Fast closest codeword search algorithm for vector quantisation

C.-H. Lee
L.-H. Chen

Indexing terms: Codebook design, Mean value, Variance, Vector quantisation

Abstract: One of the most serious problems for vector quantisation is the high computational complexity of searching for the closest codeword in the codebook design and encoding phases. The authors present a fast algorithm to search for the closest codeword. The proposed algorithm uses two significant features of a vector, mean value and variance, to reject many unlikely codewords and saves a great deal of computation time. Since the proposed algorithm rejects those codewords that are impossible to be the closest codeword, this algorithm introduces no extra distortion than conventional full search method. The results obtained confirm the effectiveness of the proposed algorithm.

1 Introduction

Vector quantisation (VQ) is a very efficient approach to low-bit-rate image compression [1]. It is defined as a mapping Q from a k-dimensional Euclidean space R^k to a finite subset Y of R^k. That is

$$Q: R^k \rightarrow Y$$

where $Y = \{y_i | i = 1, 2, \ldots, N\}$ is called the codebook, and N is the size of the codebook. Each $y_i = (y_{i1}, y_{i2}, \ldots, y_{ik})$ in Y is called a codeword. For the compression purpose, a VQ consists of three phases: codebook design phase, encoding phase and decoding phase. The objective of codebook design is to find a codebook Y which contains the most representative codewords. This codebook will be used by encoder and decoder. In the encoding phase, the encoder designs a mapping Q and assigns an index i to each input vector $x = (x_1, x_2, \ldots, x_k)$ with $Q(x) = y_i$. In this paper, we will consider the mapping Q, which is designed to map x to y_i with y_i satisfying the following condition

$$d^2(x, y_i) = \min_j d^2(x, y_j) \quad \text{for } j = 1, 2, \ldots, N \tag{1}$$

where $d^2(x, y_j)$ is the distortion of representing the input vector x by the codeword y_j, as measured by the squared Euclidean distance, i.e.

$$d^2(x, y_j) = \sum_{n=1}^{k} (x_n - y_{jn})^2 \tag{2}$$

When encoding an image, the encoder first divides the image into several blocks, usually in square. Each block x containing k entities is considered as a k-dimensional vector. Hence, for each input vector x (i.e. a block), the encoder only needs to transmit or store the index i assigned to x. The decoder has the same codebook as the encoder. In the decoding phase, for each index i, the decoder merely performs a simple table look-up operation to obtain y_i, and then uses y_i to represent the input vector x.

From the above description, we see that the compression ratio is determined by the codebook size and the vector dimension; the distortion is dependent on the codebook size and selection of codewords. Therefore, designing a good codebook is the main task of VQ. Many algorithms for codebook design have been proposed [1–3]. Among these algorithms, the most popular one was developed by Linde, Buzo and Gray [1, 2] and is referred to as the LBG algorithm. This algorithm iteratively minimises the total distortion of representing the training vectors by their corresponding codewords. It divides the training vectors into several classes, each class is represented by the centroid of that class. The set of all centroids forms a codebook, and each centroid is referred to as a codeword. The algorithm is iterative, it first takes an initial codebook Y_0 with predetermined size N and then starts iterating. In each iteration, for each training vector x, it searches the current codebook exhaustively to find the closest codeword y_i and assign x to class i. After all training vectors have been classified, the distortion between the set of training vectors and their corresponding codewords is calculated. The distortion difference between the current iteration and the previous iteration is then calculated. The ratio of the distortion difference to the distortion of the current iteration is checked to see if it is greater than a preset value ε. If it is true, y_i is then replaced by the centroid of the new class i, and a new iteration starts; otherwise, the algorithm stops. Note that the LBG algorithm uses a full codebook search to find the closest codeword for each training vector. If the codebook size is N, the full codebook search (i.e. to

Paper 1140K (E4), first received 15th February 1993 and in revised form 5th January 1994

The authors are with the Department of Computer and Information Science, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu, Taiwan 30050, Republic of China

This research was supported in part by the National Science Council of ROC under contract NSC-82-0408-E009-063

evaluate eqns. 1 and 2, requires Nk multiplications, $N(2k-1)$ additions, and $N-1$ comparisons for each k-dimensional training vector).

As mentioned above, N determines the accuracy of VQ. The larger N is, the more accurate VQ. However, when N is large, the computational complexity problem for full codebook search will occur. This problem is critical in the codebook design and encoding of VQ. To avoid such an exhaustive search through the codebook, many fast algorithms [4–10] have been proposed. These algorithms reduce the computational complexity by first performing some simple tests before computing the distortion between the training vector and each codeword, and then rejecting those codewords which fail in the tests. In the next Section, we will briefly review some of these algorithms.

2 Some existing fast closest codeword search algorithms

2.1 Partial distortion elimination algorithm

The partial distortion elimination (PDE) algorithm [4] allows early termination of the distortion calculation between a training vector and a codeword by introducing a premature exit condition in the search process. For each training vector x, the algorithm first calculates the distortion between x and an arbitrary codeword and takes this distortion as the current minimum distortion d^2_{min}. Then, for any other codeword $y_i = (y_{i1}, y_{i2}, \ldots, y_{ik})$, if there exists $q < k$ with the accumulated distortion for the first q samples in eqn. 2 larger than the current minimum distortion d^2_{min}, i.e.

$$\sum_{n=1}^{q} (x_n - y_{in})^2 \geq d^2_{min}$$

this algorithm stops computing the distortion for codeword y_i and begins trying the next codeword. This will reduce $(k-q)$ multiplications and $2(k-q)$ additions. Simulations [4] indicate that the PDE method can reduce a good number of multiplication operations and addition operations in the search process, and only increases some comparison operations.

2.2 Partial search partial distortion algorithm

The partial search partial distortion (PSPD) algorithm [5] builds up a partial codebook based on the mean value m_x of a k-dimensional training vector $x = (x_1, x_2, \ldots, x_k)$, in which m_x is defined as

$$m_x = \text{integer part of} \left[\frac{1}{k} \sum_{j=1}^{k} x_j + 0.5 \right]$$

The algorithm then uses the PDE method to search the partial codebook for the closest codeword.

The PSPD algorithm first calculates the mean values of all codewords and sorts the codebook according to increasing order of the codeword means. For each training vector, it then finds the codeword y_p with minimum mean difference to the training vector. The codewords with mean differences to y_p less than a predetermined threshold T form the partial codebook. The PDE method is then employed to find the closest codeword in this partial codebook. Experimental results [6] show that the execution time of the PSPD algorithm is about 12% of that required by the LBG algorithm. However, sometimes the closest codeword may not be located in the partial codebook, this will introduce more distortion than the LBG algorithm.

2.3 Fast nearest neighbour search algorithm

The fast nearest neighbour search (FNNS) algorithm [6] uses the triangle inequality to reject a great many unlikely codewords. For a vector x, it first finds a probably nearby codeword y_l with distortion $d^2(x, y_l)$. This algorithm then eliminates those codewords which are impossible to be the closest codeword, based on the triangle inequality and a precomputed table which contains the distances of all pairs of codewords. That is, for each codeword y_j, if

$$d(y_j, y_l) > 2d(x, y_l)$$

through the triangle inequality, we have

$$d(x, y_j) + d(x, y_l) \geq d(y_j, y_l) > 2d(x, y_l)$$

The above inequality can be reduced to be

$$d(x, y_j) > d(x, y_l)$$

Therefore, those codewords with distances to y_l larger than $2d(x, y_l)$ will be eliminated from consideration to be a candidate of the closest codeword. Simulations show that high saving rate over conventional full codebook search method can be achieved. However, this algorithm requires a table of size $N^2/2$ to store the distances of all pairs of codewords. When N is large, the memory requirement is a serious problem.

2.4 Equal-average nearest neighbour search algorithm

The closest codeword search problem in vector quantisation is a nearest neighbour search (NNS) problem which can be stated as: given a set Y, of N prototypes in a k-dimensional space R^k, for a query point $x \in R^k$, determine which prototype is closest to x. Guan *et al.* [7] proposed an equal-average nearest neighbour search (ENNS) algorithm which uses hyperplanes orthogonal to the central line l to partition the search space. Each coordinate value of any point $p = (p_1, p_2, \ldots, p_k)$ on l has the same value (i.e. $p_i = p_j$, $i, j = 1, 2, \ldots, k$). Each point on a fixed hyperplane H, which is orthogonal to the central line l and intersects l at point $L_H = (m_H, m_H, \ldots, m_H)$, will have the same mean value m_H, such a hyperplane is called an equal average hyperplane. For an input vector $x = (x_1, x_2, \ldots, x_k)$, the algorithm first calculates its mean value m_x with $m_x = (1/k) \sum_{j=1}^{k} x_j$. The algorithm then finds the codeword y_p which has the minimum mean difference to x and calculates the distance r_p between x and y_p. It is obvious that any other codeword which is closer to x than y_p has to be located inside the hypersphere centred at x with radius r_p. By projecting the hypersphere on l, two boundary projection points, $L_{max} = (m_{max}, m_{max}, \ldots, m_{max})$ and $L_{min} = (m_{min}, \ldots, m_{min}, \ldots, m_{min})$ on l can be found, where

$$m_{max} = m_x + \frac{r_p}{\sqrt{(k)}} \tag{3}$$

and

$$m_{min} = m_x - \frac{r_p}{\sqrt{(k)}} \tag{4}$$

The hypersphere can be bounded by two equal-average hyperplanes with mean values m_{max} and m_{min}. Hence, it is only necessary to search those codewords with mean values ranging from m_{min} to m_{max}. Fig. 1 shows the geometric interpretation of the method for a two-

dimensional case, the search area is bounded by two lines l_1 and l_2, which are perpendicular to the central line l at L_{max} and L_{min}, respectively.

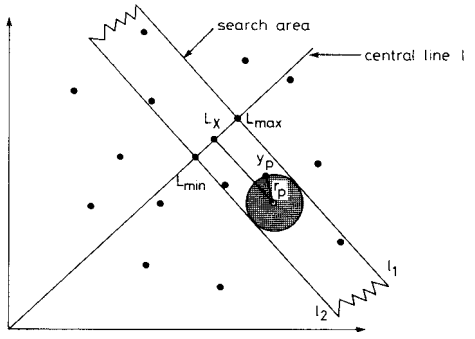


Fig. 1 *Example of ENNS algorithm for 2-dimensional case*
○ input vector x
● codewords

As discussed previously, the ENNS algorithm uses mean value as a feature to reject unlikely codewords and thus saves much of computation time. However, two vectors with the same mean value may have a large distance. For example, one vector represents an almost homogeneous block (i.e. entities in the vector are almost the same); the other represents an edge block (i.e. some entities will be larger than others). The distance between these two types of vectors will be large. To treat this phenomenon, we propose a new search method to reduce the search area of the ENNS algorithm. Since the variance of a vector is a simple measure to detect whether a vector is homogeneous, the proposed method uses both the mean value and the variance of a vector as two significant features to search for the closest codeword.

3 The proposed algorithm

In this Section, we will present the proposed algorithm, which uses the mean value and the variance of a vector as two significant features to speed up the closest codeword search process. Before describing the algorithm, we will first give some definitions and a theorem.

Definition 1: Let $x = (x_1, x_2, \ldots, x_k)$ be a vector, define the mean value m_x of x and the variance V_x^2 as

$$m_x = \frac{1}{k}\sum_{j=1}^{k} x_j$$

and

$$V_x^2 = \sum_{j=1}^{k} (x_j - m_x)^2$$

Definition 2: Let l be a line on R^k, if any point $p = (p_1, p_2, \ldots, p_k)$ on l satisfies the following condition

$$p_1 = p_2 = \cdots = p_k$$

l is called the central line of R^k.

Let l be the central line of R^k and L_x be the projection point of x on l. It can be seen that $L_x = (m_x, m_x, \ldots, m_x)$.

By the above definitions, we have the following theorem.

Theorem 1: Let $x = (x_1, x_2, \ldots, x_k)$ be a vector and $y_i = (y_{i1}, y_{i2}, \ldots, y_{ik})$ be a codeword. If the distortion between x and y_i is defined to be the squared Euclidean distance, i.e.

$$d^2(x, y_i) = \sum_{j=1}^{k} (x_j - y_{ij})^2$$

then $V_x = d(x, L_x), \quad V_{yi} = d(y_i, L_{yi})$

and

$$d(x, y_i) \geq |V_x - V_{yi}| \tag{5}$$

The proof is given in Appendix 7.1. By Theorem 1, we obtain the following corollary immediately.

Corollary 1: Let x be a vector and d_{min}^2 be a known current minimum distortion of x represented by a certain codeword. For any codeword y_i, if $(V_x - V_{yi})^2 \geq d_{min}^2$, y_i will not be the closest codeword of x and it is unnecessary to calculate $d^2(x, y_i)$.

With the above theorem and corollary in hand, we now turn to describe the proposed algorithm. This algorithm consists of two steps. The first step is the same as the ENNS algorithm, the second is a new one. For a training vector x, the proposed algorithm first calculates the mean value m_x and the squared root of the variance V_x of x. The algorithm then finds the codeword y_p with the minimum mean difference to x, calculates the distance r_p between x and y_p and sets the current minimum distortion d_{min}^2 as r_p^2. For each codeword y_i, the algorithm checks if m_{yi} is between m_{min} and m_{max}, where

$$m_{min} = m_x - \frac{d_{min}}{\sqrt{(k)}} \quad \text{and} \quad m_{max} = m_x + \frac{d_{min}}{\sqrt{(k)}}$$

If the answer is no, codeword y_i is rejected without calculating the distortion $d^2(x, y_i)$. Otherwise, the second step is conducted. In the second step, if $(V_x - V_{yi})^2 \geq d_{min}^2$, the codeword y_i is rejected. If y_i is not rejected, the distortion $d^2(x, y_i)$ is calculated. If $d^2(x, y_i) < d_{min}^2$, the current minimum distortion d_{min}^2 is replaced by $d^2(x, y_i)$ and m_{min} and m_{max} are also updated.

Note that, in the ENNS algorithm, for any codeword y_i with m_{yi} between m_{min} and m_{max}, the distortion $d^2(x, y_i)$ is always evaluated. In contrast, the proposed algorithm will calculate $d^2(x, y_i)$ only when $(V_x - V_{yi})^2 \leq d_{min}^2$. This will avoid those codewords with mean values similar to m_x, but with variances very different from V_x^2 being considered to be the closest codeword of x. Hence, the proposed algorithm can reduce the search area and speed up the search process than the ENNS algorithm. Fig. 2 depicts the geometric interpretation of the proposed method for a two-dimensional case. Comparing Fig. 1 and Fig. 2, we can see that the search area, which is originally an area bounded by two lines l_1 and l_2 perpendicular to the central line l, has been reduced to be the two shaded squares.

A detailed description of how to apply the proposed algorithm to design a codebook is given below.

Step 0: Initialisation: Given N = codebook size, n = the number of training vectors, k = the vector dimension, Y_0 = initial codebook, ε = distortion threshold. Set iteration counter $c = 0$, initial total distortion $D_{-1} = \infty$.

Step 1: Compute the mean value of each codeword in the codebook Y_c, and sort Y_c according to increasing order of the codeword means, i.e. the sorted codebook Y_{sc}

is

$$Y_{sc} = \{y_i \mid m_{yi} \leqslant m_{y(i+1)} \quad 1 \leqslant i \leqslant N-1\}$$

Step 2: Compute the squared root of the variance V_{yi}, of each codeword y_i.

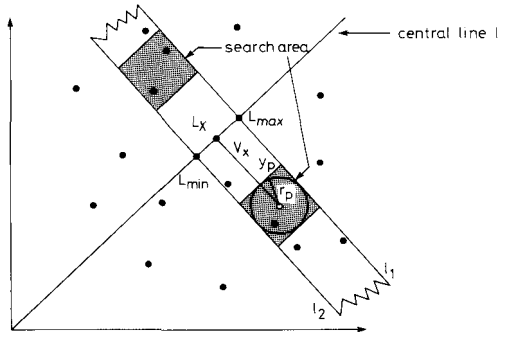


Fig. 2 *Example to illustrate proposed method for 2-dimensional case*
○ input vector x
● codewords

Step 3: For each training vector x_t, find the closest codeword $y_{i(t)}$ in the codebook Y_{sc} and assign x_t to class $i(t)$. The procedure includes the following substeps:

Step 3.1: Input a training vector $x_t = (x_{t1}, x_{t2}, \ldots, x_{tk})$, compute its mean value m_{xt} and its square root of variance V_{xt}.

Step 3.2: Find the codeword y_p which has the minimum mean difference to x_t (using binary search), i.e.

$$|m_{xt} - m_{yp}| \leqslant |m_{xt} - m_{yi}| \quad \text{for all } i \neq p$$

Set

$$d^2_{min} = d^2(x_t, y_p), \quad i(t) = p, \quad m_{max} = m_{xt} + \frac{d_{min}}{\sqrt{(k)}}$$

and

$$m_{min} = m_{xt} - \frac{d_{min}}{\sqrt{(k)}}$$

Step 3.3: Find the closest codeword $y_{i(t)}$ in Y_{sc} and assign x_t to class $i(t)$. The search procedure is as follows

```
Set d = 1
while(m_{p+d} < m_max or m_{p-d} > m_min)begin
  if(m_{p+d} < m_max)begin
    if((V_xt - V_{y(p+d)})^2 < d^2_min)begin
      if(d^2(x_t, y_{p+d}) < d^2_min)begin
        d^2_min = d^2(x_t, y_{p+d})
        m_max = m_xt + d_min/sqrt(k)
        m_min = m_xt - d_min/sqrt(k)
        i(t) = p + d
      end
    end
```

Table 1: Comparison of execution time (in seconds) for codebook design. Values in parentheses denote the ratio of execution time of the current algorithm to that of the LBG algorithm

Codebook size	Method	Design image			
		Lena	Peppers	Jet	Baboon
128	LBG	2815	2814	2668	2793
	ENNS	305 (0108)	284 (0.101)	320 (0.120)	847 (0.303)
	our method	266 (0.094)	242 (0.086)	269 (0.101)	771 (0.276)
256	LBG	5608	5622	5334	5583
	ENNS	509 (0.091)	472 (0.084)	535 (0.100)	1529 (0.274)
	our method	383 (0.068)	347 (0.062)	392 (0.073)	1313 (0.235)
512	LBG	11263	11263	10709	11212
	ENNS	865 (0.077)	812 (0.072)	921 (0.086)	2798 (0.250)
	our method	576 (0.051)	525 (0.047)	602 (0.056)	2292 (0.204)
1024	LBG	22732	22589	21679	22610
	ENNS	1525 (0.067)	1464 (0.065)	1596 (0.074)	5148 (0.228)
	our method	925 (0.041)	866 (0.038)	960 (0.044)	4001 (0.177)

Table 2: Comparison of execution time (in seconds) for image encoding

Codebook size	Method	Encoded image			
		Lena	Peppers	Jet	Baboon
128	Full search	140.9	140.1	125.1	137.5
	ENNS	14.1 (0.100)	15.0 (0.107)	14.3 (0.114)	37.1 (0.270)
	our method	11.5 (0.082)	11.9 (0.082)	11.4 (0.091)	32.2 (0.233)
256	LBG	278.6	279.5	249.7	275.0
	ENNS	23.6 (0.085)	27.4 (0.098)	23.9 (0.096)	69.4 (0.252)
	out method	17.2 (0.062)	18.8 (0.067)	17.4 (0.070)	54.1 (0.197)
512	LBG	557.6	558.1	499.3	547.8
	ENNS	40.8 (0.073)	47.9 (0.086)	43.4 (0.087)	130.9 (0.239)
	our method	26.7 (0.048)	30.2 (0.054)	26.7 (0.053)	96.2 (0.176)
1024	LBG	1108.5	1108.5	992.7	1087.9
	ENNS	71.4 (0.064)	89.3 (0.081)	79.2 (0.080)	249.2 (0.229)
	our method	40.7 (0.037)	51.1 ((0.046)	43.6 (0.044)	174.6 (0.160)

```
end
if(m_{p-d} > m_{min})begin
  if((V_{xt} - V_{y(p-d)})^2 < d^2_{min})begin
    if(d^2(x_t, y_{p-d}) < d^2_{min})begin
      d^2_{min} = d^2(x_t, y_{p-d})
      m_{max} = m_{xt} + d_{min}/√(k)
      m_{min} = m_{xt} - d_{min}/√(k)
      i(t) = p - d
    end
  end
end
end{of while}
```

Step 4: Compute the total distortion for the cth iteration D_c. Here D_c is defined to be

$$D_c = \sum_{t=1}^{n} d^2(x_t, y_{i(t)})$$

Step 5: If $(D_{c-1} - D_c)/D_c \leqslant \varepsilon$, halt with final codebook being Y_{sc}. Otherwise, go to Step 6.

Step 6: Compute the centroid of each class. The centroids are regarded as the codewords of a new codebook. Set $c = c + 1$ and to to Step 1 for next iteration.

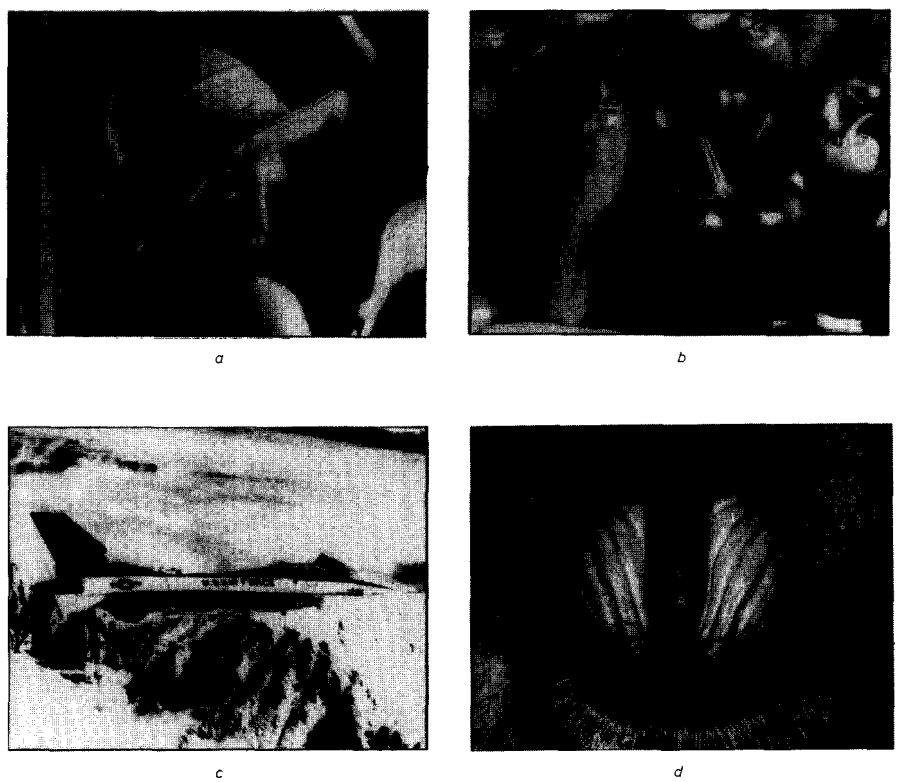
To speed up the codebook design procedure, the proposed algorithm needs two tables. One stores the mean values of all codewords, its size is N. The other stores the squared root of the variances, its size is also N. The total table size is $2N$, which is smaller than the FNNS algorithm. Note that the proposed algorithm does not produce any extra error than the LBG algorithm.

The encoder finds the closest codeword from a predesigned codebook for each input vector and then uses the codeword to represent the corresponding input vector. Therefore, the proposed algorithm can be used to find the closest codeword for each input vector to speed up the encoding process. The detail of the encoding procedure is similar to those in Step 3 of the codebook design algorithm described above.

4 Simulation results

To examine the efficiency of the proposed algorithm, we performed some experiments on a Sun SPARC-station-IPC using several 512 × 512 monochrome images with 256 grey levels. Each image is divided into 4 × 4 blocks, so that the training sequence contains 16384 16-dimensional vectors. The proposed algorithm was compared with the ENNS algorithm and the LBG algorithm in terms of the execution time required in codebook design and image encoding.

Table 1 shows the execution time required to design a codebook. The different images shown in Fig. 3 were used to design several different codebooks. Table 2 shows the time needed to encode an image given a predesigned codebook. In this simulation, the image Lena shown in Fig. 3*a* was used to design the codebook. The resulting codebook was then used to encode the four images (Lena, Peppers, Jet, and Baboon) shown in Fig. 3. From these two tables, we see that the proposed algorithm out-



a *b* *c* *d*

Fig. 3 *Four test images*
a Lena *b* Peppers *c* Jet *d* Baboon

performs the ENNS algorithm in both codebook design and image encoding.

5 Conclusions

A fast closest codeword search algorithm for vector quantisation has been proposed in this paper. This algorithm uses two significant features of a vector, mean value and variance, to reject a lot of unlikely codewords. It can speed up the search process in conventional VQ codebook design and encoding. The performance of the proposed algorithm has been evaluated in both codebook design and image encoding. The results obtained show that the proposed algorithm outperforms the ENNS algorithm and reduces a great deal of computation time required by the LBG algorithm. Furthermore, it is worth mentioning that the proposed algorithm does not introduce any extra error other than the LBG algorithm.

6 References

1 GRAY, R.M.: 'Vector quantisation', *IEEE ASSP Mag.*, 1984, pp. 4–29

2 LINDE, Y., BUZO, A., and GRAY, R.M.: 'An algorithm for vector quantiser design', *IEEE Trans.*, 1980, **C-28**, pp. 84–95

3 EQUITZ, W.H.: 'A new vector quantisation clustering algorithm', *IEEE Trans.*, 1989, **ASSP-37**, pp. 1568–1575

4 BEI, C.D., and GRAY, R.M.: 'An improvement of the minimum distortion encoding algorithm for vector quantisation', *IEEE Trans.*, 1985, **C-33**, pp. 1132–1133

5 HSIEH, C.H., LU, P.C., and CHANG, J.C.: 'Fast codebook generation algorithm for vector quantisation of images', *Pattern Recognition Lett.*, 1991, **12**, pp. 605–609

6 ORCHARD, M.D.: 'A fast nearest-neighbour search algorithm', *IEEE ICASSP*, 1991, pp. 2297–2300

7 GUAN, L., and KAMEL, M.: 'Equal-average hyperplane partitioning method for vector quantisation of image data', *Pattern Recognition Lett.*, 1992, **13**, pp. 693–699

8 CHENG, D.Y., and GERSHO, A.: 'A fast codebook search algorithm for nearest-neighbour pattern matching', *IEEE ICASSP*, 1986, pp. 6.14.1–6.14.4

9 SOLEYMANI, M.R., and MORGERA, S.D.: 'A high-speed search algorithm for vector quanitisation', *IEEE ICASSP*, 1987, pp. 45.6.1–45.6.3

10 PALIWAL, K.K., and RAMASUBRAMANIAN, V.: 'Effect of ordering the codebook on the efficiency of the partial distance search algorithm for vector quantisation', *IEEE Trans.*, 1989, **C-37**, pp. 538–540

7 Appendix

7.1 Proof of Theorem 1

By definition 1, we have $V_x = d(x, L_x)$ and $V_{yi} = d(y_i, L_{yi})$. By the triangle inequality, we can obtain

$$d(x, y_i) \geq d(x, L_{yi}) - d(y_i, L_{yi}) \tag{6}$$

Since L_x is the projection point of x on the central line l, thus

$$d(x, L_{yi}) \geq d(x, L_x) \tag{7}$$

By inequality eqn. 7, inequality eqn. 6 can be reduced to be

$$d(x, y_i) \geq d(x, L_x) - d(y_i, L_{yi}) = V_x - V_{yi} \tag{8}$$

Similarly, we can obtain

$$\begin{aligned} d(x, y_i) &\geq d(y_i, L_x) - d(x, L_x) \\ &\geq d(y_i, L_{yi}) - d(x, L_x) \\ &= V_{yi} - V_x \end{aligned} \tag{9}$$

Combining inequalities eqn. 8 and eqn. 9, we prove Theorem 1.