Fast algorithm for VQ codebook design

Sin-Horng Chen
W.M. Hsieh

Indexing terms: Binary codeword splitting, Image coding, LBG algorithm, Vector quantisation

Abstract: The paper presents a fast codebook training algorithm for vector quantisation. It uses an elimination rule, based on triangular inequality criteria, as well as the partial distortion elimination method, to relieve the computational burden of a conventional codebook training algorithm, including a binary codeword splitting algorithm for the initial codebook and the LBG recursive algorithm. Over 95% savings in both multiplication and addition operations were achieved in the simulation of a VQ codebook training of a 'Lena' image using 16-dimensional vectors.

1 Introduction

Recently, vector quantisation (VQ) is a widely used technique in data compression. A fundamental result of Shannon's rate-distortion theory is that better performance can always be achieved by coding vectors rather than scalars [1, 2]. A complete description of a vector quantisation process includes three phases, namely training, encoding, and decoding. The training phase is a codebook design procedure which tries to find the best set of representatives (i.e. codewords) from a large set of training vectors. The encoding phase finds the best matched codeword for a test vector and uses the index of the codeword to represent it. In principle, a full codebook search could be used in an encoder of vector quantisation to find the codeword which is the nearest neighbour to the test vector. The decoding phase is simply a table look-up procedure which uses the received index to deduce the reproduction codeword. The particularly simple table look-up decoding procedure makes VQ an attractive method of data compression in practice. However, both the training and encoding phases are computation-intensive procedures. This limits the applicability of VQ. To solve this problem, many fast algorithms [3–13] have been proposed in recent years for the encoding search. But few [e.g. 7, 11] have dealt with the training procedure. In this paper, the conventional training algorithm, including the codeword splitting for the initial codebook, and the well-known LBG recursive algorithm will be analysed and a fast algorithm proposed.

A generalised Lloyd's algorithm was developed by Linde, Buzo and Gray [2] for VQ codebook training. It is the most popular codebook training method and is referred to as the LBG algorithm. A distortion measure D must be assigned to assess the cost $D(X, C)$ of reproducing an input vector X as the reproduction codeword C before the LBG algorithm can be applied. Usually the squared error distortion measure is used. Furthermore, an initial codebook with the correct size should be given at the start of the LBG algorithm run. The LBG algorithm recursively uses two steps: first, it partitions the set of training vectors into disjoint sets using the current codebook, and then it finds the centroids of these disjoint sets and takes them as the new codewords. The overall averaged distortion is theoretically proved to be gradually improved by recursively applying these two steps. This means that the codebook improves as more iterations are taken. Actually, the LBG algorithm will always converge to a local optimum codebook which is not necessarily a global one. Iterations are usually terminated when the decreasing rate of the overall averaged distortion is negligible.

In each iteration of the LBG algorithm, the partitioning of the training set can be accomplished by encoding all training vectors using the current codebook. Full codebook search with a nearest-neighbour decision rule are used in these encodings. Because the conventional full-search algorithm is a very time-consuming process and the number of vectors in the training set is in general very large, the computational complexity of partitioning the training set in each iteration of the LBG algorithm is very high. On the other hand, the computational complexity of finding the centroids of the disjoint partitioned sets is relatively low. Based on the above analysis, it can be seen why almost all training procedures of practical VQ-based applications are performed off-line. One disadvantage of performing the codebook design off-line, however, is that a fixed codebook must be used. This limits the applicability of vector quantisation, especially in those cases where on-line adjustment of the codebook is necessary to adapt the changing environment. VQ-based TV picture coding is an example where the codebook should be updated every few frames.

Another component of a codebook training algorithm is the method of initial codebook generation. Two basic approaches have been developed for generating an initial codebook [1]. The first technique uses some simple codebook of the correct size. Randomly selecting the first N vectors in the training sequence as the initial codebook is a typical example of this approach. The second technique starts with a simple small codebook and recursively constructs a larger one. Binary codeword splitting is the most popular example of this approach. In this, the centroid of the entire training sequence is first found, and this single codeword is then slightly perturbed and split into two codewords. The first-level initial codebook is obtained by first partitioning the training set using these two codewords and then finding the centroids of the two

Paper 8119I (E5), received 12th March 1990

The authors are with the Department of Communication Engineering, National Chiao Tung University, Hsinchu, Taiwan 30039, Republic of China

partitioned sets. The algorithm continues until an initial codebook of the correct size is obtained.

The computational complexity of both the LBG algorithm and the binary codeword splitting approach can be greatly reduced if an efficient full codebook search can be applied to the partitioning of the training set. To reduce the computational complexity of a full codebook search, many methods have been studied in recent years [3–13]. Fukunaga and Narendar [3] proposed a branch-and-bound (BAB) algorithm for computating k nearest neighbours. A BAB algorithm [3–6] is a tree search algorithm using a hierarchical decomposition of the sample set of known patterns. It uses the criterion of triangular inequality to develop rules to eliminate the distance computation in the tree classifier. Bei and Gray [7] proposed an elimination algorithm to compress the computation time, using the same idea as the 'partial distance method'. Vidal [8] presented the approximating and eliminating search algorithm (AESA) in which the computation time is approximately constant for a codeword search in a codebook of large size. Chen and Pan [13] employed triangular inequality elimination on VQ-based isolated-word recognition to take advantage of the high correlation relationship between feature vectors of adjacent speech frames. There are many other high-speed search algorithms for vector quantisation [9–12].

In this paper, a fast training algorithm for vector quantisation is proposed. The conventional full-search encoding method used in the partitioning steps of both the binary codeword splitting and LBG algorithms is modified in this algorithm to speed up the training process. It first uses an elimination rule based on triangular inequality criteria to eliminate all unnecessary distortion computations associated with matching an encoding vector with wildly mismatched codewords. It then applies the partial distortion elimination method to the computations for matching these surviving codewords. An advantage of applying the triangular inequality elimination before using the partial distortion method is that, in general, very small minimum distortion exists after applying the triangular inequality elimination and therefore the partial distortion method can be applied with high efficiency.

2 Fast VQ training algorithm

2.1 Triangular inequality elimination

Because the computational complexity of a conventional full codebook search is very high, many methods have been proposed for compressing the distortion computations of codeword matching. Among them, the elimination rule based on triangular inequality criteria is the most popular. One form of the triangular inequality elimination rule is presented below.

Let Dis (X, C_1) be the distance between the encoding vector X and codeword C_1. If Dis $(C_1, C_2) > 2$ Dis(X, C_1), then eliminate the computation of Dis (X, C_2) because it is always greater than Dis (X, C_1). For squared error distortion measure, the condition of elimination is changed to Dis $(C_1, C_2) > 4$ Dis (X, C_1).

An advantage of using the squared error distortion measure is that the square root operation in Dis $(\cdot, \cdot)$ is not needed.

Although many other triangular inequality elimination rules exist [3–6, 8], the one stated above is the simplest and is very efficient if initially a codeword which has small distortion (distance) to the encoding vector can be identified. This is because many distortion computations can then be eliminated. In the partitioning step of the LBG algorithm, an intuitive choice of the codeword to be used as the first matching codeword for an encoding vector is the one associated with the disjoint partitioned set that this encoding vector belonged to in the previous iteration. In the steady state (i.e. the number of iterations in the LBG algorithm approaches infinity), almost every training vector will be encoded to the codeword associated with the disjoint partitioned set to which it belonged in the previous iteration. Therefore, it is the best codeword to be found that has minimum distortion to an encoding vector at this iteration. For the first iteration, the method of initial codebook generation using binary codeword splitting provides the cue to apply the triangular inequality elimination. Because the last step of the codeword splitting method is the same as an iteration of the LBG algorithm, each codeword in the initial codebook is associated with a disjoint partitioned set of the training vectors, and the triangular inequality elimination can therefore be directly applied to the LBG algorithm. Similar considerations indicate that the triangular inequality elimination can be employed for the binary codeword splitting in the fast training algorithm.

Multiple use of the triangular inequality elimination in the encoding of a vector is possible. Codewords that have small distortion to the first selected codeword can be used for the successive triangular inequality eliminations. But simulation results in Section 3 show that little computational advantage is gained as more than one triangular inequality elimination is required. Therefore, a single triangular inequality elimination is used in the proposed fast training algorithm.

In the realisation of the triangular inequality elimination, a table showing the distortion between any pair of codewords is needed, and every row of this table must be sorted in increasing order so that the triangular inequality elimination can be efficiently applied. Constructing the distortion table from a codebook is the overhead that must be paid for applying triangular inequality elimination. For a codebook of size N, the number of operations for establishing the distortion table is analysed as follows:

$$\text{Multiplication} = \frac{N(N-1)M}{2}$$

$$\text{Addition} = \frac{N(N-1)(2M-1)}{2}$$

$$\text{Comparison} = \frac{N(N-1)(N-2)}{2}$$

Here M is the dimension of the codeword, and bubble sort is used for sorting the distortion table.

2.2 Partial distortion elimination

Partial distortion elimination [7] has been proposed to reduce the computation load of the codeword search in a codebook. It first calculates the distortion between the encoding vector and an arbitrary codeword and takes this as the current minimum distortion. It then continuously compares the cumulative partial distortion between the encoding vector and a candidate codeword with the current minimum distortion. Based on the result of this comparison, it decides whether or not to eliminate the remaining partial distortion computation. In the meantime, if a total distortion is obtained, it updates the current minimum distortion by choosing the minimum of

the current minimum distortion and the calculated total distortion. The efficiency of applying partial distortion elimination depends on whether or not the current minimum distortion is small enough. If it is not, then partial distortion elimination is of little use. On the other hand, a very small current minimum distortion obtained at the beginning of the full codebook search for an encoding vector can eliminate most unnecessary computations and shows the full capability of this elimination method. In the LBG algorithm, the codeword (centroid) of the disjoint partitioned set to which the current encoding vector belongs is a good candidate for initial matching. Its distortion can be used as the current minimum distortion for the partial distortion elimination. Actually, in the following fast training algorithm, this distortion is first used in the triangular inequality elimination to compress unnecessary codeword matchings and then used in the partial distortion elimination to reduce the number of computations in matching all surviving codewords.

2.3 Fast VQ training algorithm

A fast VQ training algorithm can be obtained by incorporating the triangular inequality elimination and the partial distortion elimination in the conventional codebook training algorithm which includes the binary codeword splitting and LBG algorithms. The procedure is as follows.

Binary codeword splitting

Step 1: Initialisation. Given a set of training vectors $S = \{X_i; i = 1, 2, \ldots, L\}$, the size of codebook N, and an initial partition $P^1 = \{S\}$, let $j = 1$.

Step 2: Centroid calculation. Find the centroids C_k of all $P_k^j \in P^j$ from

$$C_k = \frac{1}{L_k} \sum_{l=1}^{L_k} X_l \quad \text{for } k = 1, \ldots, j$$

where L_k is the number of vectors in P_k^j and $X_l \in P_k^j$. If j equals N, go to Step 6; otherwise, continue.

Step 3: Codeword splitting. Split each C_k into two by letting

$$B_{2k-1} = C_k + \delta$$

and

$$B_{2k} = C_k - \delta$$

for $k = 1, \ldots, j$. Here δ is a small perturbation vector. Let $j = 2j$.

Step 4: Distortion table construction. Construct a distortion table for the codebook $B^j = \{B_k; k = 1, \ldots, j\}$ by calculating distortions for all codeword pairs and sorting each row in increasing order using the bubble sort.

Step 5: Training set partitioning. Partition the training set into disjoint sets by encoding all training vectors using the codebook B^j. The encoding of a training vector X_i includes the following sub-steps:

Step 5.1: Calculate the distortions, D_1 and D_2, of matching X_i with B_{2k-1} and B_{2k} split from C_k associated with the partitioned set $P_k^{j/2}$ in $P^{j/2}$ to which X_i previously belonged. Choose the minimum of D_1 and D_2, denote it as D_{min}, and take the index of the codeword associated with it as l_{min}.

Step 5.2: Compare all elements in the l_{min}-th row of the distortion table with $4D_{min}$. Eliminate all matching computations of codewords which have larger distortions than the codeword $B_{l_{min}}$ in the above comparisons.

Step 5.3: Calculate the distortions of matching X_i with all surviving codewords by using a full-search method with partial distortion elimination. Update l_{min} and D_{min} when a smaller total distortion is obtained.

Step 5.4: The codeword $B_{l_{min}}$ associated with D_{min} is the encoded result. Put X_i in the set $P_{l_{min}}^j$.

After encoding all training vectors, a partition of S, $P^j = \{P_1^j, P_2^j, \ldots, P_{j^j}^j\}$, is obtained. Go to Step 2.

LBG algorithm

Step 6: Initialisation of the LBG algorithm. Let $C^0 = \{C_k^0; C_k^0 = C_k, \text{ for } k = 1, \ldots, N\}$ be the initial codebook, $P^0 = \{P_k^0; P_k^0 = P_k^N, \text{ for } k = 1, \ldots, N\}$ be the initial partition of S, $j = 0$ and $D_{av}^0 = \infty$.

Step 7: Distortion table construction. Construct the distortion table D^j for codebook C^j.

Step 8: Training set partitioning. Partition the training set into disjoint sets by encoding all training vectors X_i using the codebook C^j. The encoding of X_i includes the following sub-steps:

Step 8.1: Calculate the distortion of matching X_i with codeword C_k^j associated with the set P_k^j that X_i belonged to and denote it as D_{min}. Let l_{min} be the index of C_k^j.

Step 8.2: Apply the triangular inequality elimination rule to eliminate all unnecessary codeword matchings. Here the distortion table D^j and the current minimum distortion D_{min} are used.

Step 8.3: Calculate the distortions between X_i and all surviving codewords by using the full-search method with partial distortion elimination. Update l_{min} and D_{min} when a smaller total distortion is obtained.

Step 8.4: The codeword $C_{l_{min}}^j$ associated with D_{min} is the encoded result. Put X_i in $P_{l_{min}}^{j+1}$ and accumulate the overall encoding distortion.

After obtaining the partition P^{j+1}, calculate the overall averaged distortion D_{av}^{j+1}.

Step 9: Centroid calculation. Let $j = j + 1$ and find centroids of all disjoint partitioned sets in P^j from

$$C_k^j = \frac{1}{L_k} \sum_{l=1}^{L_k} X_l$$

where L_k is the number of vectors in P_k^j and $X_l \in P_k^j$.

Step 10: Termination checking. Calculate the decreasing rate of the overall averaged distortion from

$$\Delta D = D_{av}^{j-1} - D_{av}^j$$

If ΔD is less than ε, then go to Step 11; otherwise go to Step 7. Here ε is a predetermined small threshold.

Step 11: Termination. Take the codebook C^j as the final codebook and terminate the algorithm.

3 Simulation

The efficiency of the proposed fast training algorithm was examined by simulation on VQ-based image coding. A 512 × 512 image ('Lena' image) is used in all the following simulations. The image is first divided into 4 × 4 subimages such that the training group contains 16 384 16-dimensional vectors. Several codebooks of different sizes are trained from the same training set.

First, the effectiveness of applying triangular inequality elimination (TIE) and partial distortion elimination (PDE) in the binary codeword splitting algorithm was tested. Figs. 1 and 2 show the elimination efficiencies of TIE and PDE, respectively. The average percentage of

eliminated codeword matchings using TIE is shown in Fig. 1. The efficiency of applying TIE steadily increases as the size of the initial codebook increases. The average number of calculated components for a vector in the matching computation using PDE is shown in Fig. 2. Only about one half of the vector components are used in

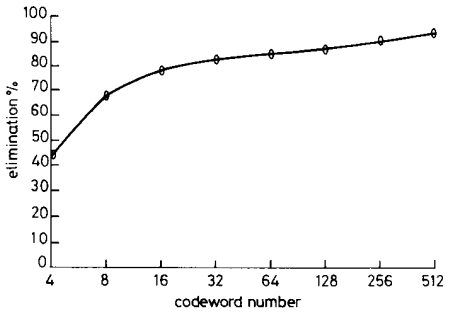
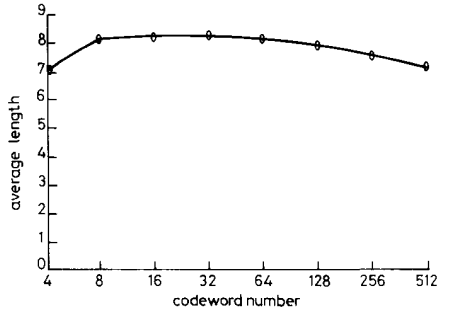
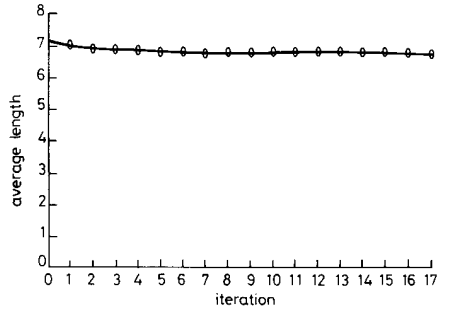


Fig. 1 *Average percentage of eliminated codeword matchings using TIE in the binary codeword splitting algorithm*



a



b

Fig. 2 *Average number of calculated components for a vector in the matching computations with codewords surviving after TIE*

a Binary codeword splitting
b LBG algorithm (codeword number 256)

the matching computations of a vector whose codewords survived after applying TIE. The computational efficiency of applying both TIE and PDE in binary codeword splitting is shown in Table 1. The savings in both multiplications and additions increase from about 48% when the initial codebook size is 4 to about 95% when it is 512.

A codebook of size 256 was generated to test the effectiveness of applying TIE in the LBG algorithm. In each iteration, TIE is applied one to five times for the encoding of each training vector. Then, full-search methods with and without PDE are employed to match the encoding vector with all surviving codewords. Table 2 shows the average number of eliminated codeword

Table 1 :Computational efficiency of applying TIE and PDE in the binary codeword splitting algorithm

Initial codebook	Fast algorithm			Conventional algorithm		
size	mult.	add.	comp.	mult.	add.	comp.
4	33.9 (53.0)	63.6 (51.3)	4.1 (134.7)	64	124	3
8	36.6 (28.6)	66.1 (26.7)	7.2 (102.6)	128	248	7
16	44.6 (17.4)	73.2 (14.8)	16.1 (107.2)	256	496	15
32	63.9 (12.5)	90.5 (9.1)	38.1 (123.0)	512	992	31
64	101.8 (9.9)	125.2 (6.3)	85.8 (136.2)	1024	1984	63
128	160.4 (7.8)	182.5 (4.6)	200.4 (157.8)	2048	3968	127
256	251.1 (6.1)	285.7 (3.6)	720.6 (282.6)	4096	7936	255
512	437.4 (5.3)	547.6 (3.5)	4391.3 (859.4)	8192	15872	511

Values in parentheses denote the percentage ratio of operations to the conventional algorithm.

Table 2: Average number of eliminated codeword matchings for the encoding of a training vector by the first five TIE operations in an iteration of the LBG algorithm

Number of TIE		1	2	3	4	5
Average eliminated codeword matchings	number	242.0	0.44	0.38	0.23	0.17
	%	94.9	0.17	0.15	0.09	0.07

matchings for the encoding of a training vector by the first five TIE operations in an iteration. The results obtained are very impressive in that, an average, about 242 out of 255 codeword matchings are eliminated when the first TIE operation is applied. Successive TIE operations are of little use. This result confirms that, in applying TIE in the LBG algorithm, the codeword associated with the partitioned set that the encoding vector belongs to is a good choice for initial calculation. Fig. 3 shows the relationship between the efficiency of codeword elimination and the overall average distortion. It shows that, as the number of iterations increases, the efficiency of codeword elimination increases, while the overall average distortion decreases. Table 3 lists the average number of operations for a training vector in one iteration. Note that the overhead of constructing the distortion table is included in Table 3. When PDE is not applied, it trades multiplications and additions against comparisons as more TIE operations are used. But, when PDE is used, more than one TIE operation is shown to be inefficient. Using only one TIE operation, both multiplications and additions are drastically reduced at the cost of a small increase in comparisons. From the above discussion, it is seen that the strategy of choosing the first matching codeword in the fast training algorithm makes both TIE and PDE very efficeint.

Finally, the computational complexity of the fast training algorithm was examined by generating codebooks of different sizes from the same image data. Table 4 lists the simulation results for codebooks of levels from 6 to 9. Note that the computations of centroids are not

included in Table 4. Compared with the conventional full-search training algorithm, about 85% and 95% savings in both multiplications and additions are achieved for the level-6 codebook training in binary codeword splitting and the LBG algorithm, respectively. This increases to about 94% and 97% savings respectively for a level-9 codebook. The increase in comparisons is not serious, considering the great savings in both multiplications and additions. For a level-8 codebook training, the average computations needed for a training vector in the fast training algorithm is shown in Table 5. Over 96% savings in both multiplications and additions are obtained, but only about 2.36 times the number of comparisons are needed compared with the conventional training algorithm. These results show that the fast training algorithm is a very efficient one.

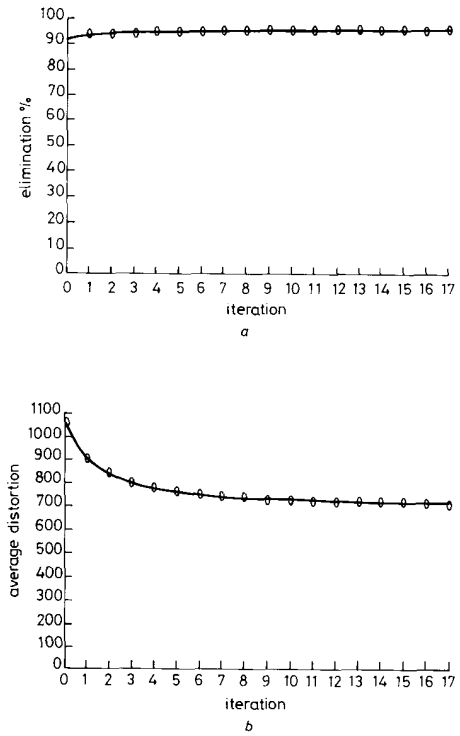


Fig. 3 *Relationship between elimination efficiency and overall average distribution (codeword number: 256)*

a Elimination efficiency
b Overall average distortion

Table 3: Average number of operations for a training vector in an iteration of the LBG algorithm

	Number of TIE	Average number of operations		
		mult.	add.	comp.
TIE only	1	256.0	496.0	533.0
	2	248.9	482.3	544.4
	3	242.8	470.5	554.6
	4	239.2	463.4	563.8
	5	236.5	458.2	572.2
TIE and PDE	1	136.5	168.4	608.7
	2	140.7	188.3	610.4
	3	145.2	207.6	611.9
	4	150.3	225.5	614.6
	5	154.9	240.8	617.9
Full-search		4096	7936	255

Table 4: Average number of operations in *a* the binary codeword splitting and *b* the LBG algorithm for a training vector using the fast training algorithm to generate codebooks of levels from 6 to 9.

Codebook size	Fast algorithm			Conventional algorithm		
	mult.	add.	comp.	mult.	add.	comp.
64	312.8 (15.5)	480.6 (12.3)	152.3 (126.9)	2016	3906	120
128	473.2 (11.6)	663.1 (8.4)	352.7 (142.8)	4064	7874	247
256	724.3 (8.9)	948.8 (6.0)	1073.3 (213.8)	8160	15810	502
512	1161.8 (7.1)	1496.3 (4.7)	5464.6 (539.5)	16352	31682	1013

a

Codebook size	Fast algorithm			Conventional algorithm		
	mult.	add.	comp.	mult.	add.	comp.
64	55.7 (5.4)	66.5 (3.4)	50.5 (80.2)	1024	1984	63
128	81.6 (4.0)	95.8 (2.4)	129.4 (101.9)	2048	3967	127
256	136.5 (3.3)	168.4 (2.1)	608.7 (238.7)	4096	7936	255
512	274.0 (3.3)	389.0 (2.5)	4223.0 (826.4)	8192	15872	511

b

Values in parentheses denote the percentage ratio of operations to the conventional algorithm

Table 5: Average number of operations for a training vector using the fast training algorithm to generate a codebook of level 8

	Fast algorithm			Conventional algorithm		
	mult.	add.	comp.	mult.	add.	comp.
Binary codeword splitting	724.3 (8.9)	957.7 (6.1)	1073.3 (213.8)	8160	15818.97	502
LBG algorithm (18 iteration)	2457.1 (3.3)	3048.6 (2.1)	10955.8 (238.7)	73728	142865.7	4590
Total	3181.5 (3.9)	4006.4 (2.5)	12029.1 (236.2)	81888	158684.7	5092

Values in parentheses denote the percentage ratio of operations to the conventional algorithm

4 Conclusions

A fast training algorithm has been presented in this paper. It efficiently incorporates the triangular inequality elimination rule and the partial distortion elimination method into the conventional VQ training algorithm. Over 96% computation savings in both multiplication and addition operations are achieved with little increase in the number of comparison operations.

The same idea can be extended to other VQ-based applications if a method of finding a suitable codeword for initial matching exists. In some applications such as image coding, speech coding, and speech recognition, the high correlation relationship existing between adjacent vectors provides a clue to finding such a suitable candidate for the first codeword to be matched. Applying TIE and PDE to these areas is being studied.

5 References

1 GRAY, R.M.: 'Vector quantisation', *IEEE ASSP Mag.*, April 1984, pp. 4–28

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

3 FUKUNAGA, K., and NARENDRA, P.M.: 'A branch and bound algorithm for computing k-nearest neighbors', *IEEE Trans.*, 1975, **C-24,** pp. 750–753

4 KAMGAR-PARSI, B., and KANAL, L.N.: 'An improved branch and bound algorithm for computing k-nearest neighbours', *Pattern Recognit. Lett.*, 1985, **3,** pp. 7–12

5 LARSEN, S., and KANAL, L.N.: 'Analysis of k-nearest neighbor branch and bound rules, *Pattern Recognit. Lett.*, 1986, **4,** pp. 71–77

6 NIEMANN, H., and GOPPERT, R.: 'An efficient branch-and-bound nearest neighbour classifier', *Pattern Recognit. Lett.*, 1988, **7,** pp. 67–72

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

8 VIDAL, E.: 'An algorithm for finding nearest neighbours in (approximately) constant average time', *Pattern Recognit. Lett.*, 1986, **4,** pp. 145–157

9 CHENG, D.-Y., GERSHO, A., RAMAMURTHI, B., and SHOHAM, Y.: 'Fast search algorithms for vector quantisation and pattern matching'. IEEE Int. Conf. Acoust. Speech Signal Process., 1984, pp. 9.11.1–4

10 CHENG, D.-Y., and GERSHO, A.: 'A fast codebook search algorithm for nearest–neighbor pattern matching'. IEEE Int. Conf. Acoust. Speech Signal Process., 1986, pp. 6.14.1–4

11 SOLEYMANI, M.R., and MORGERAI, S. D.: 'A high-speed search algorithm for vector quantisation'. IEEE Int. Conf. Acoust. Speech Signal Process., 1987, pp. 45.6.1–3

12 FISCHER, F.P., and PATRICK, E.A.: 'A preprocessing algorithm for nearest neighbor decision rules'. Proceedings of the National Electroncs Conference, 1970, **26,** pp. 481–485

13 CHEN, S.H., and PAN, J.S.: 'A fast algorithm for VQ-based recognition of isolated words', *IEE Proc. I, Commun., Speech & Vision*, 1989, **136,** (6), pp. 391–396