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 author presents 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 computational 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

\[
d^2(x, y_i) = \min d^2(x, y_j) \quad \text{for} \quad j = 1, 2, \ldots, N
\]

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

\[d^2(x, y) = \sum_{n=1}^{k} (x_n - y_n)^2\] (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 \( x_i \), it searches the current codebook exhaustively to find the closest codeword \( y_i \) and assigns \( x \) 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 \( c \). 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

This research was supported in part by the National Science Council of ROC under contract NSC-82-0408-E009-063
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 \( y_j \) with distortion \( d^2(x, y_j) \). 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

\[
d(y_j, y_k) > 2d(x, y_k)
\]

through the triangle inequality, we have

\[
d(x, y_j) + d(x, y_k) > d(y_j, y_k) > 2d(x, y_k)
\]

The above inequality can be reduced to be

\[
d(x, y_j) > d(x, y_k)
\]

Therefore, those codewords with distances to \( y_j \) larger than \( 2d(x, y_k) \) will be eliminated from consideration to be a candidate of the closest codeword. Simulations show that higher 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 \( x \), 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 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 \( \bar{L} = (m_y, m_y, \ldots, m_y) \), will have the same mean value \( m_y \), 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 \( m_x = \frac{1}{k} \sum_{j=1}^{k} x_j \). The algorithm then finds the codeword \( y_j \) which has the minimum mean difference to \( x \) and calculates the distance \( r_p \) between \( x \) and \( y_j \). It is obvious that any other codeword which is closer to \( x \) than \( y_j \) has to be located inside the hypersphere centred at \( x \) with radius \( r_p \). By projecting the hypersphere on \( l \), two boundary projection points, \( L_{\text{max}} = (m_{\text{max}}, m_{\text{max}}, \ldots, m_{\text{max}}) \) and \( L_{\text{min}} = (m_{\text{min}}, m_{\text{min}}, \ldots, m_{\text{min}}) \) on \( l \) can be found, where

\[
m_{\text{max}} = m_x + \frac{r_p}{\sqrt{k}} \tag{3}
\]

and

\[
m_{\text{min}} = m_x - \frac{r_p}{\sqrt{k}} \tag{4}
\]

The hypersphere can be bounded by two equal-average hyperplanes with mean values \( m_{\text{max}} \) and \( m_{\text{min}} \). Hence, it is only necessary to search those codewords with mean values ranging from \( m_{\text{min}} \) to \( m_{\text{max}} \). 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 $l_{\text{max}}$ and $l_{\text{min}}$, respectively.

![Fig. 1 Example of ENNS algorithm for 2-dimensional case](image)

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 inspect 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 $V_x = d(x, L_x), \quad V_y = d(y_i, L_y)$ and

$$d(x, y_i) \geq \sqrt{V_x - V_y}$$

(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 $d_{\text{min}}^2$ be a known current minimum distortion of $x$ represented by a certain codeword. For any codeword $y_i$, if $(V_x - V_y)^2 \geq d_{\text{min}}^2$, $y_i$ 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 $y_{ij}$ with the minimum mean difference to $x$, calculates the distance $r_p$ between $x$ and $y_{ij}$ and sets the current minimum distortion $d_{\text{min}}^2$ as $r_p^2$. For each codeword $y_{ij}$, the algorithm checks if $m_{xij}$ is between $m_{\text{min}}$ and $m_{\text{max}}$, where

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

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

Note that, in the ENNS algorithm, for any codeword $y_{ij}$ with $m_{xij}$ between $m_{\text{min}}$ and $m_{\text{max}}$, the distortion $d^2(x, y_{ij})$ is always evaluated. In contrast, the proposed algorithm will calculate $d^2(x, y_{ij})$ only when $(V_x - V_y)^2 \leq d_{\text{min}}^2$. 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, $\varepsilon =$ distortion threshold. Set iteration counter $c = 0$, initial total distortion $D_c = \infty$.

**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 $Y_c$. 


145
\[ Y_I = \{ y_i | m_d \leq m_{d+1} \}, \ 1 \leq i \leq N - 1 \} \]

Step 2: Compute the squared root of the variance \( V_{ij} \) of each codeword \( y_i \).

![Fig. 2 Example to illustrate proposed method for 2-dimensional case](image)

○ input vector \( x \)

● codewords

Step 3: For each training vector \( x_i \), find the closest codeword \( y_{ij} \) in the codebook \( Y_w \) and assign \( x_i \) to class \( \hat{y}(i) \). The procedure includes the following substeps:

Step 3.1: Input a training vector \( x_i = (x_{i1}, x_{i2}, \ldots, x_{iN}) \), compute its mean value \( m_i \) and its square root of variance \( \sqrt{V_{ii}} \).

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

\[ |m_i - m_j| \leq |m_i - m_k| \quad \text{for all } i \neq p \]

Set

\[ d_{\text{max}} = d_i(x_i, y_j), \quad \hat{y}(i) = p, \quad m_{\text{max}} = m_i + \frac{d_{\text{min}}}{\sqrt{k}} \]

and

\[ m_{\text{min}} = m_j - \frac{d_{\text{min}}}{\sqrt{k}} \]

Step 3.3: Find the closest codeword \( y_{ip} \) in \( Y_w \) and assign \( x_i \) to class \( \hat{y}(i) \). The search procedure is as follows

Set \( d = 1 \)

while \( (m_{i+p} < m_{\text{max}} \text{ or } m_{i-p} > m_{\text{min}}) \)

begin

\[ \begin{align*}
& \text{if} (m_{i+p} < m_{\text{max}}) \text{begin} \\
& \text{if} (d_i(x_i, y_{ip}, p) < d_{\text{max}}) \text{begin} \\
& \quad d_{\text{min}} = d_i(x_i, y_{ip}), \\
& \quad m_{\text{max}} = m_i + \frac{d_{\text{min}}}{\sqrt{k}} \end{align*} \]

\[ m_{\text{min}} = m_j - \frac{d_{\text{min}}}{\sqrt{k}} \]

\[ \hat{y}(i) = p + d \]

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.

<table>
<thead>
<tr>
<th>Codebook size</th>
<th>Method</th>
<th>Lena</th>
<th>Peppers</th>
<th>Jet</th>
<th>Baboon</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>LBG</td>
<td>2815</td>
<td>2614</td>
<td>2608</td>
<td>2759</td>
</tr>
<tr>
<td></td>
<td>ENNS</td>
<td>305</td>
<td>284</td>
<td>320</td>
<td>847</td>
</tr>
<tr>
<td></td>
<td>our method</td>
<td>266</td>
<td>242</td>
<td>269</td>
<td>771</td>
</tr>
<tr>
<td>256</td>
<td>LBG</td>
<td>5608</td>
<td>5622</td>
<td>5334</td>
<td>5583</td>
</tr>
<tr>
<td></td>
<td>ENNS</td>
<td>508</td>
<td>472</td>
<td>535</td>
<td>1528</td>
</tr>
<tr>
<td></td>
<td>our method</td>
<td>383</td>
<td>347</td>
<td>392</td>
<td>1313</td>
</tr>
<tr>
<td>512</td>
<td>LBG</td>
<td>11263</td>
<td>11263</td>
<td>10709</td>
<td>11212</td>
</tr>
<tr>
<td></td>
<td>ENNS</td>
<td>865</td>
<td>812</td>
<td>921</td>
<td>2788</td>
</tr>
<tr>
<td></td>
<td>our method</td>
<td>576</td>
<td>525</td>
<td>602</td>
<td>2292</td>
</tr>
<tr>
<td>1024</td>
<td>LBG</td>
<td>22732</td>
<td>22589</td>
<td>21679</td>
<td>22610</td>
</tr>
<tr>
<td></td>
<td>ENNS</td>
<td>1535</td>
<td>1464</td>
<td>1596</td>
<td>5148</td>
</tr>
<tr>
<td></td>
<td>our method</td>
<td>925</td>
<td>866</td>
<td>960</td>
<td>4001</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Codebook size</th>
<th>Method</th>
<th>Lena</th>
<th>Peppers</th>
<th>Jet</th>
<th>Baboon</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>Full search</td>
<td>140.9</td>
<td>140.1</td>
<td>125.1</td>
<td>137.5</td>
</tr>
<tr>
<td></td>
<td>LBG</td>
<td>11.5</td>
<td>11.9</td>
<td>11.4</td>
<td>32.2</td>
</tr>
<tr>
<td></td>
<td>ENNS</td>
<td>14.1</td>
<td>15.0</td>
<td>14.3</td>
<td>37.1</td>
</tr>
<tr>
<td></td>
<td>our method</td>
<td>11.5</td>
<td>11.9</td>
<td>11.4</td>
<td>32.2</td>
</tr>
<tr>
<td>256</td>
<td>LBG</td>
<td>278.6</td>
<td>279.5</td>
<td>249.7</td>
<td>275.0</td>
</tr>
<tr>
<td></td>
<td>ENNS</td>
<td>23.6</td>
<td>27.4</td>
<td>23.9</td>
<td>69.4</td>
</tr>
<tr>
<td></td>
<td>our method</td>
<td>17.2</td>
<td>18.6</td>
<td>17.4</td>
<td>54.1</td>
</tr>
<tr>
<td>512</td>
<td>LBG</td>
<td>568.7</td>
<td>558.1</td>
<td>499.3</td>
<td>547.8</td>
</tr>
<tr>
<td></td>
<td>ENNS</td>
<td>40.8</td>
<td>47.9</td>
<td>43.4</td>
<td>130.9</td>
</tr>
<tr>
<td></td>
<td>our method</td>
<td>26.7</td>
<td>30.2</td>
<td>26.7</td>
<td>96.2</td>
</tr>
<tr>
<td>1024</td>
<td>LBG</td>
<td>1108.5</td>
<td>1108.5</td>
<td>992.7</td>
<td>1087.9</td>
</tr>
<tr>
<td></td>
<td>ENNS</td>
<td>71.4</td>
<td>89.3</td>
<td>79.2</td>
<td>249.2</td>
</tr>
<tr>
<td></td>
<td>our method</td>
<td>40.7</td>
<td>51.1</td>
<td>43.6</td>
<td>174.6</td>
</tr>
</tbody>
</table>
end
if(m_p > m_min)begin
  if((y_v - Y_{ref,p})^2 < d_{min})begin
    d_{min} = d^2(x_t, y_{ref,p})
  end
  m_{max} = m_{st} + d_{min} / \sqrt{(k)}
  m_{min} = m_{st} - d_{min} / \sqrt{(k)}
end
end
end{while}

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

D_c = \sum_{i=1}^{k} d^2(x_i, y_{i0})

Step 5: If(D_{c−1} − D_c)/D_c ≤ ε, halt with final codebook being Y_c. 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 go 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 pre-designed 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 pre-designed codebook. In this simulation, the image Lena shown in Fig. 3a 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-

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

7 Appendix
7.1 Proof of Theorem 1
By definition 1, we have $V_s = d(x, L_s)$ and $V_{ml} = d(y, L_{ml})$. By the triangle inequality, we can obtain
\[ d(x, y) \geq d(x, L_{ml}) - d(y, L_{ml}) \] (6)
Since $L_s$ is the projection point of $x$ on the central line $l$, thus
\[ d(x, L_{ml}) \geq d(x, L_s) \] (7)
By inequality eqn. 7, inequality eqn. 6 can be reduced to be
\[ d(x, y) \geq d(x, L_s) - d(y, L_{ml}) = V_s - V_{ml} \] (8)
Similarly, we can obtain
\[ d(x, y) \geq d(y, L_s) - d(x, L_{ml}) \geq d(y, L_{ml}) - d(x, L_s) = V_{ml} - V_s \] (9)
Combining inequalities eqn. 8 and eqn. 9, we prove Theorem 1.