Pattern Recognition Letters 30 (2009) 653–660

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

A fast VQ codebook generation algorithm via pattern reduction

Chun-Wei Tsai [a], Chao-Yang Lee [b], Ming-Chao Chiang [a,*], Chu-Sing Yang [b]

[a] *Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan, ROC*
[b] *Department of Electrical Engineering, National Cheng Kung University, Tainan 70101, Taiwan, ROC*

A R T I C L E   I N F O

Article history:
Received 22 July 2008
Received in revised form 10 February 2009
Available online 26 February 2009

Communicated by G. Borgefors

Keywords:
Vector quantization
Codebook generation
Pattern reduction

A B S T R A C T

In this paper, we present a simple but fast codebook generation algorithm, called PREGLA (Pattern Reduction Enhanced GLA). The proposed algorithm is fundamentally different from the previous approaches in that the previous approaches focus on reducing the size of the codebook whereas the proposed algorithm focuses on using pattern reduction to reduce the computation time. The proposed algorithm is motivated by the observation that input vectors that are "static" during the training process can be considered as part of the final solutions and thus can be compressed and removed to eliminate the redundant computations at the later iterations of the training process. To evaluate the performance of the proposed algorithm, we compare the proposed algorithm with "efficient" state-of-the-art GLA or GLA-based algorithms such as Codeword Displacement, Nearest Partition Set Search, Fast Vector Quantization Algorithm, Law of Cosines, and standard GLA. Our experimental results indicate that the proposed algorithm can reduce the computation time from 29.45% up to about 77.98% compared to those of standard GLA and other fast GLA-based algorithms alone.

1. Introduction

Vector quantization (VQ) is one of the most important technologies for the lossy signal compression in use today. In general, VQ uses a codebook to encode and decode the signal and transmits the compressed signal over a communication channel. Because it is simple and easy to implement, VQ has been widely used in different applications such as pattern recognition (Abdel-Galil et al., 2005; Kugler and Lopes, 2003; Kusumoputro et al., 2002; Winger, 2001), pattern compression (Laha et al., 2004; Laskaris and Fotopoulos, 2004), speech recognition (Barszcz et al., 2000; Kim and Kleijn, 2004), and face detection (Trentin and Gori, 2002). As one of the most important algorithms for codebook generation and due to its simplicity and relatively good fidelity, Generalized Lloyd Algorithm (GLA), also known as Linde–Buzo–Gary (LBG) algorithm (Linde et al., 1980), is the most widely used VQ method. It is basically an iterative algorithm that is composed of two steps: (1) partitioning and (2) codebook generation. The idea behind GLA is to find an optimal codebook that minimizes the distortion between the training set and the codebook. In other words, GLA is a search technique that finds the closest codewords (or winners) for the input vectors.

A vector quantizer maps a set of ℓ-dimensional vectors $X = \{x_1, x_2, \ldots, x_n\}$ in the vector space $\mathbb{R}^\ell$ into another set of vectors $C = \{\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_m\}$ in the same space. Each vector $\bar{c}_j$ is called a code vector (or a codeword), and the set C is called a codebook. With each input vector x_i is associated a nearest neighbor codeword $\bar{c}_j$ in terms of the Euclidean distance. That is, $\forall k \neq j, \|x_i - \bar{c}_j\| \leqslant \|x_i - \bar{c}_k\|$. In other words, an optimal vector quantizer can be defined as a partition of the input vectors as follows:

Definition 1 (*Optimal vector quantizer*). Given a set of input vectors $X = \{x_1, x_2, \ldots, x_n\}$ in the space $\mathbb{R}^\ell$, the output of an optimal vector quantizer are a partition $\pi = \{B_1, B_2, \ldots, B_m\}$[1] and a set of vectors (or codewords) $C = \{\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_m\}$ in the same space as the input vectors such that

$$\bar{c}_j = \frac{1}{|B_j|} \sum_{x \in B_j} x \tag{1}$$

and

$$B_j = \{x \in X \mid \|x - \bar{c}_j\| \leqslant \|x - \bar{c}_k\|, \forall j \neq k\}. \tag{2}$$

In short, the definition says that codewords generated by an optimal vector quantizer as given in Eq. (1) must satisfy the constraints given in Eq. (2).

Patané and Russo (2001) pointed out that all the VQ algorithms fall into two categories: (1) k-means-based and (2) competitive-learning-based. GLA and GLA-based algorithms belong to the first category. That is, they are all k-means-based and thus share the pros and cons of k-means and k-means-based algorithms. On one

* Corresponding author. Tel.: +886 7 5252000x4321; fax: +886 7 5254301.
E-mail address: mcchiang@cse.nsysu.edu.tw (M.-C. Chiang).

[1] In other words, $X = \cup_{j=1}^{m} B_j$ and $\forall j \neq k, B_j \cap B_k = \emptyset$.

0167-8655/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2009.02.003

side, the good news is that many "efficient" algorithms based on GLA and k-means are essentially interchangeable. On the other side, the bad news is that GLA shares the same major shortcomings of k-means such as extremely sensitive to the initial conditions (Xiong et al., 2004), trapped to the local minima (Shen et al., 2003), and scalability (Ebrahimi-Moghadam and Shirani, 2005). Insofar as the scalability of GLA is concerned, it is apparent that one of the most serious obstacles nowadays is the computation time. The basic idea behind GLA is straightforward and can be outlined as given in Fig. 1.

1.1. Motivation of the work

In this paper, we propose a simple but efficient algorithm, called PREGLA (Pattern Reduction Enhanced GLA), to reduce the computation time required to generate the codebook for GLA and GLA-based algorithms. Similar to the k-means and k-means-based algorithms, the characteristics of convergence of the GLA and GLA-based algorithms make it very unlikely to move the input vectors from one codeword to another at the later iterations of the evolution process. For instance, Fig. 2 shows that if a vector, say, x_i, is *known* to belong to a codeword, say, $\bar{c}_j$, at a certain iteration, say, t, and is very unlikely to be moved to another codeword at later iterations, then it can be first compressed and then removed, to prevent it from being involved in the computations at the later iterations of the evolution process.

The key question now is how these patterns are compressed and removed in such a way that it would reduce the computation time while at the same time retaining the accuracy of the end results. PREGLA uses three methods to accomplish this task, which are implemented as three different modules: refining module, detection module, and removal module. One of the strengths of the proposed algorithm is that it can be easily combined with GLA or GLA-based algorithms or even other iterative algorithms. In addition, the proposed algorithm would not deteriorate the accuracy of the end results compared to those of standard GLA and other fast GLA-based algorithms alone.

1.2. Contribution of the paper

Most, if not all, of the researches in this area have focused their attention on improving the computation time of VQ codebook generation based on codeword reduction or structured codebook. The main contributions of the paper are twofold:

1. We present a simple but fast VQ codebook generation algorithm that is fundamentally different from all the existing methods with respect to how the computation time is reduced. Instead of the codeword reduction or structured codebook methods, the proposed algorithm uses the pattern reduction method to reduce the computation time of GLA or fast GLA-based algorithms. Moreover, it can also retain the quality of the end result in terms of PSNR (Peak Signal-to-Noise Ratio) as defined in Eq. (5). In addition, the proposed algorithm can be easily combined with other fast VQ codebook generation algorithms to further enhance their performance.

```
GLA1. Create an initial codebook.
GLA2. Partition the input vectors into a set of disjoint blocks as given in
      Definition 1.
GLA3. Compute the centroids of each block using Eq. (1).
GLA4. If the average distortion satisfies the termination criterion, then stop
      and output the result; otherwise, go to step 2.
```

Fig. 1. Outline of the GLA algorithm.

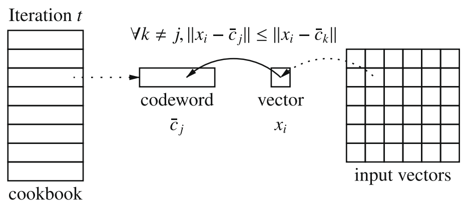

Iteration t
$\forall k \neq j, \|x_i - \bar{c}_j\| \leq \|x_i - \bar{c}_k\|$
codeword
$\bar{c}_j$
vector
x_i
input vectors
cookbook

Fig. 2. Example illustrating that the result of GLA at the end of iteration t where vector x_i is assigned the codeword $\bar{c}_j$.

2. We compare the performance of the proposed algorithm with all state-of-the-art GLA or GLA-based algorithms, by using a couple of well-known datasets. All our simulation results showed that the proposed algorithm outperforms all state-of-the-art algorithms we evaluated in this paper.

1.3. Organization of the paper

The remainder of the paper is organized as follows. Section 2 gives a brief introduction to the fast GLA-based algorithms and shows how state-of-the-art researches in this area have tried to solve the computation time issue. Section 3 provides a detailed description of the proposed algorithm. Performance evaluation of the proposed algorithm is presented in Section 4. Conclusion is given in Section 5.

2. Related works

Researches on VQ codebook generation can be divided into two categories: one on improving the quality of the codebook and the other on reducing the computation time of the codebook generation. Recently, it is saving the computation time that has become more and more important because the size of problems relying on VQ for a solution nowadays is much larger than, say, 20 years ago. Vasuki and Vanathi (2006) gave a good survey of several different methods for enhancing the performance of vector quantization and a brief introduction to those methods. Giuseppe and Marco (2001) first showed that the end results of traditional LBG depend heavily on the initial conditions. He then proposed an enhanced LBG algorithm (ELBG) to avoid from falling into bad local minimum, which can be used to improve the search capability of the LBG's algorithm as well. In (Pan and Cheng, 2007), Pan and Chen presented an evolution-based tabu search approach (ETSA) for designing codebooks for vector quantization with smaller distortion values. ETSA can also prevent premature convergence. In (Shen and Hasegawa, 2006), Shen and Hasegawa presented a new method, an adaptive incremental GLA, which generates codevectors incrementally for VQ. If the distortion error is high, new codevectors are inserted in regions of the input vector space until the desired number of codevectors is achieved. During the incremental process, a removal-insertion technique is used to adjust the codebook.

During the training process, extensive computations are usually required to generate a codebook. Thus, many fast methods have been developed to reduce the complexity barrier of the VQ techniques. Some of these methods find the best matching codevector in a codebook for a training vector without increasing the time complexity. The approaches for achieving this objective can be divided into two classes. One is based on using a suitable data structure to ease the search process, such as tree-structured vector quantization (TSVQ) (Buzo et al., 1980). The other is to reduce the search complexity of the partitioning step of GLA. In other words, the former builds a codeword structure for speeding up the search and mapping the input vector and codeword quickly. The latter uses the relationships between the input vectors and codewords to reduce the redundant computations.

2.1. Structured codebook

There exist several researches (Buzo et al., 1980; Cardinal, 2000; Chena and Chung, 2005; Yang, 2004) on using the structured codebook for enhancing the performance of VQ. In (Buzo et al., 1980), Buzo et al. present a binary tree-structured vector quantization (TSVQ) method for reducing the search time of the closest codeword. In addition, Ramasubramanian and Paliwal (Ramasubramanian and Paliwal, 1992) present a method that uses a tree structure named *k*-*d* tree and is very efficient for codevector search. Each node of the tree examines only one component of the training vector. To improve the quality of the end result, Yang (Yang, 2004) combines the genetic algorithm with TSVQ to modify the codeword training procedure, called the variable-branch TSVQ (VBTSVQ). The study in (Chang et al., 2006) presents a full-search equivalent TSVQ (FSE-TSVQ) approach, which is an improvement to TSVQ. It can obtain the closest codevector for each input vector. FSE-TSVQ employs the triangle inequality for efficiently avoiding impossible codevectors. Moreover, that study also develops the enhanced DP-TSVQ (EDP-TSVQ) algorithm. It achieves a better trade-off between encoding time and image quality. Another fast codebook search method is given in (Chang and Wu, 2007). The proposed method is built on the planar Voronoi diagram to label a ripple search domain. The appropriate codevector can easily be found by searching the local region.

2.2. Codeword Reduction

A different approach (Chang and Lin, 2005; Chung and Lai, 2004; Mielikainen, 2002; Pan et al., 2004) to saving the computation time of VQ is to reduce the distance comparison of codeword. In (Qian, 2004), a simple and effective fast codevector search method based on codeword reduction is proposed. A training vector does not require a search to find the best matching codevector if its distance to the partition is improved at the current iteration compared to that of the previous iteration. In (Pan et al., 2003), an efficient algorithm, which uses the mean values and variances of an input vector and its two sub-vectors to reject unlikely codevectors, is proposed. In (Lai and Liaw, 2004), a fast-searching algorithm using projection and inequality (FAUPI) is addressed to reject unlikely codevectors. Two inequalities are used to reduce the distortion computations: one for terminating the searching process and the other for deleting the impossible codevectors. This algorithm uses features of vector to reject unlikely codevectors. The nearest partition set (NPS) for improving the quantization complexity of full-search vector quantization (VQ) is presented in (Qian, 2006). It uses the fact that a vector in a training sequence is either placed in the same minimum distance partition (MDP) with the previous iteration or in a partition within a very small subset of partitions. In (Lai et al., 2008), a fast codebook generation algorithm, called CGAUCD (Codebook Generation Algorithm Using Codeword Displacement), is presented. It makes use of the codevector displacement between successive partition processes. The codebook generation time can be reduced. It can generate the same codebook as that produced by the GLA.

3. The proposed method

In this section, we give a detailed description of the proposed method. We begin with a list of notations to be used throughout this section. Then, a brief introduction to the proposed algorithm will be given, followed by a more detailed explanation. After that, we present a simple example to show how the proposed algorithm works. Finally, the performance analysis of the proposed algorithm is presented.

3.1. Notations

To simplify our discussion that follows, the following notations will be used throughout this section except where no confusion is possible.

X	The input vector (or pattern) set. In other words, $X = \{x_1, x_2, \ldots, x_n\}$ where x_i is the *i*th input vector and n the number of input vectors.
C	The codebook, i.e., $C = \{\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_m\}$ where $\bar{c}_j$ is the *j*th codeword and m the number of codewords.
B_j	The Voronoi region of codeword $\bar{c}_j$. In other words, $X = \cup_{j=1}^{m} B_j$ and $\forall j \neq k, B_j \cap B_k = \emptyset$.
d_{ij}	The distance between input vector x_i and codeword $\bar{c}_j$.
N_j	The number of input vectors in the cluster represented by the codeword $\bar{c}_j$.
t	The iteration or generation number.
R	The set of removed patterns.
Θ_t	The removal bound at iteration t, which is defined as $\Theta_t = \Theta_{t-1} \times (1+\alpha)^{t-2}$ where α is the removal rate and $t \geqslant 3$.
D_t^j	The radius of the circle centered at $\bar{c}_j$. Its value is determined by the value of Θ_t and represents the top Θ_t percent of all the patterns in each cluster. In other words, if $d_{ij} < D_t^j$, then x_i will be removed.

3.2. The PREGLA

Fig. 3 gives an outline of the PREGLA, built on the framework of standard GLA. The main difference between these two algorithms is in that PREGLA adds three modules named refining, detection, and removal to the standard GLA, as shown in Fig. 3.

Initially, PREGLA uses sampling to determine a better initial pattern set to start with (lines 3 and 4). The partitioning procedure, on line 7, is then carried out to find the codeword $\bar{c}_j$ for each input vector x_i in such a way that $\forall j \neq k, \|x_i - \bar{c}_j\| \leqslant \|x_i - \bar{c}_k\|$. The determination procedure, on line 8, takes care of updating each codeword value $\bar{c}_j$ in the codebook C for the next iteration. In other words, the partitioning and determination procedures are but the standard GLA. After the standard GLA is applied, the main procedure of PREGLA starts, on line 11. A very simple strategy for saving the computing time is for PREGLA to locate patterns that are

```
1  Algorithm PREGLA
2  {
3     Randomly select a set of samples $S = \{s_1, s_2, \ldots, s_q\}$ from the input vector
         set using the refining module.
4     Use the sample set $S$ to train and generate the initial codebook, i.e.,
         $C$ = Standard_GLA($S$).
5     Let $t = 1$ and $R = \emptyset$.
6     /* The process of GLA */
7     Partition the remaining input vector $X - R$ into a set of disjoint blocks $B_j$,
         $j = 1, 2, \ldots, m$.
8     Determine the centroids $\bar{c}_j$ of $B_j$, $j = 1, 2, \ldots, m$.
9     /* The process of PREGLA */
10    /* detect, remove, and compress */
11    For each $x_i$, $x_i \in (X - R)$ and $x_i \in B_j$, if $d_{ij} < D_t^j$, then let $R = R \cup \{x_i\}$.
12    If the average distortion satisfies the stopping criterion, go to line 14.
13    Else let $t = t + 1$, go to line 7.
14    Output the results.
15 }
```

Fig. 3. The proposed algorithm PREGLA.

"static," in the sense that they have a very small probability to be reassigned a different codeword. The detection module (on line 11) of PREGLA does exactly that, trying to find patterns that have a small chance to be reassigned a different codeword and thus can be considered as static. The removal module (also on line 11) then compresses the static patterns so that they are no longer involved in any further computations, thus effectively eliminating all the redundant computations. On line 12, the stopping criterion is defined to be either all the values of the centroids remain the same as those of the previous iteration or the predefined maximum number of iterations has been reached, whichever occurs first. The results are output on line 14. In what follows, we will discuss how each module of PREGLA works.

3.2.1. Refining module

Bradley and Fayyad (1998) pointed out that both GLA and k-means are extremely sensitive to the initial points (conditions). For this reason, PREGLA uses the refining module to find a set of better initial points that would decrease the sensitivity of GLA. In this paper, a simple and straightforward sampling method is adopted for creating the initial codebook. The number of samples is set to 10% of the input vector set, i.e., $n/10$. In other words, the refining module randomly chooses a set of samples $S = \{s_1, s_2, \ldots, s_q\}$ from the input vector set where $q = n/10$ (line 3 of Fig. 3). The set of samples S is then used for training and generating the initial codebook C via standard GLA (line 4 of Fig. 3). In this paper, the refining module is designed primarily for providing a stable final result.

3.2.2. Detection module

The detection module plays the role of finding patterns that have a small probability to be reassigned a different codeword, i.e., finding the so-called "static" patterns, as we have previously stated. In this paper, we are assuming that input vectors *near* a codeword, say, $\bar{c}_j$ will not be reassigned another codeword. Then, the question becomes what patterns near $\bar{c}_j$ can be considered as static and thus would not be assigned a different codeword at later iterations of the evolution process? In (Tsai et al., 2007), we presented a fast pattern reduction algorithm for k-means and k-means-based clustering algorithms. As the simulation results in (Tsai et al., 2007) showed, the removal rate α has a strong impact on the effectiveness of convergence. However, we assume there that the distances between all the patterns and codewords are normally distributed. The detection module uses the mean μ and standard deviation σ for finding the top 50% of the patterns near the codeword and thus can be removed. For instance, if the distance $\|x_i - \bar{c}_j\|$ between the pattern x_i and codeword $\bar{c}_j$ is smaller than μ, then it can be considered as among the top 50% near the codeword $\bar{c}_j$. If $\|x_i - \bar{c}_j\| < \mu + \sigma$, the pattern x_i can be considered as among the top 84% near the codeword $\bar{c}_j$.

If the distances are not normally distributed, then PREGLA has to count on other statistical measures such as median for locating the patters near the codeword. In other words, it may affect the running time required to find the patterns to be compressed and removed and thus the running time of PREGLA. As for the quality of the end results, it depends, to a large extent, on the GLA and GLA-based algorithms themselves.

Besides, the reduction rate at each iteration is very important and may even influence the end result. The higher the reduction rate, the more the computation time can be reduced. The problem is that it may deteriorate the end result (Tsai et al., 2007). In this paper, we present a different strategy for pattern reduction – a slow reduction method for keeping the search diversity of GLA. With each codeword are associated a removal bound Θ_t and a radius D^j_t (line 11 of Fig. 3) at iteration t. Θ_t is set to 10% for the first time the pattern reduction process is applied. Accordingly, D^j_t is set to the distance from the codeword $\bar{c}_j$ within which the top 10% of patterns in B_j are located. Then, from the second time onwards, $\Theta_{t+1} = \Theta_t \times 1.8$. In other words, the removal bound is enlarged by a factor of 1.8 each time the pattern reduction process is applied but with an upper limit of 80%, which is the same as that used in (Tsai et al., 2007). As a consequence, not only can the proposed algorithm PREGLA significantly reduce the computation time compared to that of traditional GLA and GLA-based algorithms alone, but it can also provide a more stable reduction strategy for keeping the quality of the end result.

3.2.3. Removal module

After the detection module finds the patterns that may be removed, the removal module will store them away. The detection module uses a set R to collect all the removed patterns (line 11 of Fig. 3). Initially, R is empty. Then, after the detection and removal process of PREGLA, some patterns will be removed from $(X - R)$ and added to R (the compression process of PREGLA). Consequently, the removed patterns need no longer be considered by the partitioning, determination, and detection processes at later iterations in the evolution process. In other words, every time when the removal module deletes some patterns, it will reduce the total number of patterns that the other three processes need to compute.

It is worth mentioning that PREGLA differs from other fast GLA algorithms mainly in two respects. First, PREGLA uses the removal module to remove the static patterns from the input vector set for saving the computation time, which is fundamentally different from fast GLA algorithms that remove codewords from the codebook for reducing the computation time. Second, the idea of compression used by PREGLA is not orthogonal to other fast GLA algorithms. Eventually, we will show that PREGLA can be combined with other fast GLA algorithms to make them even faster.

3.3. An example

In this section, we present a simple example to illustrate how PREGLA works. As Fig. 4 shows, there are three patterns x_1, x_3,

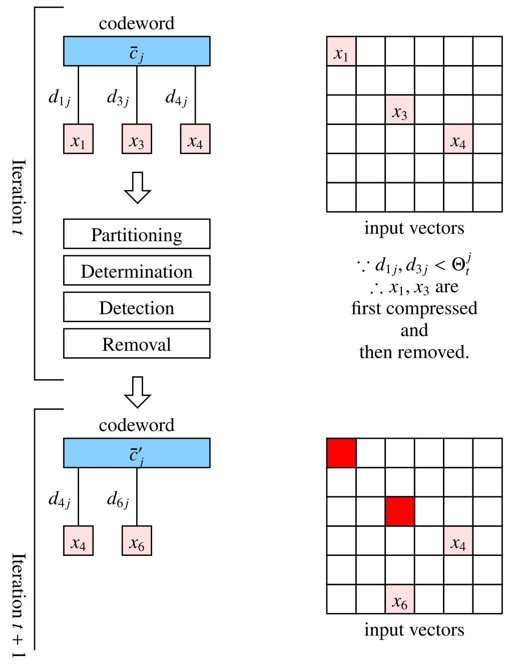


Fig. 4. A simple example illustrating how PREGLA works. See the text for more detailed explanation.

and x_4 that belong to the codeword $\bar{c}_j$ at iteration t. That is, $\bar{c}_j = (x_1 + x_3 + x_4)/3$. After the process of partitioning, determination, detection, and removal, the patterns x_1 and x_3 are removed from the input vector set because the distances between these two patterns and the codeword $\bar{c}_j$ are smaller than the radius D_t^j and thus have a smaller probability to be assigned to other codewords. At iteration $t+1$ and onwards, the removed patterns x_1 and x_3 will no longer involve in the codebook generation process that, as we have previously stated, includes partitioning, determination, detection, and removal. In other words, $\bar{c}_j' = (x_4 + x_6)/2 + (x_1 + x_3)/2$ where the average value $(x_1 + x_3)/2$ of the patterns x_1 and x_3 has already been computed at iteration t and will eventually bypass all the steps (computations) in the codebook generation process at iteration $t+1$. This is how PREGLA reduces the computation time; that is, by removing all the redundant computations on the fly so that they are not involved in the computation at the later iterations of the evolution process.

3.4. Performance analysis

The underlying ideas of GLA and k-means are essentially the same. As such, the time complexity of GLA (Fränti et al., 1998; Kaukoranta et al., 2000) and k-means (Jain et al., 1999; Day, 1992) are the same. In (Fränti et al., 1998, Kaukoranta et al., 2000), the authors pointed out that the time complexity of GLA is $O(gnm)$, where g is the number of iterations, n is the number of input vectors, and m is the number of clusters (codewords). If the number of dimensions of each input vector is taken into consideration, the time complexity will become $O(gnm\ell)$, where ℓ is the number of dimensions of the input vectors.

In the ideal case, PREGLA can reduce the time complexity of GLA from $O(gnm\ell)$ down to $O(nm\ell)$, i.e., independent of the number of iterations g. This can be easily proved by letting Δ $(0 < \Delta < 1)$ be a constant indicating the percentage of patterns retained at each iteration. In other words, $1 - \Delta$ is the percentage of patterns removed at each iteration. Then,

$$\begin{aligned}\sum_{i=0}^{g-1} \Delta^i nm\ell &= nm\ell \sum_{i=0}^{g-1} \Delta^i \\ &\leqslant nm\ell \sum_{i=0}^{\infty} \Delta^i \\ &= nm\ell \frac{1}{1-\Delta} \\ &= O(nm\ell).\end{aligned} \tag{3}$$

However, our experimental result shows that in practice, a removal bound is required to limit the number of patterns that can be compressed and removed by the PREGLA algorithm. One of the purposes of this bound is to reduce the impact the noises in the input data may have on the accuracy rate of the clustering results. For the results given in this paper, the bound has been set to 80% because we do not know exactly how GLA or GLA-based algorithms converge. If the PREGLA algorithm removes too many patterns at each iteration, the accuracy rate will be decreased. So all we can claim about the time complexity of PREGLA is that it will fall somewhere between $O(nm\ell)$ and $O(gnm\ell)$. In other words, the time complexity of PREGLA is bound from above by $O(gnm\ell)$ and from below by $O(nm\ell)$. In the best case, when the PREGLA algorithm is started at the very first iteration and the removal bound is set to 100%, the time complexity of PREGLA will be $O(nm\ell)$. In the worst case, the PREGLA algorithm is never started or the removal bound is set to 0%, it will fall back into GLA, and the time complexity will be $O(gnm\ell)$. In other words, the time complexity of PREGLA depends on (1) the iteration at which the PREGLA algorithm starts, (2) the number of patterns removed at each iteration, and (3) the removal bound. Our simulation results (to be discussed in Section 4) demonstrated that PREGLA can significantly decrease the computation time of GLA when the removal bound is set to 80%. The same results further showed that if the removal bound is set to an even larger value at the right time, we can reduce the time complexity of GLA to approach that of the ideal case, i.e., $O(nm\ell)$.

Table 1
Dataset for benchmarks.

	Name of images		
	Peppers	Lena	Baboon
Dataset	Parrot	Airplane	Island

Table 2
Enhancement of the PSNR of the PR enhanced vs. the original GLA and GLA-based algorithms alone, in percentage.

m	Δ_E				
	PREGLA	PRECD	PRENPS	PREFVQ	PRELC
64	−0.54	−0.07	−0.44	−0.51	−0.51
128	−0.56	−0.03	−0.46	−0.53	−0.33
256	−0.55	−0.03	−0.51	−0.48	−0.23
512	−0.63	−0.29	−0.63	−0.63	−0.31
1024	−0.49	−0.25	−0.49	−0.46	−0.03
2048	−0.48	−0.37	−0.51	−0.56	−0.01

Table 3
Enhancement of the running time of CD, NPS, FVQ, and LC with respect to GLA, in percentage.

m	T_{GLA} (in seconds)	δ_T			
		CD	NPS	FVQ	LC
64	32.59	−82.66	−50.42	−22.18	−2.84
128	56.89	−87.17	−59.47	−31.48	−26.90
256	117.70	−89.70	−65.77	−38.70	−46.50
512	199.67	−88.68	−65.34	−29.91	−48.58
1024	475.63	−90.54	−71.38	−37.41	−68.12
2048	947.43	−90.52	−64.68	−37.15	−76.86

Table 4
Enhancement of the running time of the PR enhanced vs. the original GLA and GLA-based algorithms alone, in percentage.

m	Δ_T				
	PREGLA	PRECD	PRENPS	PREFVQ	PRELC
64	−63.96	−29.45	−48.79	−57.15	−60.63
128	−71.03	−37.17	−55.57	−66.94	−58.23
256	−75.73	−41.75	−59.06	−71.56	−52.63
512	−71.95	−43.43	−56.02	−73.61	−46.57
1024	−77.39	−46.65	−49.95	−74.90	−36.83
2048	−77.98	−48.74	−39.44	−75.58	−30.60

4. Experimental results

Let us suppose that the sizes of the input (original) and reconstructed images are indicated by $w \times h$.[2] Let us further suppose that v_{ij} and $\tilde{v}_{ij}$ indicate, respectively, the pixel values at row i and column j of the input image and the reconstructed image. Then, the mean-squared-error (MSE) of the reconstructed image is as defined in Eq.

[2] w and h are, respectively, the width and height of an image.

(4), and the quality of the reconstructed image can be measured by the Peak Signal-to-Noise Ratio (PSNR) as defined in Eq. (5).

$$\mathrm{MSE} = \frac{1}{w \times h} \sum_{i=1}^{w} \sum_{j=1}^{h} \left(\tilde{v}_{ij} - v_{ij}\right)^2, \tag{4}$$

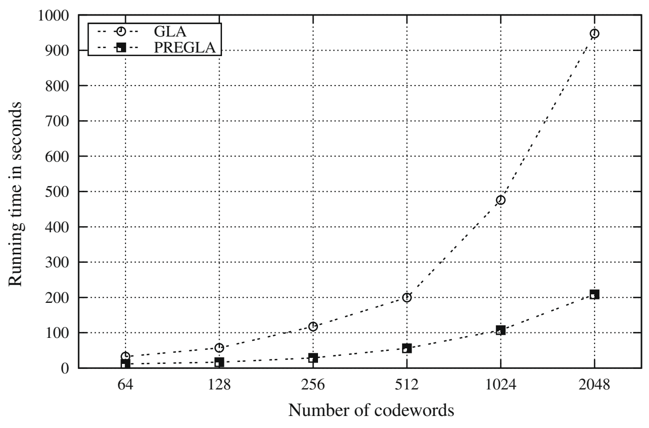
$$\mathrm{PSNR} = -10\log_{10} \frac{\mathrm{MSE}}{255^2}. \tag{5}$$

In what follows, let us suppose that E_ψ indicates the PSNR of using ψ for VQ; $E_{\mathrm{PRE}\psi}$ the PSNR of using PREψ for VQ. Let us further suppose that T_ψ indicates the computation time of ψ; $T_{\mathrm{PRE}\psi}$ the computation time of *PREψ*. Then, Δ_E and Δ_T are computed as follows:

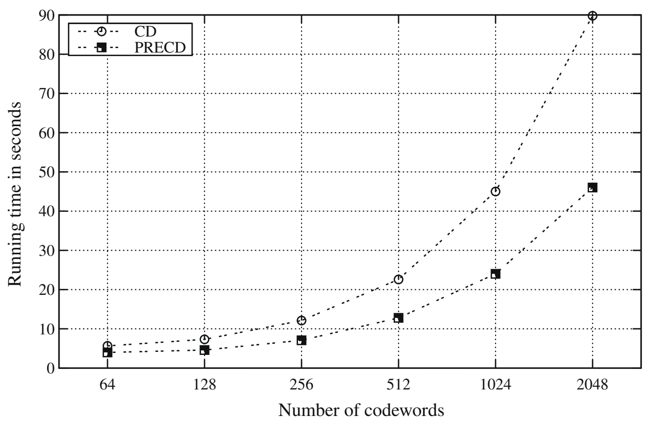
$$\Delta_E = \frac{E_{\mathrm{PRE}\psi} - E_\psi}{E_\psi} \times 100\%, \tag{6}$$

$$\Delta_T = \frac{T_{\mathrm{PRE}\psi} - T_\psi}{T_\psi} \times 100\%, \tag{7}$$

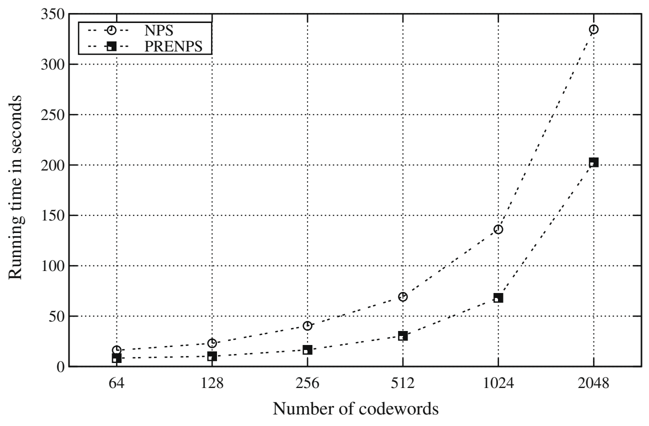
where ψ is to be replaced by one of the following algorithms: GLA, CD, NPS, FVQ, and LC, to be discussed shortly.



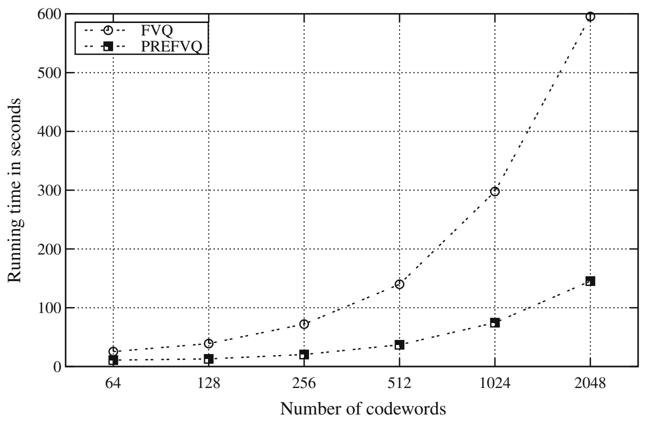
(a) Standard GLA versus PREGLA



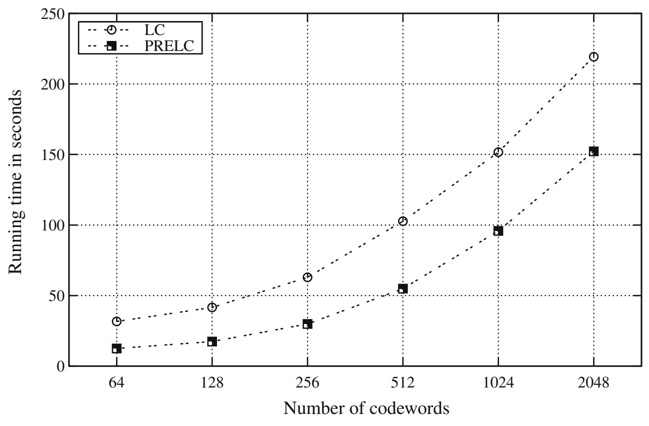
(b) CD versusPRECD



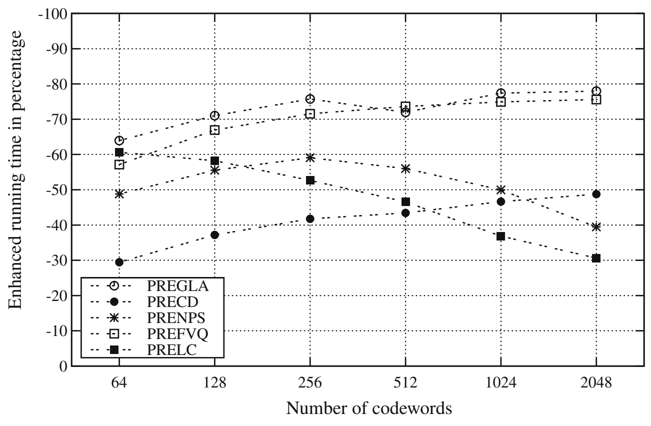
(c) NPS versus PRENPS



(d) FVQ versus PREFVQ



(e) LC versus PRELC



(f) Enhancement of the computation time in percentage.

Fig. 5. Simulation results of PR enhanced versus original GLA and GLA-based algorithms.

4.1. Datasets and parameter settings

The set of real images, all of which are of size 512×512 and in 8-bit grayscale, as given in Table 1, is used in the evaluation of the performance of the proposed algorithm. All the simulations are carried out for 30 runs. The parameter settings of the simulation are as follows:

- The sampling size is set to 10% of the input vectors.
- The pattern reduction process starts at iteration 2.
- The maximum removal bound is set to 80%.
- The removal rate α is set to 80%, and the removal bound Θ_t starts at 10% and is kept increasing by 80% for each iteration, i.e., $\Theta_1 = 0\%$ for iteration 1, $\Theta_2 = 10\%$ for iteration 2, $\Theta_3 = (10 \times 1.8)\%$ for iteration 3, $\Theta_4 = (10 \times 1.8^2)\%$ for iteration 4, and so on.

4.2. Simulation results

To evaluate the performance of the proposed algorithm, we compare the PR enhanced version of Generalized Lloyd Algorithm (GLA), Codeword Displacement (CD) (Lai et al., 2008), Nearest Partition Set Search (NPS) (Qian, 2006), Fast Vector Quantization Algorithm (FVQ) (Qian, 2004), and Law of Cosines (LC) (Pan et al., 2004) with the algorithms themselves. Shown in Tables 2 and 4 are the simulation results in terms of two metrics: PSNR and the computation time.

As the simulation results in Table 2 show, PSNR of all the PR enhanced versions is deteriorated by no more than 0.63% for all the algorithms evaluated. In addition, Table 3 shows the enhancement of the running time of fast GLA algorithms including CD, NPS, FVQ and LC with respect to the GLA. Note that δ_T in Table 3 is computed as follows:

$$\delta_T = \frac{T_\Phi - T_{\mathrm{GLA}}}{T_{\mathrm{GLA}}} \times 100\%, \tag{8}$$

where Φ is to be replaced by either CD, NPS, FVQ, or LC.

However, as the simulation results in Table 4 show, PREGLA can reduce the computation time from 63.96% up to 77.98% compared to GLA, from 29.45% up to 48.74% compared to CD (Lai et al., 2008), from 39.44% up to 59.06% compared to NPS (Qian, 2006), from 57.15% up to 75.58% compared to FVQ (Qian, 2004) and from 30.60% up to 60.63% compared to LC (Pan et al., 2004). In other words, the simulation results show that PREGLA can reduce the computation time from 29.45% up to 77.98% for all the algorithms evaluated.

Fig. 5 compares the computation time of the PR enhanced GLA and GLA-based algorithm with those of the original algorithms themselves. In general, the amount of computation time that can be reduced depend, to a large extent, on the size of the problem. The simulation results indicate that the larger the problem, the better the performance of the proposed algorithm. However, as Fig. 5f shows, the performance of PR enhanced LC (Pan et al., 2004) and NPS (Qian, 2006) are not as good as the others. This is due to the fact that these two algorithms by themselves can reduce the computation time of GLA by 76.86% and 71.38%, respectively. Even so, the proposed algorithm can *further* reduce the computation time of both algorithms from 30.60% up to 60.63%.

5. Conclusion

This paper gives a detailed description of how PREGLA (Pattern Reduction Enhanced GLA) can be used to enhance the performance of GLA and GLA-based algorithms. All of our simulation results showed that the proposed algorithm can significantly reduce the computation time of standard GLA and other fast GLA-based algorithms in generating the codebook, especially when the size of the images or codebooks is large. This can be easily justified because the PREGLA algorithm first compresses and then removes most of the patterns at early iterations in its revolution process, thus effectively reducing the computation time of the GLA and GLA-based algorithms at later iterations. Besides, the refining module uses sampling to provide a better initial pattern set to start with and thus a more stable end result.

Acknowledgements

The authors would like to thank the editor and all the reviewers for their constructive comments and suggestions on the paper. This work was supported in part by National Science Council, Taiwan, ROC, under Contract No. 95-2221-E-006-509-MY2.

References

Abdel-Galil, T.K., Hegazy, Y.G., Salama, M.M.A., Bartnikas, R., 2005. Fast match-based vector quantization partial discharge pulse pattern recognition. IEEE Trans. Instrum. Measure. 54 (1), 3–9.
Barszcz, M., Chen, W., Boulianne, G., Kenny, P., 2000. Tree-structured vector quantization for speech recognition. Comput. Speech Language 14 (3), 227–239.
Bradley, P.S., Fayyad, U.M., 1998. Refining initial points for *k*-means clustering. In: Proc. of the Internat. Conf. on Machine Learning, pp. 91–99.
Buzo, A., Augustine, J., Gray, H., Gray, R.M., Markel, J.D., 1980. Speech coding based upon vector quantization. IEEE Trans. Acoust. Speech Signal Process. 28 (5), 562–574 (see also IEEE Trans. Signal Process.).
Cardinal, J., 2000. A lagrangian optimization approach to complexity-constrained TSVQ. IEEE Signal Process. Lett. 7 (11), 304–306.
Chang, C.-C., Lin, I.-C., 2005. Fast search algorithm for vector quantisation without extra look-up table using declustered subcodebooks. IEE Proc. Vision Image Signal Process. 152 (5), 513–519.
Chang, C.C., Wu, W.C., 2007. Fast planar-oriented ripple search algorithm for hyperspace VQ codebook. IEEE Trans. Image Process. 16 (6), 1538–1547.
Chang, C.C., Li, Y.C., Yeh, J.B., 2006. Fast codebook search algorithms based on tree-structured vector quantization. Pattern Recognition Lett. 27 (10), 1077–1086.
Chena, H.-N., Chung, K.-L., 2005. Improved adaptive vector quantization algorithm using hybrid codebook data structure. Real-Time Imaging 11 (4), 270–281.
Chung, K.-L., Lai, J.-Y., 2004. An efficient law-of-cosine-based search for vector quantization. Pattern Recognition Lett. 25 (14), 1613–1617.
Day, W., 1992. Complexity theory: An introduction for practitioners of classification. In: Arabie, P., Hubert, L. (Eds.), Clustering and Classification. World Scientific Publishing Co. Inc., River Edge, NJ.
Ebrahimi-Moghadam, A., Shirani, S., 2005. Progressive scalable interactive region-of-interest image coding using vector quantization. IEEE Trans. Multimedia 7 (4), 680–687.
Fränti, P., Kivijärvi, J., Nevalainen, O., 1998. Tabu search algorithm for codebook generation in vector quantization. Pattern Recognit. 31 (8), 1139–1148.
Giuseppe, P., Marco, R., 2001. The enhanced LBG algorithm. Neural Networks 14 (9), 1219–1237.
Jain, A., Murty, M., Flynn, P., 1999. Data clustering: A review. ACM Comput. Surveys 31 (3).
Kaukoranta, T., Fränti, P., Nevalainen, O., 2000. A fast exact GLA based on code vector activity detection. IEEE Trans. Image Process. 9 (8), 1337–1342.
Kim, M.Y., Kleijn, W.B., 2004. KLT-based adaptive classified VQ of the speech signal. IEEE Trans. Speech Audio Process. 12 (3), 277–289.
Kugler, M., Lopes, H.S., 2003. Using a chain of LVQ neural networks for pattern recognition of EEG signals related to intermittent photic-stimulation. In: Proc. of the National Brazilian Symposium on Neural Networks, pp. 173–177.
Kusumoputro, B., Budiarto, H., Jatmiko, W., 2002. Fuzzy-neuro LVQ and its comparison with fuzzy algorithm LVQ in artificial odor discrimination system. ISA Trans. 41 (4), 395–407.
Laha, A., Pal, N.R., Chanda, B., 2004. Design of vector quantizer for image compression using self-organizing feature map and surface fitting. IEEE Trans. Image Process. 13 (10), 1291–1303.
Lai, J.Z.C., Liaw, Y., 2004. Fast-searching algorithm for vector quantization using projection and triangular inequality. IEEE Trans. Image Process. 13 (12), 1554–1558.
Lai, J.Z.C., Liaw, Y.C., Liu, J., 2008. A fast VQ codebook generation algorithm using codeword displacement. Pattern Recognit. 41 (1), 315–319.
Laskaris, N.A., Fotopoulos, S., 2004. A novel training scheme for neural-network-based vector quantizers and its application in image compression. Neurocomputing 61 (1), 421–427.
Linde, Y., Buzo, A., Gray, R.M., 1980. An algorithm for vector quantizer design. IEEE Trans. Comm. 28 (1), 84–95.
Mielikainen, J., 2002. A novel full-search vector quantization algorithm based on the law of cosines. IEEE Signal Process. Lett. 9 (6), 175–176.

Pan, S.M., Cheng, K.S., 2007. An evolution-based tabu search approach to codebook design. Pattern Recognit. 40 (2), 476–491.

Pan, J.S., Lu, Z., Sun, S., 2003. An efficient encoding algorithm for vector quantization based on subvector technique. IEEE Trans. Image Process. 12 (3), 265–270.

Pan, Z., Kotani, K., Ohmi, T., 2004. An improved full-search-equivalent vector quantization method using the law of cosines. IEEE Trans. Signal Process. Lett. 11 (2), 247–250.

Patané, G., Russo, M., 2001. The enhanced LBG algorithm. Neural Networks 14 (9), 1219–1237.

Qian, S.E., 2004. Hyperspectral data compression using a fast vector quantization algorithm. IEEE Trans. Geosci. Remote Sensing 42 (8), 1791–1798.

Qian, S.E., 2006. Fast vector quantization algorithms based on nearest partition set search. IEEE Trans. Image Process. 15 (8), 2422–2430.

Ramasubramanian, V., Paliwal, K., 1992. Fast *k*-dimension tree algorithm for nearest neighbour search with application to vector encoding. IEEE Trans. Signal Process. 40 (3), 518–531.

Shen, F., Hasegawa, O., 2006. An adaptive incremental LBG for vector quantization. Neural Networks 19 (5), 694–704.

Shen, G., Zeng, B., Liou, M.-L., 2003. Adaptive vector quantization with codebook updating based on locality and history. IEEE Trans. Image Process. 12 (3), 283–295.

Trentin, E. and Gori, M., 2002. Face recognition using vector quantization histogram method. In: Proc. of the 2002 Internat. Conf. on Image Processing, pp. 22–25.

Tsai, C.-W., Yang, C.-S., Chiang, M.-C., 2007. A time efficient pattern reduction algorithm for *k*-means based clustering. In: IEEE Internat. Conf. on Systems, Man and Cybernetics, pp. 504–509.

Vasuki, A., Vanathi, P., 2006. A review of vector quantization techniques. IEEE Potentials 25 (4), 39–47.

Winger, L.L., 2001. Linearly constrained generalized Lloyd algorithm for reduced codebook vector quantization. IEEE Trans. Signal Process. 49 (7), 1501–1509.

Xiong, H., Swamy, M.N.S., Ahmad, M.O., 2004. Competitive splitting for codebook initialization. IEEE Signal Process. Lett. 11 (5), 474–477.

Yang, S.-B., 2004. Variable-branch tree-structured vector quantization. IEEE Trans. Image Process. 13 (9), 1275–1285.