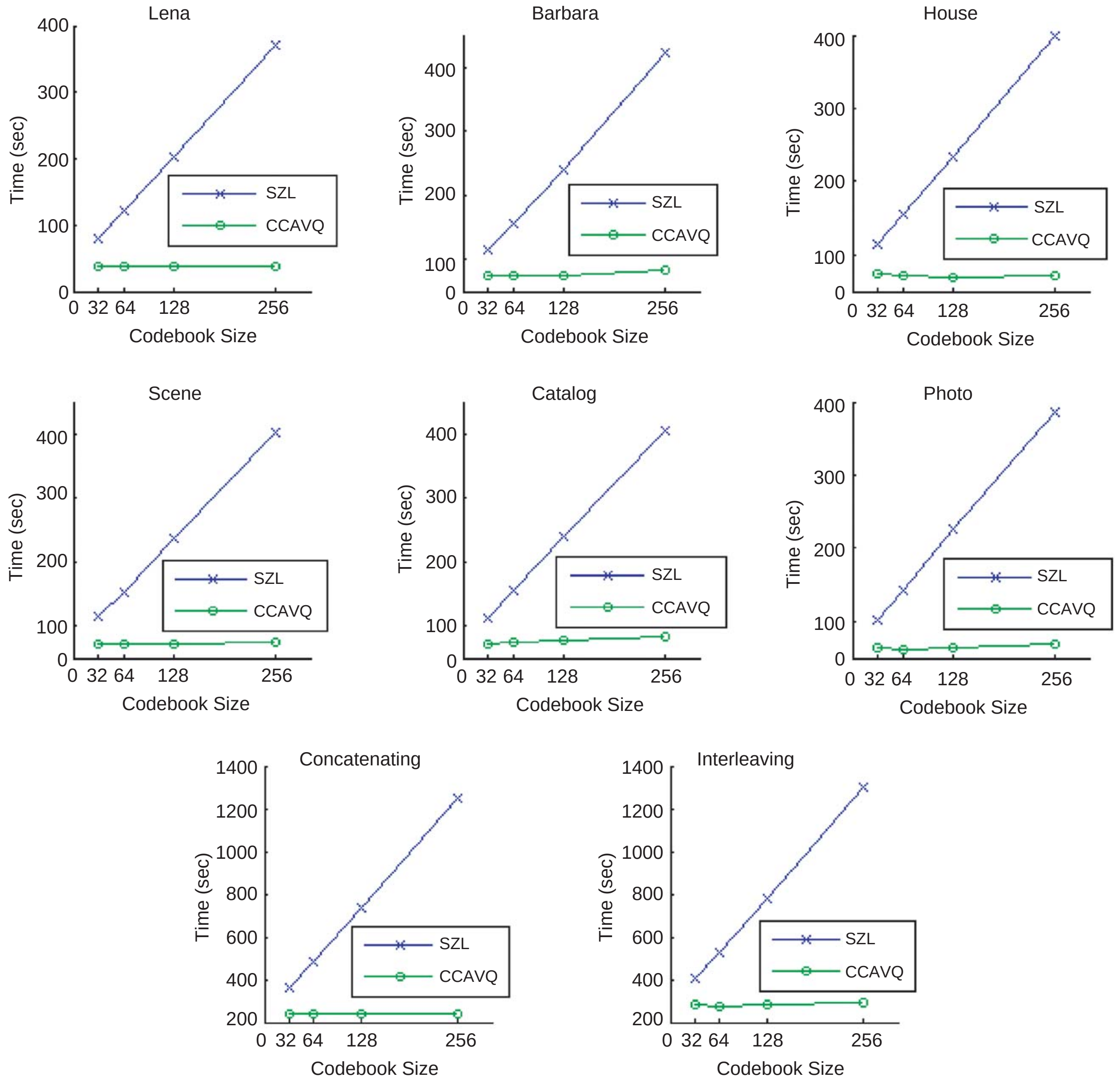

Fig. 10. Codebook size vs. time curves for six testing images and two testing video sequences.

the OC frequently. Thus, the larger the OC size is, the more the required time is. On the contrary, in the proposed CCAVQ algorithm, the existing VQ searching technique can be adopted in the SC and it is unnecessary to search the whole SC for finding the winning codeword. In addition, updating the small LC in the

Table 5
Time comparison for testing images in Fig. 5(b) included in the training set

Codebook size	Lena		Barbara		House		Scene	
	SZL	CCAVQ	SZL	CCAVQ	SZL	CCAVQ	SZL	CCAVQ
32	77	37	117	75	115	73	114	72
64	118	36	157	74	154	70	154	70
128	198	34	238	74	233	69	235	71
256	362	37	406	78	397	71	401	73

Table 6
Decoding time improvement ratios for testing images and video sequences

Image or video sequence	SZL		CCAVQ		Decoding time improvement ratio (%)
	Bit rate	Decoding time	Bit rate	Decoding time	
Lena	0.506958	0.177	0.50634	0.057	68
Barbara	0.500408	0.214	0.500259	0.057	73
House	0.498959	0.199	0.506954	0.058	71
Scene	0.499939	0.187	0.506134	0.058	69
Catalog	0.505173	0.164	0.50346	0.057	65
Photo	0.500961	0.131	0.500088	0.054	59
Concatenating	0.504297	0.66	0.505374	0.16	76
Interleaving	0.503497	0.67	0.501574	0.15	78

CCAVQ algorithm is easier than updating the whole OC in the previous SZL algorithm. Consequently, the proposed CCAVQ algorithm leads to a significant time-saving effect. When the codebook size is set to 256 and the searching size of the HC is set to 256, the proposed CCAVQ algorithm over the previous SZL algorithm has about 75% encoding time improvement ratio while preserving the similar PSNR performance as in the SZL algorithm.

In order to compare the performance difference between the training set excluding testing images and the training set including testing images (see Fig. 5(b)), both cases are included in the experiments. Because under the training set excluding testing images in Fig. 5(b), the performance comparison among the testing images has been demonstrated in Fig. 8 and Table 2, we now demonstrate the performance comparison among the testing images under the training set including testing images. Experimental results reveal that all the testing images in Fig. 5(b) have the similar PSNR performance for both concerning algorithms. For saving the context space, only one example is illustrated. For example, when the testing image is "Lena" with codebook size 256 and $\lambda = 70$, we have the bit rate 0.30645 bpp and the PSNR 30.851319 dB by using the SZL algorithm; the bit rate 0.250546 bpp and the PSNR 30.88489 dB by using the proposed CCAVQ algorithm. The time requirement is shown in Table 5 with λ ranging from 1 to 89. The time requirement in Table 5 is almost the same as the time requirement in Table 2.

When the codebook size is set to 256 and the bit rate is about 0.5 bpp, the decoding time requirements generated by the SZL algorithm and the proposed CCAVQ algorithm for all testing images and video sequences are shown in Table 6 where the time unit is second. Since the variable length coding is used in the OC of the SZL algorithm and the codes are generated by updating probability each time, the adaptive arithmetic coding [28] is used to produce the bitstreams in the OC. In the decoding process, it is unnecessary to search the codebooks for finding the winning codeword in both concerning algorithms. However, updating the small LC in the CCAVQ algorithm is easier than updating the whole OC in the SZL algorithm. Consequently, the proposed CCAVQ algorithm leads to a significant time-saving effect. When the codebook size is set to 256 and the bit rate is about 0.5 bpp, the proposed CCAVQ algorithm over the previous SZL algorithm has about 70% decoding time improvement ratio in average. The significant decoding time improvement could meet the real-time demand.

5. Conclusion

Using the rate-distortion criterion as the cost function, this paper has presented an improved AVQ algorithm based on the proposed HCDS. Due to the easy maintenance advantage, the proposed CCAVQ algorithm leads to a considerable computation-saving effect when compared to the previous SZL algorithm while preserving the similar PSNR performance as in the SZL algorithm. When the codebook size is set to 256 and the searching size of the HC is set to 256, the proposed CCAVQ algorithm has about 75% encoding time improvement ratio while preserving the similar PSNR performance as in the SZL algorithm. When the codebook size is set to 256 and the bit rate is about 0.5 bpp, the proposed CCAVQ algorithm over the previous SZL algorithm has about 70% decoding time improvement ratio in average.

References

[1] Gray RM. Vector quantization. IEEE Acoustics, Speech, and Signal Processing Magazine 1984; 4–29.

[2] Nasrabadi NM, King RA. Image coding using vector quantization: a review. IEEE Transactions on Communications 1988;36: 957–71.

[3] Guan L, Kamel M. Equal-average hyperplane partitioning method for vector quantization of image data. Pattern Recognition Letters 1992;13(10):693–9.

[4] Gersho A, Gray RM. Vector quantization and signal compression. Norwell, MA: Kluwer; 1992.

[5] Ramamurthi B, Gersho A. Classified vector quantization of images. IEEE Transactions on Communications 1986;34:1105–15.

[6] Hu YC, Chang CC. Low complexity index-compressed vector quantization for image compression. IEEE Transactions on Consumer Electronics 1999;45:219–24.

[7] Kim T. Side match and overlap match vector quantizers for images. IEEE Transactions on Image Processing 1992;1:170–85.

[8] Chang RF, Chen WM. Adaptive edge-based side-match finite-state classified vector quantization with quadtree map. IEEE Transactions on Image Processing 1996;5:378–83.

[9] Chen TS, Chang CC. A new image coding algorithm using variable-rate side-match finite-state vector quantization. IEEE Transactions on Image Processing 1997;6:1185–7.

[10] Wei HC, Tsai PC, Wang JS. Three-sided side-match finite-state vector quantization. IEEE Transactions on Circuits and Systems for Video Technology 2000;10:51–8.

[11] Ra SW, Kim JK. Fast mean-distance-ordered partial codebook search algorithm for image vector quantization. IEEE Transactions on Circuits and Systems II 1993;40:576–9.

[12] Chen TS, Chang CC. Diagonal axes method (DAM): a fast search algorithm for vector quantization. IEEE Transactions on Circuits and Systems for Video Technology 1997;7:555–9.

[13] Shen G, Liou ML. Window-based fast search algorithm for image vector quantization. In: Proceedings of the picture coding symposium, April 1999. p. 147–51.

[14] Pan JS, Lu ZM, Sun SH. An efficient encoding algorithm for vector quantization based on subvector technique. IEEE Transactions on Image Processing 2003;12:265–70.

[15] Lai ZC, Liaw YC. Fast searching algorithm for vector quantization using projection and triangular inequality. IEEE Transactions on Image Processing 2004;13:1554–8.

[16] Shannon CE, Weaver W. The mathematical theory of communication. Urbana, IL: University of Illinois Press; 1963.

[17] Paul DB. A 500–800 bps adaptive vector quantization vocoder using a perceptually motivated distance measure. In: Conference on recording IEEE globecom, 1982. p. 1079–82.

[18] Gersho A, Yano M. Adaptive vector quantization by progressive codevector replacement. In: International conference on acoustics, speech, and signal processing, May 1985. p. 133–6.

[19] Bentley JL, Sleator DD, Tarjan RE, Wei VK. A locally adaptive data compression scheme. Communications on Association of Computing Machinery 1986;29:320–46.

[20] Wang X, Shende S, Sayood K. Online compression of video sequences using adaptive VQ codebook. In: Storer JA, Cohn M, editors. IEEE data compression conference, Snowbird, UT, March 1994. p. 185–94.

[21] Lightstone M, Mitra SK. Adaptive vector quantization for image coding in an entropy-constrained framework. In: International conference on image processing, Austin, TX, November 1994, vol. 1. p. 618–22.

[22] Lightstone M, Mitra SK. Image-adaptive vector quantization in an entropy-constrained framework. IEEE Transactions on Image Processing 1997;6:441–50.

[23] Fowler JE, Ahalt SC. Adaptive vector quantization using generalized threshold replenishment. In: Storer JA, Cohn M, editors. IEEE data compression conference, Snowbird, UT, March, 1997. p. 317–26.

[24] Fowler JE. Generalized threshold replenishment: an adaptive vector quantization algorithm for the coding of nonstationary sources. IEEE Transactions on Image Processing 1998;7:1410–24.

[25] Fowler JE. Adaptive vector quantization for efficient zerotree-based coding of video with nonstationary statistics. IEEE Transactions on Circuits and Systems for Video Technology 2000;10:1478–88.

[26] Shen G, Zeng B, Liou ML. Adaptive vector quantization with codebook updating based on locality and history. IEEE Transactions on Image Processing 2003;12:283–95.

[27] Linde Y, Buzo A, Gray RM. An algorithm for vector quantizer design. IEEE Transactions on Communications 1980;28:84–95.

[28] Witten IH, Neal RM, Cleary JG. Arithmetic coding for data compression. Communications on Association of Computing Machinery 1987;30:520–40.