IMPROVED ALGORITHMS FOR VQ CODEWORD SEARCH, CODEBOOK DESIGN AND CODEBOOK INDEX ASSIGNMENT

Jenq-Shyang Pan

A thesis submitted for the degree of
Doctor of Philosophy
to the Faculty of Science and Engineering of the University of Edinburgh
1996

Abstract

This thesis investigates efficient codeword search algorithms and efficient clustering algorithms for vector quantization (VQ), improved codebook design algorithms and improved codebook index assignment for noisy channels.

In the investigation of codeword search algorithms, several fast approaches are proposed, such as the improved absolute error inequality criterion, improved algorithms for partial distortion search, improved algorithms for extended partial distortion search and a fast approximate search algorithm. The bound for the Minkowski metric is derived as the generalised form of the partial distortion search algorithm, hypercube approach, absolute error inequality criterion and improved absolute error inequality criterion. This bound provides a better criterion than the absolute error inequality elimination rule on the Euclidean distortion measure. For the Minkowski metric of order n, this bound contributes the elimination criterion from the L_1 metric to the L_n metric. This bound is also extended to the bound for the quadratic metric by using methods of metric transformation. The improved absolute error inequality criterion is also extended to the generalised form of the mean-distance-ordered search algorithm for VQ image coding.

Several fast clustering algorithms for vector quantization based on the LBG algorithm are presented. Genetic algorithms are applied to codebook design to derive improved codevectors. The approach of stochastic relaxation is also applied to the mutation step of the genetic algorithm to further improve the codebook design algorithm.

Vector quantization is very efficient for data compression of speech and images where the binary indices of the optimally chosen codevectors are used. The effect of channel errors is to cause errors in the received indices. A parallel genetic algorithm is applied to assign the codevector indices for noisy channels so as to minimize the distortion due to bit errors. The novel property of multiple global optima and the average distortion of the memoryless binary symmetric channel for any bit error in the assignment of codebook index are also introduced.

Declaration of Originality

This thesis and the work reported herein were composed and originated entirely by the author.

Jenq-Shyang Pan

Acknowledgements

Firstly, I would like to thank my first supervisor, Professor Mervyn Jack for his supervision and his support. I would also like to thank my second supervisor Dr. Fergus McInnes for the help in solving many problems, debugging several programs and developing the bound for Minkowski metric.

Thanks to Fabrizio Carraro, Dr. Go-An Rau and Dr. Mark Schmidt for introducing me many facilities at University of Edinburgh and providing the speech and image data base. Thanks also to Dr. Hsiao-Lan Fang, Dave Corne and Wei-Po Lee for introducing and discussing genetic algorithms.

I would like to extend my thanks to many other colleagues and friends who have provided help and advice during my PhD study. Without particular order, thanks to: Professor Sin-Horng Chen, Professor Mingchuan Liang, Kuan-Chun Huang, Chang-Hui Shen, Yu-Tsui Lin, Jessica Chen-Burger, Dr. Martine Grice, Professor Hsien-Tang Tsai, Professor J. D. Dai, Dr. Alan Wrench, David Lin, Judy Chang and Kuo-Cheng Kuo.

Finally, I wish to thank my parents, my mother in law and my wife Shu-Chuan Chu for their encouragement and support.

Contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Importance of vector quantization

Communication by the socially rich medium of speech is one of the most important capabilities possessed by human beings. The speech waveform conveys linguistic information, speaker's tone, speaker's emotion and speaker's state of health. Since the invention of the telephone by Alexander Graham Bell, human beings have been able to exchange information via the telephone without being face to face, communicating in real-time with one another in any place by using mobile phones or any suitable communication tool. One of the major recent advances in such remote speech communication is the development of speech coding techniques. In the area of speech coding, vector quantization (VQ) (Gray, 1984; Gersho & Cuperman, 1983; Buzo *et al.*, 1980) has been shown to be a popular and essential speech coding technique. Furthermore, vector quantization also plays an important role in image coding (Gersho & Gray, 1992; Kubrick & Ellis, 1990; Ramamurthi & Gersho, 1986; Nasrabadi & King, 1988).

Human-machine (computer) communication by speech provides a convenient way to communicate with machines. It reduces the amount of typing a human needs to undertake leaving hands free and allowing access away from the terminal or screen. In addition the ears can be used as well as the eyes. The machine needs to both recognize speech and respond with the results using speech by employing speech recognition techniques and speech synthesis techniques. Although the performance of current speech recognition systems remains imperfect, implementations of efficient and accurate speech recognizers are in widespread use in many applications (Wilpon & Roe, 1994; Nitta, 1994). The automatic speech entry of data or commands in manufacture is popular and related applications include speech-based product inspection, inventory control

and material handling. Speech recognition is also applied to automatic transcription and aids for the hearing impaired or physically disabled. The importance of vector quantization in speech recognition is reported in many papers (Rabiner & Juang, 1993; Deller *et al.*, 1993). The hidden Markov model (HMM) (Huang *et al.*, 1990; Rabiner & Juang, 1986; Huang, 1992) has been shown to be a promising method in speech recognition which relies on the preprocessing stage of vector quantization for discrete or semi-continuous HMM-based recognition. In speech synthesis, vector quantization is also useful for pattern matching to reduce data storage.

Automatic speaker recognition (Forsyth, 1995) involves identifying people from their voices completely automatically. Speaker recognition can be separated into two categories: speaker verification and speaker identification. Both categories use similar techniques to speech recognition, such as dynamic time warping (DTW) (McInnes & Jack, 1988; Rabiner *et al.*, 1978) vector quantization, hidden Markov models and neural networks (NN) (Lippmann, 1987; Wu & Chan, 1993; Farrell *et al.*, 1994). Vector quantization can be seen as the preprocess of DTW, HMM and NN. Vector quantization is also a key element in speaker recognition. Vector quantization is therefore a most fundamental and important technique in speech coding, image coding, speech recognition, speech synthesis and speaker recognition.

Vector quantization has been widely used in various applications as described above. An ordered set of signal samples or parameters can be efficiently coded by matching the input vector to a similar pattern or codevector (codeword) in a predefined codebook. For any given input data vector, the encoder assigns one index to the data vector in which the index is the address of the best matching codevector. In the data compression of speech coding or image coding, the index is transmitted and the decoder replicates the corresponding codevector by a table lookup from a copy of the same codebook. The response time of encoding is a very important factor to be considered for real-time transmission (Cheng *et al.*, 1984; Cheng & Gersho, 1986; Ramasubramanian & Paliwal, 1990; Vidal, 1986; Soleymani & Morgera, 1987b; Ra & Kim, 1993). In this thesis, improvements in the partial distortion search (PDS) algorithm and the extended partial distortion search (EPDS) algorithm are presented. These improve the performance of the partial distortion search method (Bei & Gray, 1985). The bounds for the Minkowski metric and the quadratic metric are derived and applied to codeword search problems to improve the efficiency for the Minkowski distortion measure and the Mahalanobis distortion measure. An improved

fast mean-distance-ordered partial codebook search algorithm for image vector quantization is also reported together with several efficient approaches for training VQ codebooks.

In the application of vector quantization to waveform coding or recognition, the performance depends on the existence of a good codebook of representative vectors. A novel VQ codebook design algorithm using genetic algorithms (GA) (Goldberg, 1989; Davis, 1991) is proposed. This approach provides superior performance compared with the generalized Lloyd algorithm (GLA) (Linde *et al.*, 1980).

A very important problem in quantization theory is how to effectively overcome the performance degradation caused by noisy channels. One possible approach is to use redundant parity bits for error control coding. In this thesis, a parallel genetic algorithm (PGA) (Cohoon *et al.*, 1987; Pettey *et al.*, 1987; Shonkwiler, 1993) is applied to assign the codevector indices for noisy channels so as to minimize the distortion due to bit errors without adding any redundant bit.

1.2 Thesis structure

There are five main chapters in this thesis. Chapter 2 introduces the mathematics and theory essential for the reader. It includes a review of probability and stochastic processes, distortion measures, the Lagrange multiplier technique, theory of vector quantization, hidden Markov models, genetic algorithms and parallel processing. In Chapter 2 a new approach to deriving bounds for the Minkowski metric based on the Lagrange multiplier technique is highlighted. The bound for the Minkowski metric is shown to be a general form of the hypercube approach, the partial distortion search (PDS) algorithm, the absolute error inequality criterion (AEI) and the improved absolute error inequality criterion (IAEI). The improved absolute error inequality criterion is a new criterion presented in this thesis, being derived from this bound. It is shown to provide a better criterion than the absolute error inequality criterion on the Euclidean distortion measure. For the Minkowski metric of order n, this bound contributes the elimination criterion from the L_1 metric to the L_n metric. The bound for the Minkowski metric is also extended to the bounds for the quadratic metric by using the methods of the Triangular Matrix and the Karhunen-Loêve transform (KLT). The bounds for the quadratic metric can be applied to any codeword search in which the distortion measure is quadratic. One of the main contributions

in this thesis is the derivation of the bound for the Minkowski metric and the bounds for the quadratic metric.

Chapter 3 reviews the history of many fast codeword search algorithms and introduces key elements of codeword search algorithms. A range of new and efficient codeword search algorithms are presented in this chapter. The processors can be separated into two classes, i.e., general processors and DSP processors. For general processors, such as Intel 80486, Intel Pentium and Motorola 680x0, the operation of multiplication is more expensive than the operation of addition and comparison. For DSP processors, such as the TMS320 series of processors, the operation of comparison is computationally expensive. The novel partial distortion search algorithm is shown to be very suitable for use with general processors and is less suited to DSP processors. By considering the cost ratio of the comparison computation time to dimension-distortion computation time, an improved PDS algorithm and a new and improved DPPDS algorithm are also proposed here to enhance the performance of the partial distortion search algorithm which is in fact suitable for any processor. The extended partial distortion search (EPDS) algorithm is a modified version of the partial distortion search (PDS) algorithm and is an optimal PDS in the sense of reducing the number of multiplications. The EPDS algorithm is however, only suitable for general processors. An improved EPDS algorithm based on available computer architecture is derived in this chapter. By considering the cost ratio of the sorting time to dimension-distortion computation time of a given processor, the optimal inserting point of the sorting can be predicted from the derived equations. This improves the performance of the EPDS algorithm.

The improved absolute error inequality criterion (IAEI) is a special case of the bound for the Minkowski metric. It is the most efficient criterion for reducing the number of multiplications for the full search algorithm based on a Euclidean distortion measure. An efficient algorithm is proposed in Chapter 3, combining the IAEI criterion with the minimax method. Comparison of this new and efficient algorithm with the minimax method, demonstrates a reduction in the number of multiplications by more than 77% and with a slight reduction in the total number of mathematical operations for 1024 codewords. Since the absolute error inequality has already been shown to be the most efficient criterion in reducing the number of multiplications (Huang *et al.*, 1992; Soleymani & Morgera, 1987b; Soleymani & Morgera, 1989), experiments are also reported in Chapter 3 to demonstrate that the IAEI criterion can reduce the number of

multiplications by more than 21% and better than 3% for the total number of mathematical operations compared with the AEI criterion. Also, Chapter 3 shows that (in theory) the IAEI provides a tighter bound than the AEI criterion. A new fast algorithm for approximate search is also presented in this chapter and the IAEI criterion is also extended to the generalised form of the mean-distance-ordered partial codebook search (MPS) algorithm (Ra & Kim, 1993) for image coding. The improved mean-distance-ordered partial codebook search (IMPS) algorithm is developed by employing this generalised formula. The drawback of the MPS algorithm is addressed and the IMPS algorithm is shown to overcome this drawback. In codeword search experiments, without applying the PDS algorithm both in the IMPS algorithm and the MPS algorithm, the IMPS algorithm is shown to reduce the computation time by more than 43% compared with the MPS algorithm for 1024 codewords. The IMPS algorithm is also shown to reduce the number of multiplications by more than 27% and reduce the total number of mathematical operations about 15% for 1024 codewords for applying the partial distortion search algorithm both in the IMPS algorithm and the MPS algorithm.

Several fast clustering algorithms for vector quantization are introduced in Chapter 4 and two tentative match approaches (previous vector candidate and previous partitioned centre) are used in the experiments. The triangular inequality elimination (TIE) and the codebook reorder method are introduced in this chapter. Many combinations of the improved absolute error inequality criterion, absolute error inequality criterion, hypercube approach, partial distortion search, triangular inequality elimination criteria and codebook reorder method to produce fast clustering algorithms are presented here. Among these approaches, the most efficient algorithm for general processors is the IPC-type clustering algorithm which is a combination of the previous partitioned centre, the IAEI criterion and the PDS algorithm. For DSP processors, the TPC-type clustering algorithm which is the combination of a previous partitioned centre, triangular inequality elimination (TIE) and PDS algorithm, outperforms the other algorithms.

Chapter 5 reviews several codebook design algorithms. The K-means algorithm, ISODATA clustering algorithm, GLA, pairwise nearest neighbour algorithm, fuzzy C-means clustering algorithm, deviation reduction algorithm, codebook design by stochastic relaxation, codebook generation using simulated annealing method and vector quantizer design using path-following are discussed in this chapter. Finally, a novel codebook design approach based on genetic al-

gorithms is proposed. This algorithm is the combination of genetic algorithms and GLA which is called GA-GLA algorithm. An improved version of GA-GLA is also presented by inserting the stochastic relaxation method in the mutation step. Experimental results demonstrate that the GA-GLA algorithms are significantly better than the GLA algorithm.

Chapter 6 introduces the importance of codevector index assignment for noisy channels. The property of multiple global optima and the average distortion of the memoryless binary symmetric channel for any bit error are demonstrated. The ensemble average distortion for any bit error in the memoryless binary symmetric channel is derived for the first time in this thesis and the property of multiple global optima is also reported here for the first time. The property of multiple global optima can be used to reduce the search space for codebook index assignment in noisy channels. A new (good) codevector index assignment based on parallel genetic algorithm is presented. It is further shown that applying the parallel genetic algorithm in the codebook index assignment, not only speeds up the computation time but also generates better results. The proposed use of genetic algorithms for codebook index assignment for noisy channels is suggested in this thesis for the first time.

The final chapter summarizes the important discoveries and conclusions of this thesis. Several possible methods for future work are also addressed in this chapter.

Chapter 2

Mathematics and Theory

2.1 Review of Probability and Stochastics Processes

2.1.1 Theory of Probability

Probability is a set function P that assigns to each event E in the sample space Ω a number $P(E)$, called a probability of event E, such that the following properties are satisfied:

1. Probabilities are non-negative, $P(E) \geq 0$.
2. The probability of the entire space Ω is 1, $P(\Omega) = 1$.
3. The probability of the union of the mutually exclusive events E_i, $i = 1, 2, ..., M$ is the sum of the probabilities of the individual events, i.e., $P(E_1 \cup E_2 \cup ...E_M) = P(E_1) + P(E_2) + ... + P(E_M)$, where mutually exclusive means $E_i \cap E_j = \phi$ for any $i \neq j$.

A probability measure P can also be defined in terms of a real valued function f defined on R with the following properties:

1. $f(x) \geq 0$, $x \in R$.

2. $\int_{-\infty}^{\infty} f(x)dx = 1$.

3. $P(F) = \int_F f(x)dx$, F is an event.

The function f is called a probability density function (pdf). Some of the more commonly used pdf's on R are listed below.

1. Gaussian pdf:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{2.1}$$

where μ is the *mean* of x and σ is the *standard deviation*.

2. Uniform pdf:

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

where $b > a$.

3. Exponential pdf:

$$f(x) = \frac{1}{\lambda} e^{-\frac{x}{\lambda}}, \tag{2.3}$$

where $\lambda > 0$.

4. Laplacian (doubly exponential) pdf:

$$f(x) = \frac{1}{\sqrt{2\sigma^2}} e^{-\frac{\sqrt{2}|x|}{\sigma}}, \tag{2.4}$$

where σ is the *standard deviation* of x.

If the sample space Ω is a *discrete* set of real numbers, then a function p can be defined for all points in Ω which has the following properties:

1. $p(x) \geq 0, \quad x \epsilon \Omega.$

2. $\sum_{x \epsilon \Omega} p(x) = 1.$

3. $P(F) = \sum_{x \epsilon F \cap \Omega} p(x).$

The function p is called a *probability mass function* (*pmf*). Some of the more commonly used pmf's are listed below.

1. Binary pmf:

$$p(1) = q, p(0) = 1 - q, \Omega = \{0, 1\}. \tag{2.5}$$

2. Uniform pmf:

$$p(x) = \frac{1}{n}, x \epsilon \Omega = \{0, 1, ..., n - 1\}. \tag{2.6}$$

3. Geometric pmf:

$$p(x) = (1 - q)q^x, x \epsilon \Omega = \{0, 1, ...\}. \tag{2.7}$$

where q is a real number in [0,1].

4. Poisson pmf:

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!}, x \epsilon \Omega = \{0, 1, ...\}. \tag{2.8}$$

where λ is a positive real number.

Given a probability function P or the probability density function f, the *cumulative distribution function* (cdf) $F(r)$ is defined by

$$F(r) = P(x | x \leq r), \quad \text{for discrete sample space} \tag{2.9}$$

or

$$F(r) = \int_{-\infty}^{r} f(x)dx, \quad \text{for continuous sample space.} \tag{2.10}$$

This implies that

$$f(r) = \frac{dF(r)}{dr}. \tag{2.11}$$

2.1.2 Random Variable and Random Process

A (real) random variable **X** is a mapping from the sample space into the real number line: **X** : $\Omega \rightarrow R$, i.e., **X** assigns a real number to every point in the sample space. If a random variable **X** is discrete and its allowable values are $x_1, x_2, ..., x_n$, then the probability of the discrete random variable taking the value x_i is denoted as $p(X = x_i)$. The sum of the probability over all values of the random variable is

$$\sum_{i=1}^{n} p(X = x_i) = 1. \tag{2.12}$$

If **X** is a continuous random variable, then the probability of the continuous random variable taking the value x is denoted as $f_X(x)$. The integral of the probability over all values of the random variable is

$$\int_{-\infty}^{\infty} f_X(x)dx = 1. \tag{2.13}$$

A random vector is a vector whose components include multiple random variables, i.e., a random vector is a finite collection of random variables. A random vector is said to be independent and identically distributed (iid) if it has independent components with identical marginals, i.e., the corresponding probability functions are identical. A random process is an indexed family of random variables $\{X_t; t \epsilon T\}$. The index t corresponds to time. If T is continuous, then the process is called a continuous time random process. If T is discrete, then the process is called a discrete time random process or a random sequence. A discrete time random process is said to be independent and identically distributed (iid) if the random variables produced by the process are independent and have identical distributions.

2.2 Metrics and Distortion Measures

A key component of pattern matching is the measurement of dissimilarity between two feature vectors. Assume X, Y and Z are three vectors in a multidimensional space. Without loss of

generality, k-dimensional *real Cartesian space* denoted R^k is used as the collection of all k-dimensional vectors with real elements. On R^k, a metric D is a real-valued function which fulfils the following three metric properties:

1. Positive definiteness property

$$0 \leq D(X, Y) < \infty \quad \text{for} \quad X, Y \epsilon R^k, \tag{2.14}$$

$$\text{and} \quad D(X, Y) = 0 \quad \text{iff} \quad X = Y. \tag{2.15}$$

2. Symmetry property

$$D(X, Y) = D(Y, X) \quad \text{for} \quad X, Y \epsilon R^k. \tag{2.16}$$

3. Triangular inequality property

$$D(X, Y) \leq D(X, Z) + D(Y, Z) \quad \text{for} \quad X, Y, Z \epsilon R^k. \tag{2.17}$$

If the measurement of dissimilarity satisfies only the positive definiteness property, it is called the *distortion measure*, such as the Itakura distortion measure and the likelihood distortion measure (or the Itakura-Saito distortion family) (Rabiner & Juang, 1993). Each metric has its own advantages and drawbacks. Three main characteristics (Devijver & Kittler, 1982) of the metric are computational complexity, analytical tractability and feature evaluation reliability. The choice of a particular metric depends on the actual application.

2.2.1 Minkowski Metric

Most of the metrics used in speech and image processing are special cases of the Minkowski metric. Let x^i denote the *ith* component of the k-dimensional vector X. The Minkowski metric of order p (Deller *et al.*, 1993), or the L_p metric, between vectors X and Y can be expressed as

$$D_p(X, Y) = \sqrt[p]{\sum_{i=1}^{k} |x^i - y^i|^p}, \tag{2.18}$$

where $X = \{x^1, x^2, ..., x^k\}$ and $Y = \{y^1, y^2, ..., y^k\}$.

Three special cases are as follows:

1. L_1 or city block metric

$$D_1(X, Y) = \sum_{i=1}^{k} |x^i - y^i|. \tag{2.19}$$

2. L_2 or Euclidean metric

$$D_2(X, Y) = \sqrt{\sum_{i=1}^{k} |x^i - y^i|^2}. \tag{2.20}$$

3. L_∞ or Manhattan (Chebyshev) metric

$$D_\infty(X, Y) = \max_i |x^i - y^i|. \tag{2.21}$$

In the codeword search problem, usually the Euclidean metric is used because it fits the physical meaning of distance (or distortion). In order to avoid calculating the division, the squared Euclidean metric is employed instead of the Euclidean metric in pattern matching. This does not affect the result by deleting the square root from the Euclidean metric. Several researchers (Soleymani & Morgera, 1989; Lo & Cham, 1993; Mathews, 1992) also call the squared Euclidean metric as simply the Euclidean metric. For convenience and without causing confusion, the Euclidean metric is also used without the square root in this thesis.

2.2.2 Signal to Noise Ratio Measure

The signal to noise ratio (SNR) (Kitawaki, 1991) measure is appropriate for speech waveform coding. It is one of the common objective measures defined as

$$SNR = 10\log_{10}\frac{\sum_{j=1}^{m} x^{j\,2}}{\sum_{j=1}^{m} |x^j - \hat{x}^j|^2}. \tag{2.22}$$

Here, x^j is an undistorted input speech signal, $\hat{x}^j$ is the distorted output speech signal of waveform coding and m is the number of samples in the speech signal. This measure is also suitable for image coding. Generally, the SNR measure characterizes the ratio of long-term average speech power to long-term average quantizing noise power. The larger-power speech section dominates the long-term calculation of SNR measure. Hence, a smaller-power speech section is neglected, in spite of its importance, such as for consonant or transient periods. This measure can be improved by separating the speech waveform into several frames, taking the same measure over each frame and summing the measurement for all frames. It is named segmental SNR (SNR_{seg}) which is defined as

$$SNR_{seg} = \frac{1}{N}\sum_{i=1}^{N} SNR_i, \tag{2.23}$$

where N is the number of frames and SNR_i is the SNR of the ith frame. The typical duration of the frame is 20 ms for speech segments.

2.2.3 Spectral Distortion Measure

The spectral distortion measure (SD) is an objective measure containing the characteristics of the whole speech spectrum and is defined as

$$SD = [\frac{1}{b}\int_0^b \{S_x(b) - S_y(b)\}^2 db]^{1/2}, \tag{2.24}$$

where S_x and S_y are input and output speech spectra, respectively, and b is the frequency band of the signal. The speech spectrum can be computed from the fast Fourier transform (FFT).

2.2.4 Cepstral Distortion Measure

The cepstrum (Rabiner & Juang, 1993) of a signal is defined as the Fourier transform of the log of the signal spectrum. Given two cepstrum coefficients C_t and C_r in the k-dimensional vector space, the cepstral distortion between C_t and C_r is expressed as

$$D_c(C_t, C_r) = \sum_{i=1}^{k} |c_t^i - c_r^i|^2. \tag{2.25}$$

2.2.5 Quadratic Metric

The quadratic metric is an important generalization of the Euclidean metric. Let Q denote the positive definite matrix, such as the inverse of the covariance matrix, the quadratic metric between vectors X and Y is defined as follows:

$$D_q(X, Y) = (X - Y)^t Q(X - Y). \tag{2.26}$$

One particular case of the quadratic metric is the weighted cepstral distortion measure (Tohkura, 1987). It is defined as

$$D_w(C_t, C_r) = \sum_{i=1}^{k} w_i |c_t^i - c_r^i|^2, \tag{2.27}$$

where w_i is the reciprocal of the ith diagonal element of the covariance matrix of the feature vectors. The most significant characteristic of the weighted cepstral distortion is that it equalizes the importance in each dimension of cepstrum coefficients. In the speech recognition, the weighted cepstral distortion can be used to equalize the performance of the recognizer across different talkers (Pan, 1988).

2.2.6 Itakura-Saito Distortion Measure

The Itakura-Saito distortion measure (O'Shaughnessy, 1987; Rabiner & Juang, 1993; Itakura & Saito, 1970) computes a distortion between two input vectors by using their spectral densities. The definition of this measure is as follows:

$$D_{is}(X, Y) = |\frac{S_x}{S_y} - \ln(\frac{S_x}{S_y}) - 1|, \tag{2.28}$$

where S_x and S_y are the spectral densities of the vectors X and Y.

2.3 Lagrange Multiplier Technique

The *Lagrange multiplier technique* is an efficient method for finding the minimum or maximum values of a function $g(x, y, z)$ subject to a *constraint condition* $\psi(x, y, z) = 0$. It is expressed with the formation of the auxiliary function

$$f(x, y, z, \lambda) = g(x, y, z) + \lambda\psi(x, y, z), \tag{2.29}$$

subject to the conditions

$$\frac{\partial f}{\partial x} = 0, \qquad \frac{\partial f}{\partial y} = 0, \qquad \frac{\partial f}{\partial z} = 0, \tag{2.30}$$

which are necessary conditions for a relative minimum or maximum value and the parameter λ is independent of x, y, and z.

This technique can be generalized to find the minimum or maximum values of a function $g(x_1, x_2, ..., x_n)$ subject to the *constraint conditions* $\psi_1(x_1, x_2, ..., x_n) = 0, \psi_2(x_1, x_2, ..., x_n) = 0, ..., \psi_m(x_1, x_2, ..., x_n) = 0$. The auxiliary function is

$$f(x_1, x_2, ..., x_n, \lambda) = g + \lambda_1\psi_1 + \lambda_2\psi_2 + ... + \lambda_m\psi_m \tag{2.31}$$

subject to the necessary conditions

$$\frac{\partial f}{\partial x_1} = 0, \qquad \frac{\partial f}{\partial x_2} = 0, \qquad ... \qquad , \frac{\partial f}{\partial x_m} = 0, \tag{2.32}$$

where λ_i, $i = 1, 2, ..., m$, is independent of x_j, $j = 1, 2, ..., n$.

2.4 Theory of Vector Quantization

A fundamental purpose of data compression, such as image coding or speech coding, is to reduce the bit rate for transmission or data storage while maintaining the necessary fidelity of the data. One of the simple and essential examples of data compression is the transmission of speech by pulse code modulation (PCM) in which a sampler followed by scalar quantization is used to compress the speech data. According to Shannon's rate-distortion theory, improved

performance is always achievable in theory by coding vectors instead of scalars, even if the data source is memoryless. A vector can be used to represent almost any type of pattern, such as a block of image data by forming the vector which is composed of the values of the pixels in the block; or a segment of speech waveform by forming the vector from the values of the sample points in the segment. The vector may represent a number of different possible speech coding parameters including linear predictive coding (LPC) coefficients, cepstrum coefficients, gain parameters and prediction residual samples. It is also possible to represent the parameters in image coding, such as coefficients of the discrete cosine transform (DCT) or the Walsh-Hadamard transform. Vector quantization can be viewed as a generalization of scalar quantization to the quantization of a vector, an ordered set of real numbers. Fig. 2.1 illustrates the basic

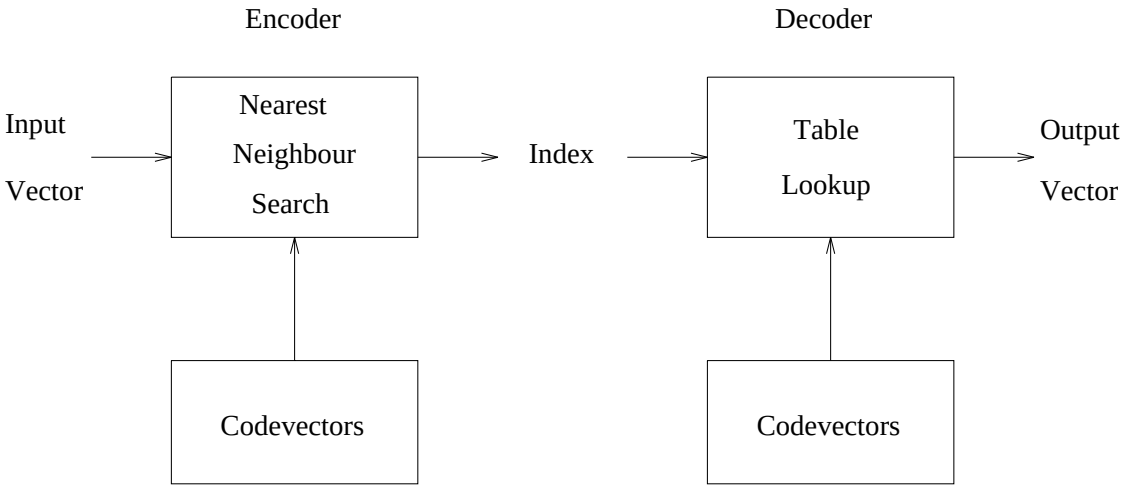

Figure 2.1: Vector quantization diagram

idea of vector quantization (Gersho & Gray, 1992; Gray, 1984; Gersho & Cuperman, 1983; Nasrabadi & King, 1988). The VQ encoder encodes a given set of k-dimensional data vectors $X=\{X_j | X_j \in R^k; j = 1, ..., T\}$ with a much smaller subset $C=\{C_i | C_i \in R^k; i = 1, ..., N\}(N \ll T)$. The subset C is called a codebook and its elements C_i are called codewords, codevectors, reproducing vectors, prototypes or design samples. Only the index i is transmitted to the decoder. The decoder has the same codebook as the encoder, and decoding is operated by table look-up procedure. The performance of data compression depends on creation of a good codebook of representative vectors.

2.4.1 Vector Quantizers

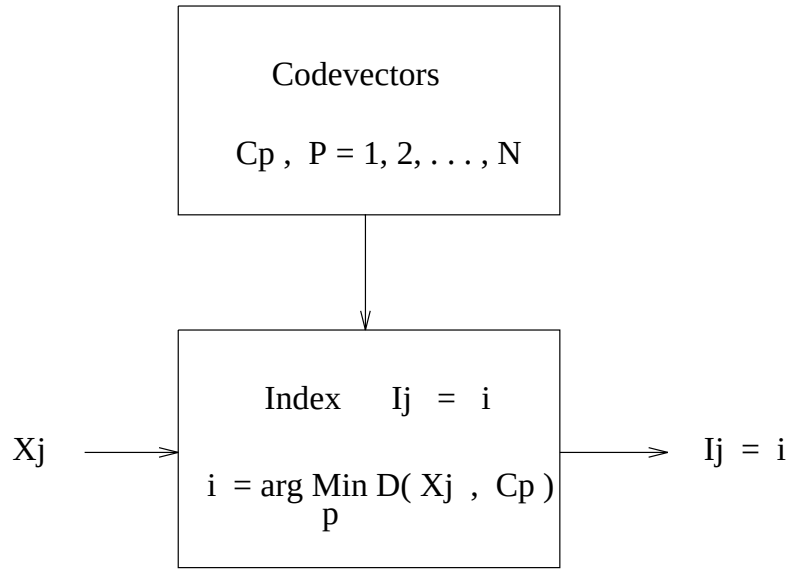

Figure 2.2: Vector quantization encoder

As shown in Fig. 2.2, the index I_j of the jth data vector is i which is transmitted to the receiver if the codeword C_i is the nearest neighbour to the data vector X_j. This class of vector quantizers called *Voronoi* or *nearest neighbour* vector quantizer is particularly useful. The nearest neighbour encoding algorithm is described as follows:

Step 1: Set $d_{min} = \infty$, $p = 1$, $i = 1$.

Step 2: Calculate $d_p = D(X_j, C_p)$.

Step 3: If $d_p < d_{min}$, set $d_{min} = d_p$ and $i = p$.

Step 4: If $p < N$, set $p = p + 1$ and go to step 2.

Step 5: Terminate the search program and record the search index i.

The initial value $d_{min} = \infty$ means that the initial d_{min} is larger than any possible distortion in the decoding approach. The LBG algorithm (Linde *et al.*, 1980) is a popular *VQ* training algorithm which was proposed by Linde, Buzo and Gray and their names are used to refer to this algorithm. LBG based vector quantizer belongs to the class of nearest neighbour quantizer. A *lattice* vector quantizer (Conway & Sloane, 1983; Jeong & Gibson, 1989; Gersho & Gray, 1992) is a different class of vector quantizers whose codebook is either a lattice or a coset of a

lattice or a truncated version of a lattice or its coset so that the codebook size is finite. The lattice based vector quantizer provides design simplicity, reduces encoding complexity and yields high quantization performance especially for large codebook size.

The performance of the vector quantizer can be evaluated by a distortion measure D which is a non-negative cost $D(X_j, \hat{X}_j)$ associated with quantizing any input vector X_j with a reproduction vector $\hat{X}_j$. Usually, the Euclidean distortion measure is used. The performance of a quantizer is always qualified by an average distortion $D_v = E[D(X_j, \hat{X}_j)]$ between the input vectors and the final reproduction vectors, where E represents the expectation operator. Normally, the performance of the quantizer will be good if the average distortion is small. If the data vector process is stationary and ergodic, then the overall measure of performance can be expressed as the long term sample average or time average

$$\overline{D} = \lim_{n\to\infty} \frac{1}{n} \sum_{j=1}^{n} D(X_j - \hat{X}_j) \approx \frac{1}{T} \sum_{j=1}^{T} D(X_j - \hat{X}_j), \tag{2.33}$$

where T is the number of data vectors and it is large enough to qualify the performance.

2.4.2 Speech Coding and Image Coding

The most simple and original application of vector quantization to speech coding is to perform block waveform coding called vector pulse code modulation (VPCM) (Gersho & Cuperman, 1983; Abut *et al.*, 1982) on the speech signal vector. Vector quantization has been applied to the efficient coding of linear predictive coding (LPC) parameters (Kang & Coulter, 1976; Buzo *et al.*, 1980; Wong *et al.*, 1982), parameters of pitch predictor, gain parameters (Chen & Gersho, 1987; Sabin & Gray, 1984), the coding of the excitation or residual signal in analysis-by-synthesis predictive coding techniques, such as vector excitation coding (VXC) (or code excited linear prediction (CELP)) (Davidson *et al.*, 1987; Atal & Schroeder, 1985; Ahmed & Al-Suwaiyel, 1993; Cuperman *et al.*, 1991).

The application of vector quantization on digital images has been investigated in the spatial domain, such as the mean/shape VQ (Budge & Baker, 1985), the classified VQ (Gersho & Ramamurthi, 1982; Ramamurthi & Gersho, 1986) codebook replenishment VQ (Sun & Goldberg, 1985), hierarchical VQ (Nasrabadi, 1985) and the interframe VQ (Goldberg

& Sun, 1986). The goal of transform coding for digital images is to convert statistically dependent or correlated picture elements into independent or uncorrelated coefficients. In the transform domain, vector quantization has been applied to the coding of $adaptive\ transform$ (Saito *et al.*, 1986), $one-dimensional\ transform$ (Nasrabadi & King, 1983), $two-dimensional\ transform$ (Habibi, 1974) and $interframe\ transform$ (Nasrabadi & King, 1984).

2.4.3 Computational Complexity

In the VQ coding area, fidelity increases with the transmission rate r (bits per vector dimension). For a fixed transmission rate r and vector dimension k, the size of a VQ codebook N is 2^{kr}. The search complexity to find a nearest codeword for a given input data vector is $O(k2^{kr})$, i.e., $k2^{kr}$ multiplications, $(2k-1)2^{kr}$ additions and $2^{kr}-1$ comparisons for exhaustive full search (EFS). The search complexity increases exponentially as the vector dimension grows. This is one major drawback of VQ codeword search and it limits the fidelity of coding for real time transmission. In order to reduce the computational cost, two general approaches have been reported. The first proposes fast search algorithms for searching the same codebook (Cheng *et al.*, 1984; Cheng & Gersho, 1986; Ra & Kim, 1991; Huang & Chen, 1990; Lo & Cham, 1993; Soleymani & Morgera, 1987a). The other reported techniques use structured codebook (Juang & Gray, 1982; Lowry *et al.*, 1987; Moayeri *et al.*, 1991; Mohammadi & Holmes, 1994) to achieve efficient codeword search.

2.5 Hidden Markov Model

Signal modelling based on hidden Markov models (HMM) may be considered as a technique that extends conventional stationary spectral analysis principles to the analysis of time-varying signals. Hidden Markov model theory (Forsyth, 1995; Huang *et al.*, 1990; Rabiner & Juang, 1986) has been applied successfully in speech and speaker recognition. The principle of the hidden Markov model is to provide a probabilistic framework for VQ codewords for modelling temporal and contextual information. It is a collection of states connected by transitions which include a set of state transition probabilities and a set of output probability mass functions. The state transition probability is the probability of a state transition occurring. The output

probability mass function defines the conditional probability of each possible output symbol from a finite alphabet given a state.

In the training phase, the $forward-backward$ algorithm and $Baum-Welch$ re-estimation algorithm are generally used to train the sets of state transition probabilities and output probability density functions. In the recognition phase, the $Viterbi$ dynamic programming algorithm can be used to find the optimal assignment of frames to the states, based on maximising the total probability. There are four main categories in hidden Markov models: $discrete$ hidden Markov model ($DHMM$), $continuous$ hidden Markov model ($CHMM$), $semi-continuous$ hidden Markov model ($SCHMM$) and Multi-VQ hidden Markov model ($MVQ\ HMM$). In $DHMM$, VQ codewords are assigned probabilities and the probability of the codeword which is nearest to the feature vector is used as the observation probability. These VQ codewords are shared for all states of all models. In $CHMM$, each state of each model has different $mixture\ Gaussian\ VQ$ functions. If the training data is inadequate, it is difficult to use the parameters of the $Gaussians$ to estimate each state of each model. $SCHMM$ overcomes this difficulty by having a fixed set of $Gaussians$ in a codebook that are shared for all states of all models. $MVQ\ HMM$ is an approach to fill the gap between the $SCHMM$ and $CHMM$ by having different $Gaussian$ codebooks for different models.

2.6 Genetic Algorithms

Genetic algorithms (GA) (Fang, 1994) are a group of methods which solve problems using approaches inspired by the processes of Darwinian evolution. The current genetic algorithms in science and engineering refer to a model introduced and investigated by Holland (Holland, 1975) and by students of Holland. In genetic algorithms, a set of solutions to a problem is called $chromosomes$. A chromosome (string of solution) is composed of $genes$ (features, characters or detectors). Usually, the individual of the whole population contains only one chromosome. The performance of the solution is called $fitness$. The fitness of chromosomes are evaluated and ordered, then new chromosomes are produced by using the selected candidates as $parents$ and applying $mutation$ and $crossover$ operations. The new set of chromosomes is then evaluated and ordered again. This cycle continues until a suitable solution is found. The conventional

genetic algorithm is described as the following steps:

Step 1. *Initialization*: Encode and assign a chromosome to each individual in the population.

Step 2. *Evaluation*: Decode and evaluate each chromosome's fitness.

Step 3. *Selection*: Select the survivors for the next generation from the better fitness of chromosomes. These survivors will be the candidate for the crossover and mutation operation.

Step 4. *Crossover*: Pairs of survivors are selected as the parents to crossover to produce new chromosomes (children) for the next generation.

Step 5. *Mutation*: Mutation is operated among genes in chromosomes randomly.

Step 6. Steps 2 to 5 are repeated until adequate fitness is found.

2.6.1 Initialization

A set of chromosomes is randomly generated. For example, if the problem is to minimize a function of a, b, c and d, then the initial step may be to generate a collection of random vectors (a_i, b_i, c_i, d_i), $i = 1, 2, ..., P$, P is the number of chromosomes or population size. The gene in the chromosome can be binary or non-binary. The length of the chromosome (string) or the number of genes in the chromosome can be fixed or variable. The representation of the chromosome plays an important role in genetic algorithms.

2.6.2 Selection

The goal of selection is to choose the better individuals as the survivors which are then used as the parents and undergo subsequent crossover and mutation operations. Without the selection step, the crossover and mutation operations are useless, i.e., no better offspring could be generated in the next generation. The probability of each parent being selected is the function of its fitness. Even only keeping the best individual as the survivor, with the other survivors selected randomly, and the parents selected from the survivors, the *GA* can perform well.

Roulette Wheel Selection

Roulette Wheel selection (Goldberg, 1989) or fitness-based selection is the original approach for parent selection. In Roulette Wheel selection, the probability of selection for each parent is directly proportional to its fitness. The performance of this selection depends strongly on the range of fitness values in the current population. For example, if the population size is 5 and the fitness values are 1, 10, 100, 500 and 1000, then the probability of selecting the first individual is $\frac{1}{1611}$. There is almost no chance to select the parent whose fitness is relatively low compared with the individual with highest fitness, even if the individual with lower fitness has some important genes. So a few individuals dominate the selection. Another example is where the range of fitness is very narrow, such as 1000 to 1005. Then the probability of selection for each individual is almost the same. These two conditions are undesirable. One way to overcome this difficulty is to scale the fitness before selection (Goldberg, 1989).

Tournament-Based Selection

The basic $\mathtt{tournament}$ selection (Brindle, 1981) is to choose M individuals randomly and return the best one of these. This is generally called size M Tournament selection. After this selection, the crossover and mutation operator are applied to generate a new child. When the fitness of this new child is better than the worst individual of the previous generation, replacement of this worst individual occurs. $\mathtt{Boltzmann\ tournament}$ selection (Goldberg, 1990) evolves a Boltzmann distribution across a population and time using pairwise probabilistic acceptance and anti-acceptance mechanisms.

2.6.3 Crossover

The crossover operator is one of the most important operators in genetic algorithms. The basic idea is to combine some genes from different chromosomes. It is the recombination of bit strings by exchanging the segments between pairs of chromosomes. Many crossover techniques and the example will be illustrated here.

One-Point Crossover

The one point crossover technique is the most simple crossover technique, but it is very efficient. The procedure of one-point crossover is to select one crossover point at random. Genes up to and including the crossover point are copied to the respective child. The remaining genes are copied to the alternate child. Assume the chromosomes of parent1 and parent2 are as follows:

parent1: 3 6 7 1 8 4 2 5

parent2: 1 8 4 7 3 2 5 6

If position 3 is randomly generated as the crossover position, then two children are as follows:

child1: 3 6 7|7 3 2 5 6

child2: 1 8 4|1 8 4 2 5

where the first child has the first three genes from parent1, the others from parent2, the second child has the first three genes from parent2, the others from parent1.

Two-Point Crossover

The procedure of two point crossover is similar to that of one-point crossover except that two positions are selected and only the bits between the two positions are swapped. The first part and last part of chromosomes are preserved. With the same parents above, positions 2 and 5 are generated as the crossover positions, then two children are as follows:

child1: 3 6|4 7 3|4 2 5

child2: 1 8|7 1 8|2 5 6

N-Point Crossover

The procedure of n-point crossover is also similar to one-point crossover except that n positions are selected and only the bits between odd and even crossover positions are swapped. The bits between even and odd crossover positions are unchanged. Assume the chromosomes of parent1 and parent2 are as follows:

parent1: 3 6 6 7 1 4 8 4 2 5 8

parent2: 1 8 6 4 7 3 2 3 2 5 6

If positions 2, 5, 6 and 9 are selected as the crossover positions, then two children can be generated as follows:

child1: 3 6|6 4 7|4|2 3 2|5 8

child2: 1 8|6 7 1|3|8 4 2|5 6

Uniform Crossover

There are two popular multi-point crossover techniques, one is n-point crossover, the other is uniform crossover. In uniform crossover (Syswerda, 1989), each gene is copied from a parent based on a random flip of a fair coin, i.e., each gene of the first parent has a 0.5 probability of swapping with the corresponding gene of the second parent. Assume the chromosomes of parent1 and parent2 are as follows:

parent1: 3 4 5 1 8 6 2 5

parent2: 4 8 3 7 9 2 6 1

A number between 0 and 1 is generated randomly for each position. If the random number generated for a given position is less than 0.5, then child1 copies the gene from parent1, and child2 copies the gene from parent2; otherwise, vice versa. If the random numbers generated for each position are 0.9, 0.4, 0.1, 0.8, 0.6, 0.5, 0.4 and 0.7, then two children are as follows:

child1: 4* 4 5 7* 9* 2* 2 1*

child2: 3* 8 3 1* 8* 6* 6 5*

where the crossover points are marked by the symbol *.

Order-Based Crossover

Order-based crossover technique (Davis, 1991) is used when the search space is a permutation, so that, somehow, crossing 1 3 2 5 4 with 3 2 1 4 5 is always sure to yield another valid permutation, such as 1 5 3 2 4. Let the following be two parents:

parent1: 1 3 2 5 4

parent2: 5 4 2 1 3

One kind of order-based crossover operator works as follows:

Choose two random genes of the first parent, for example

parent1: 1 3* 2 5* 4

Make up the child by first copying the unchosen genes:

child: 1 _ 2 _ 4

and then fill in the other values, 3 and 5, but in the same order as they occur in the second parent, yielding:

child: 1 5 2 3 4

The uniform order-based crossover is a powerful order-based crossover technique. In this kind of crossover, several gene positions of the chromosome are chosen randomly and the order in which these genes appear in the second parent is imposed on the first parent to produce offspring. The genes in the other positions are the same as the first parent.

2.6.4 Mutation

Selection and crossover effectively search and recombine the chromosomes, but occasionally they may lose some potentially useful genes and it is also possible that some useful genes are not generated in the initial step. A better result cannot be reached for lack of some useful genes. This difficulty can be overcome by using the mutation technique. The basic mutation operator is to randomly generate a number as the crossover position and then change the value of this gene randomly. For example, if the length of the chromosome is 6 and a chromosome after crossover is

1 4 7 2 8 3

A random number generator generates a position 2 as the gene position and the other random number generator generates any valid gene value, such as 6, then the chromosome is mutated to

$1\ 6^*\ 7\ 2\ 8\ 3$

Another possible approach is to check each gene position using a random number compared with the mutation rate, if this number is less than the mutation rate, then this gene needs to mutate. Assume the mutation rate is 0.01, if the random number is 0.005 for the first position and the random numbers are larger than the mutation rate in the other positions, then this gene is mutated by a random number in the first position, such as 7. The result is as following:

$7^*\ 4\ 7\ 2\ 8\ 3$

If the search space is a permutation, the mutation operation can work by swapping several genes in the chromosome randomly.

2.6.5 Inversion

In the procedure of inversion, two points are chosen at random along the length of the chromosome and the order of the genes between these two points is inverted. Only one parent is needed in the inversion operation. If two positions 3 and 6 are chosen and the inversion operator is applied to the string

$4\ 7\ 2^*\ 5\ 9\ 8^*\ 3$

then the new string is

$4\ 7\ 8^*\ 9^*\ 5^*\ 2^*\ 3$

2.6.6 Schema Theorem

A schema H is a pattern of gene values which may be represented by a string of binary symbols $\{0, 1\}$ and a symbol # which matches any gene values. For example, the chromosome "01011" contains, among others, the schemata "#101#", "#10##", "01##1" and "#1011". The order of a schema denoted by $o(H)$ is the number of non-# symbols in the schemata. The defining length of a schema denoted by $\delta(H)$ is the distance between the outermost non-# symbols. In this example, the order is 3, 2, 3 and 4 respectively; the defining length is 3, 2, 5 and 4 respectively.

$m(H,t)$ denotes the frequency of a schema H at generation t. It will change for the next generation in proportion to the selection of probability of strings. Let $m(H,t+1)$ be the frequency of schema H at generation $t+1$. The relationship between $m(H,t)$ and $m(H,t+1)$ can be expressed as the following formula:

$$m(H,t+1) = m(H,t)\frac{f(H)}{\bar{f}} \tag{2.34}$$

where $f(H)$ is the average fitness of a string containing schema H in generation $t+1$ and $\bar{f}$ represents the average fitness of the whole population.

Let P_c be the crossover probability and l be the length of the string. Because a schema survives when the crossover point is selected outside the defining length, the survival probability under simple crossover is

$$P_s \geq 1 - P_c\frac{\delta(H)-1}{l-1}. \tag{2.35}$$

So, if the reproduction and simple crossover operation are independent, the frequency of schema for the next generation can be estimated as following:

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}[1 - P_c\frac{\delta(H)-1}{l-1}]. \tag{2.36}$$

The probability that the given schema H exists in the next generation will be high if the length of the string l is long and the defining length $\delta(H)$ is short.

If the mutation operator is applied, a single gene will survive with probability $1-p_m$, where p_m is the mutation probability. Since each of the mutations is statistically independent, the probability of surviving mutation is $(1-p_m)^{o(H)}$. This is approximated by the expression $1 - P_m o(H)$ if $p_m \ll 1$. Hence, if reproduction, crossover and mutation operators are applied, the frequency of schema for the next generation can be expressed as following:

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}[1 - P_c\frac{\delta(H)-1}{l-1} - P_m o(H)]. \tag{2.37}$$

This equation implies that the short, low order, above average schemata receives exponentially increasing probability in the subsequent generations, i.e., highly fit schemata of low order and short defining length are particularly important to genetic algorithms.

2.7 Parallel Processing

A conventional computer uses one processor which executes a set of instructions in order to produce results. At any instant time, there is only one operation being carried out by the processor. Parallel processing is concerned with producing the same results using multiple processors. The goal of using parallel processing is to reduce the running time in a computer system.

Two basic parallel processing methods are pipeline processing and data parallelism. The principle of pipeline processing is to separate the problem into a cascade of tasks where each of the tasks is executed by an individual processor. As shown in Fig. 2.3, data is transmitted through each processor which executes a different program on each of the data elements. Since the program is distributed over the processor in the pipeline and the data moves from one processor to the next, no processor can proceed until the previous processor has finished its task and passed the data to it. Data parallelism is a popular approach which involves distributing all the data to

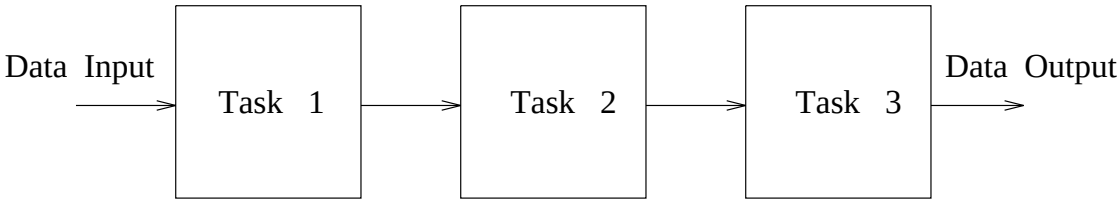

Figure 2.3: Task and data distribution of pipeline processing

be processed equally amongst all the processors in the computer. As shown in Fig. 2.4, each processor contains the same program task operating on the subset of the data. Data parallelism can be easily applied to genetic algorithms by dividing the population into several groups and running the same algorithm for each group at the same time using different processors. This is called a *parallel genetic algorithm* (PGA). The purpose of applying parallel processors to genetic algorithms is more than just a hardware accelerator. Rather a distributed formulation is developed which gives better solutions with less computation. In order to reach this function,

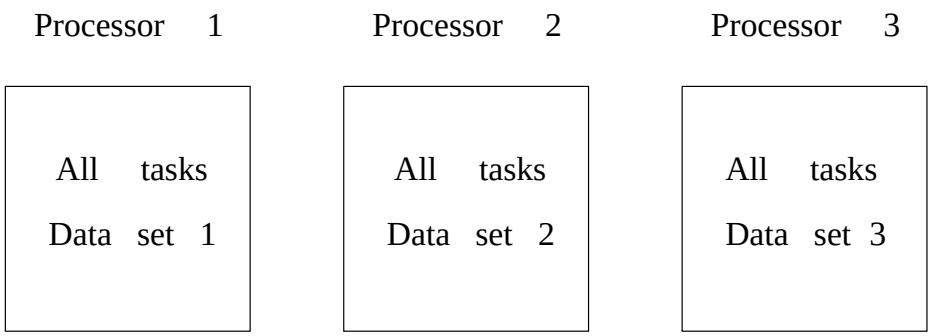

Figure 2.4: Task and data distribution of data parallelism

the communication among these groups is executed for some fixed generations, i.e., the parallel genetic algorithm periodically selects promising individuals from each subpopulation and migrates them to different subpopulations. With the migration (communication), each subpopulation will receive some new and promising chromosomes to replace the worst chromosomes in this subpopulation. This helps to avoid premature convergence.

2.8 Bound for Minkowski Metric

Given one codeword C_t and the test vector X in k-dimensional space, the distortion of the Minkowski metric of order n can be expressed as follows:

$$D_{min} = D(X, C_t) = \sum_{i=1}^{k} |x^i - c_t^i|^n \tag{2.38}$$

where $C_t = \{c_t^1, c_t^2, ..., c_t^k\}$ and $X = \{x^1, x^2, ..., x^k\}$.

The generalized bound for the Minkowski metric based on the L_p distortion measure can be found as follows:

$$\text{If} \qquad \sum_{i=1}^{s} |x^i - c_j^i|^p \geq \sqrt[\frac{n}{p}]{h^{\frac{n}{p}-1} D_{min}} \tag{2.39}$$

$$\text{then} \qquad \sum_{i=1}^{k} |x^i - c_j^i|^n \geq D_{min} \tag{2.40}$$

where $s \leq h \leq k$ and $p \leq n$.

If $p = n$, then Eq. 2.39 reduces to Eq. 2.40. For the case where $p < n$, the bound can be proved

as follows :

Apply Lagrange multiplier technique to

$$\text{minimize} \qquad \sum_{i=1}^{h} a_i^m \tag{2.41}$$

$$\text{subject to} \qquad \sum_{i=1}^{h} a_i = c, \qquad a_i \geq 0 \quad \forall i. \tag{2.42}$$

If the minimum is at an interior point, then it is a stationary point of $f(a_i, \lambda) = \sum_{i=1}^{h} a_i^m - \lambda(\sum_{i=1}^{h} a_i - c)$ with respect to $a_i (1 \leq i \leq h)$ and λ.

Taking derivatives, $\frac{\partial f}{\partial a_i} = m a_i^{m-1} - \lambda = 0 \; \forall i$.

Hence $a_i = (\lambda/m)^{\frac{1}{m-1}} \; \forall i$ (which implies $a_i = a_j \; \forall i, j$) and so, to make $\frac{\partial f}{\partial \lambda} = 0$, $a_i = c/h \; \forall i$.

Here $\sum_{i=1}^{h} a_i^m = \sum_{i=1}^{h} (c/h)^m = h(c/h)^m = h^{1-m} c^m$.

The next step is to prove that the climax $\sum_{i=1}^{h} a_i^m = h^{1-m} c^m$ is the minimum point, and so to prove the following proposition.

$$\text{If} \qquad \sum_{i=1}^{h} a_i = c \tag{2.43}$$

$$\text{then} \qquad \sum_{i=1}^{h} a_i^m \geq h^{1-m} c^m \tag{2.44}$$

where $m > 1$, $h \geq 1$, and $a_i \geq 0$ for all i.

This can be proved by induction. When $h = 1$, Eq. 2.43 reduces to $a_1 = c$ and Eq. 2.44 to $a_1^m \geq c^m$. Hence the proposition is true for $h = 1$. Assume it is true for $h - 1$. By using the Lagrange multiplier technique, if the minimum of $\sum_{i=1}^{h} a_i^m$ is at an interior point ($a_i > 0$ for all i), then this must be at the point where $a_i = c/h$ for all i, at which point $\sum_{i=1}^{h} a_i^m = h^{1-m} c^m$. At a non-interior point (without loss of generality, a_h=0), $\sum_{i=1}^{h} a_i^m = \sum_{i=1}^{h-1} a_i^m \geq (h-1)^{1-m} c^m > h^{1-m} c^m$.

The minimum cannot be at the non-interior point since the value there is greater than at the

interior point already found and hence the value where $a_i = c/h$ is in fact the minimum. The proof is completed.

Hence if $c \geq \sqrt[m]{h^{m-1} D_{min}}$, then $\sum_{i=1}^{h} a_i^m \geq h^{1-m} c^m \geq h^{1-m}(h^{m-1} D_{min}) = D_{min}$.

Set $a_i = b_i^p$, hence if $\sum_{i=1}^{h} b_i^p \geq \sqrt[m]{h^{m-1} D_{min}}$, then $\sum_{i=1}^{h} b_i^{pm} \geq D_{min}$.

Set $pm = n$, hence if $\sum_{i=1}^{h} b_i^p \geq \sqrt[\frac{n}{p}]{h^{\frac{n}{p}-1} D_{min}}$, then $\sum_{i=1}^{h} b_i^n \geq D_{min}$.

Set $b_i = |x^i - c_j^i|$, hence $\sum_{i=1}^{h} |x^i - c_j^i|^p \geq \sum_{i=1}^{s} |x^i - c_j^i|^p$ if $h \geq s$, then the bound for Minkowski metric based on L_p metric is derived.

If Eq. 2.39 is met, then C_j cannot be the nearest neighbour to X for the Minkowski metric of order n.

This bound has the following properties :

1. Set $s = p = h = 1$ and $n = 2$, the hypercube approach.
2. Set $p = 2$ and $n = 2$, the partial distortion search (PDS) for the Euclidean distortion measure.
3. Set $p = n$, PDS for L_p distortion measure.
4. Set $n = 2$, $p = 1$ and $h = k$, absolute error inequality (AEI) criterion.
5. Set $n = 2$ and $p = 1$, defined here as the improved absolute error inequality (IAEI) criterion, provides a tighter bound than the absolute error inequality (AEI) criterion.
6. For the Minkowski metric of order n, this bound provides the elimination criterion from L_1 metric to L_n metric and also provides an advanced approach by adapting parameters s and h from 1 to k, i.e., this bound can be separated into several sections. For 13-dimensional coefficients and the Euclidean metric, it is possible to separate this bound into four sections. These four sections are to set $h = 1$ to check the first dimension-difference, $h = 4$ for the sum from the first dimension-difference to the fourth, $h = 9$ for the sum from the first dimension-difference to the ninth and $h = 13$ for the sum of all dimension-differences.

2.9 Bound for Quadratic Metric

In speech recognition, the hidden Markov model (HMM) with the Gaussian mixture VQ codebook probability density function has been shown to be a promising method. The main computation time is in searching the nearest neighbour by evaluating the log likelihood of Gaussian mixture distributions, i.e., the calculation of

$$\log \frac{1}{(2\pi)^{k/2}|W_m|^{1/2}} e^{-\frac{1}{2}(X-C_m)^t W_m^{-1}(X-C_m)} \tag{2.45}$$

That is to compute $\frac{k}{2}\log(2\pi) + \frac{1}{2}\log|W_m| + \frac{1}{2}(X - C_m)^t W_m^{-1}(X - C_m)$, where m is from 1 to N and N is the number of mixture components. C_m and W_m are the mean value and the covariance of mixture component m. Obviously, the quadratic metric $(X - C_m)^t W_m^{-1}(X - C_m)$ dominates the computation time.

For convenience and brevity, assume that the covariance of every mixture component m is the same. The quadratic metric can be expressed as

$$D(X, C_m) = (X - C_m)^t W^{-1}(X - C_m) \tag{2.46}$$

where $(X - C_m)$ is error column vector and W is the covariance matrix given as:

$$W = \frac{1}{T}\sum_{i=1}^{T}(X_i - \bar{X})(X_i - \bar{X})^t \tag{2.47}$$

where T is the number of training vectors and $\bar{X}$ is the mean of X_i, $i = 1, ..., T$, i.e.,

$$\bar{X} = \frac{1}{T}\sum_{i=1}^{T} X_i \tag{2.48}$$

W^{-1} is the inverse of the covariance matrix W. For the conventional exhaustive method, $k(k+1)N$ multiplications, $(k^2 + k - 1)N$ additions and $N - 1$ comparisons are needed for every test frame.

2.9.1 Metric Transform Using Triangular Matrix

The quadratic metric is transformed to the Euclidean metric using the lower triangular matrix and the upper triangular matrix in this subsection. By applying the improved absolute error

inequality criterion to the metric transform, the bound for quadratic metric is obtained.

Given that W^{-1} can be represented in terms of the product of the lower triangular matrix and the upper triangular matrix according to Eq. 2.49 as

$$W^{-1} = LL^t \tag{2.49}$$

$$L = \begin{bmatrix} l_{11} & l_{21} & l_{31} & \dots & l_{k1} \\ 0 & l_{22} & l_{32} & \dots & l_{k2} \\ 0 & 0 & l_{33} & \dots & l_{k3} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & l_{kk} \end{bmatrix} \tag{2.50}$$

Set $E_m = X - C_m$, then the quadratic metric can be expressed as follows:

$$D(X, C_m) = E_m^t LL^t E_m = |E_m^t L|^2 \tag{2.51}$$

Set $L = [V_1 V_2 V_3 \dots V_k]$, and assume

$$D_{min} = D(X, C_m) = \sum_{i=1}^{k} |E_m^t V_i|^2 \tag{2.52}$$

$$\text{If} \qquad \sum_{i=1}^{s} |E_j^t V_i| \geq \sqrt{hD_{min}}, \tag{2.53}$$

$$\text{then} \qquad D(X, C_j) \geq D_{min} \tag{2.54}$$

where $s \leq h \leq k$.

After modification of the quadratic metric to Eq. 2.51, the improved absolute error inequality

(IAEI) criterion can be easily applied as shown in Eq. 2.52 to Eq. 2.54.

2.9.2 Metric Transform Using KLT

The Karhunen-Loêve transform (KLT) is also called the eigenvector transform, principal component transform and Hotelling transform. It is an optimal transform in a statistical sense under a variety of criteria. The KLT has the following properties (Elliott, 1982):

1. It is the best vector transform in the sense of decorrelating the sequence completely in the transform domain.
2. It packs the most energy (variance) to the low order elements.
3. It minimizes the mean squared error (MSE) between the original and reconstructed data for any specified bandwidth reduction or data compression.
4. It minimizes the total entropy of the data sequence.

Eigenvectors of the covariance matrix of a given sequence are the basis functions of the KLT. Assume P and Λ are the eigenvector and diagonal matrix of eigenvalues, respectively. The quadratic metric can be transformed to the Euclidean metric as follows:

$$D(X, C_m) = (X - C_m)^t W^{-1}(X - C_m)$$

$$= (X - C_m)^t \{P\Lambda P^t\}^{-1}(X - C_m)$$

$$= (X - C_m)^t \{P \begin{bmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ 0 & 0 & \lambda_3 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \lambda_k \end{bmatrix} P^t\}^{-1}(X - C_m)$$

$$= (X - C_m)^t \{P^{t-1} \begin{bmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ 0 & 0 & \lambda_3 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \lambda_k \end{bmatrix}^{-1} P^{-1}\}(X - C_m)$$

$$= (X - C_m)^t \{P \begin{bmatrix} \frac{1}{\lambda_1} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{\lambda_2} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{\lambda_3} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \frac{1}{\lambda_k} \end{bmatrix} P^t\}(X - C_m)$$

$$= (X - C_m)^t \{P \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{\lambda_2}} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{\sqrt{\lambda_3}} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \frac{1}{\sqrt{\lambda_k}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{\lambda_2}} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{\sqrt{\lambda_3}} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \frac{1}{\sqrt{\lambda_k}} \end{bmatrix} P^t\}(X - C_m)$$

$$= (X - C_m)^t Q Q^t (X - C_m) = U^t U = \sum_{i=1}^{k} |u^i|^2, \qquad (2.55)$$

where

$$Q = P \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{\lambda_2}} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{\sqrt{\lambda_3}} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \frac{1}{\sqrt{\lambda_k}} \end{bmatrix},$$

$U = Q^t(X - C_m)$ and u^i is the element of the row matrix U.

Apply the IAEI to Eq. 2.55 and assume the current minimum distortion

$$D_{min} = D(X, C_m). \tag{2.56}$$

$$\text{If} \qquad \sum_{i=1}^{s} |u^i| \geq \sqrt{hD_{min}}, \tag{2.57}$$

$$\text{then} \qquad D(X, C_j) \geq D_{min} \tag{2.58}$$

where $s \leq h \leq k$.

After the transform of quadratic metric to Eq. 2.55 using KLT, another bound for quadratic metric is derived as shown in Eq. 2.56 to Eq. 2.58.

Chapter 3

Efficient Codeword Search Algorithms

Vector Quantization (VQ) (Gray, 1984; Gersho & Cuperman, 1983; Buzo *et al.*, 1980) has been widely used for various applications involving VQ-based encoding and VQ-based recognition. The response time of encoding and recognition is a very important factor to be considered for real-time applications. Unfortunately, a full search algorithm is applied in VQ encoding and recognition and this is a time consuming process when the codebook size is large. A vector quantizer of rate r bits/sample and dimension k is a mapping from a k-dimensional vector space into some finite subset $C = \{C_j; j = 1, ..., N\}$, where $N = 2^{kr}$. The subset C is called a codebook and its elements C_j are called codewords, codevectors, reproducing vectors, prototypes, or design samples. A distortion measure $D(X, C_j)$ is a non-negative dissimilarity measure between vector X and codewords C_j. This distortion is used to measure how close the input vector X is to these codewords C_j. The nearest codeword is to be selected in order to encode the input vector X. Therefore, encoding each input vector requires N distortion computations and $N - 1$ comparisons.

The codeword search problem in vector quantization is to assign one codeword to the test vector in which the distortion between this codeword and the test vector is the smallest among all codewords. Given one codeword C_t and the test vector X in the k-dimensional space, the distortion of the squared Euclidean metric can be expressed as follows:

$$D(X, C_t) = \sum_{i=1}^{k} (x^i - c_t^i)^2, \tag{3.1}$$

where $C_t = \{c_t^1, c_t^2, ..., c_t^k\}$ and $X = \{x^1, x^2, ..., x^k\}$.

Each distortion calculation requires k multiplications and $2k - 1$ additions. Therefore, it is necessary to perform $k2^{kr}$ multiplications, $(2k - 1)2^{kr}$ additions, and $2^{kr} - 1$ comparisons for encoding each input vector. The computation complexity depends on codebook size and dimensions. It needs large codebook size and higher dimension for high performance in VQ encoding and recognition systems resulting in increased computation load during codeword search.

3.1 History of Codeword Search

Since codeword search is a serious problem in real time application of vector quantization, the history of codeword search will be introduced first, then a series of efficient methods will be presented in this chapter.

3.1.1 Partial Distortion Search

The partial distortion search (PDS) algorithm (Bei & Gray, 1985) is a simple and efficient codeword search algorithm which allows early termination of the distortion calculation between a training vector and a codeword by introducing a premature exit condition in the search process. Given the current minimum distortion,

$$D(X, C_t) = D_{min}, \tag{3.2}$$

$$\text{if} \qquad \sum_{i=1}^{s} (x^i - c_j^i)^2 \geq D_{min}, \tag{3.3}$$

$$\text{then} \qquad D(X, C_j) \geq D(X, C_t), \tag{3.4}$$

where $s \leq k$.

The efficiency of PDS derives from elimination of an unfinished distortion computation if its partial accumulated distortion is larger than the current minimum distortion. This will reduce computation to $(k - s)$ multiplications and $2(k - s)$ additions at the expense of s comparisons. The detail algorithm of the partial distortion search is described as follows:

Step 0: Set $D_{min} = \infty$ (a very large number), $i = 1$, and $j_{min} = j = 1$.

Step 1: If $j > N$, then terminate the algorithm and record j_{min} as the index of the nearest codeword; otherwise $D = 0$.

Step 2: $D = D + (x^i - c_j^i)^2$.

Step 3: If $D > D_{min}$, then $j = j + 1$ and go to step 1; otherwise go to step 4.

Step 4: If $i < k$, then $i = i + 1$ and go to step 2; otherwise $D_{min} = D$, $j_{min} = j$, $j = j + 1$ and go to step 1.

The efficiency of the partial distortion search (PDS) algorithm can be further improved by ordering the codewords (Paliwal & Ramasubramanian, 1989). This requires calculation of the probability P_i for each codeword from the training data: P_i is the probability of the codeword C_i which is nearest neighbour to the training data. The codewords are then arranged in the codebook in the order of decreasing P_i. After this arrangement, the probability of obtaining the nearest codeword in the early stage can be increased which helps in saving computation time.

3.1.2 Hypercube Approach

The hypercube approach is a well known premature method (Lo & Cham, 1993) which is efficient if the difference for any coefficient is generally larger than the difference of the other coefficients, such as the first coefficient of cepstrum coefficients. Assume Eq. 3.2 has already existed,

$$\text{if} \qquad |x^i - c_j^i| \geq \sqrt{D_{min}}, \qquad 1 \leq i \leq k, \tag{3.5}$$

then C_j will not be the nearest neighbour to X.

There is no multiplication operation required for the test of the hypercube approach.

3.1.3 Absolute Error Inequality Criterion

The absolute error inequality (AEI) criterion (Soleymani & Morgera, 1987b) is the mathematical relationship between the city block metric (or L_1) and the Euclidean metric (or L_2). Assume C_t is the current nearest neighbour to X, that is,

$$D(X, C_t) = D_{min},$$

$$\text{if} \qquad \sum_{i=1}^{s} |x^i - c_j^i| \geq \sqrt{kD_{min}}, \tag{3.6}$$

$$\text{then} \qquad \sum_{i=1}^{k} (x^i - c_j^i)^2 \geq D_{min}, \tag{3.7}$$

where $s \leq k$.

This means C_j will not be the nearest neighbour to X if Eq. 3.6 is satisfied. This criterion can be estimated by comparing the first dimension-difference of the test vector and codeword with the right hand side of Eq. 3.6. If Eq. 3.6 is not satisfied for $s = 1$, then this criterion is checked for higher s. This criterion is checked by increasing s until $s = k$ or the criterion is satisfied.

3.1.4 Triangular Inequality Elimination

Triangular inequality elimination (Pan, 1988) is an efficient method for codeword search. Let V be the set of data vectors and C be the set of codewords and x, y belong to the set V. On V, a distortion measure is defined as a mapping d: $V \times V \rightarrow R$, which is assumed to fulfill the metric properties:

$$d(x, y) \geq 0; \; d(x, y) = 0 \quad \text{iff} \quad x = y \tag{3.8}$$

$$d(x, y) = d(y, x) \tag{3.9}$$

$$d(x, y) + d(y, z) \geq d(x, z) \tag{3.10}$$

As shown in Fig. 3.1, let C_1, C_2, C_3 be three different codewords and t be a test vector, then the following three criteria are obtained.

- Criterion 1:

  Given the triangular inequality

$$d(t, C_2) + d(t, C_1) \geq d(C_1, C_2); \tag{3.11}$$

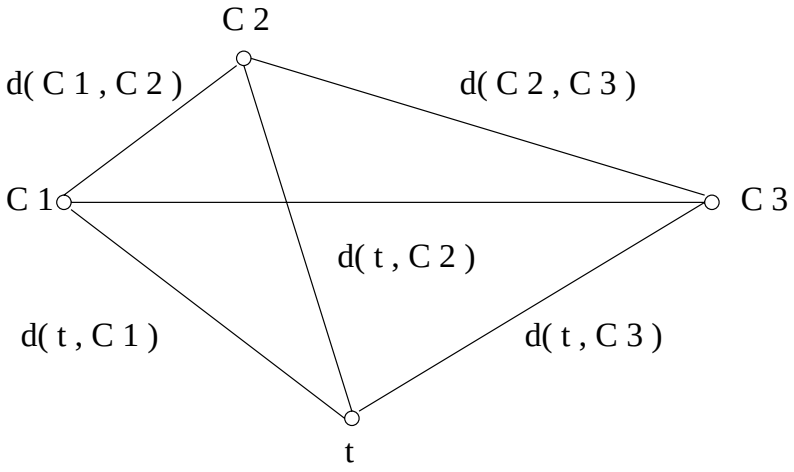

Figure 3.1: Distortion diagram of test sample and codewords

$$\text{if} \qquad d(C_1, C_2) \geq 2.d(t, C_1), \tag{3.12}$$

$$\text{then} \qquad d(t, C_2) \geq d(t, C_1). \tag{3.13}$$

- Criterion 2:

  Given the triangular inequality

$$d(C_3, C_2) \leq d(t, C_2) + d(t, C_3); \tag{3.14}$$

$$\text{if} \qquad d(C_3, C_2) \geq d(t, C_1) + d(t, C_2), \tag{3.15}$$

$$\text{then} \qquad d(t, C_1) \leq d(t, C_3). \tag{3.16}$$

- Criterion 3:

$$\text{Assume} \qquad d(t, C_1) \leq d(t, C_2).$$

$$\text{Given} \qquad d(C_3, C_2) \geq d(t, C_2) - d(t, C_3); \tag{3.17}$$

$$\text{if} \qquad d(C_3, C_2) \leq d(t, C_2) - d(t, C_1), \tag{3.18}$$

$$\text{then} \qquad d(t, C_1) \leq d(t, C_3). \tag{3.19}$$

Criterion 2 and 3 can be merged to one criterion only, i.e.,

$$\text{if} \qquad d(t, C_1) \leq |d(C_3, C_2) - d(t, C_2)|, \tag{3.20}$$

$$\text{then} \qquad d(t, C_1) \leq d(t, C_3). \tag{3.21}$$

To use Criterion 1, these distortions between all pairs of codewords are calculated in advance. If Eq. 3.12 is met, then the computation of $d(t, C_2)$ can be omitted if $d(t, C_1)$ has already been computed. Criterion 1 can be modified for square error distortion measure. In the codeword searching system, a table is made to store one-fourth of the values of square distortion between codewords, i.e., store the value of $d^2(C_i, C_j)/4$, for $i = 1, 2, ..., N$; $j = 1, 2, ..., N$. Here N is the number of codewords. The overhead of criterion 1 is to establish the distortion table in which $N(N-1)k/2$ multiplications and $N(N-1)(2k-1)/2$ additions are needed. As shown in Fig. 3.2, the physical meaning of Criteria 2 and 3 can be described as follows :

If the codeword $C_i, i \neq 1, 2$, does not locate between the two concentric circles (or in general hyperspheres) centered on C_2 with radii $d(t, C_2) \pm d(t, C_1)$, the computation of its distortion to the test sample can be omitted, i.e., if $d(C_i, C_2) > d(t, C_2) + d(t, C_1)$ or $d(C_i, C_2) < d(t, C_2) - d(t, C_1)$, then eliminate the computation of C_i. For the special case $d(t, C_1) = d(t, C_2)$, Criterion 3 is inappropriate and Criterion 2 reduces to Criterion 1. Since Criterion 2 and 3 will induce square root computation, it is simple and efficient to use Criterion 1 only.

3.1.5 Approximating and Eliminating Search Algorithm

The approximating and eliminating search algorithm ($AESA$) was proposed by (Vidal, 1986). The detail of this algorithm is described as follows:

Step 0: Calculate $\frac{N(N-1)}{2}$ distortions for every possible pair of codewords, N is the number of codewords.

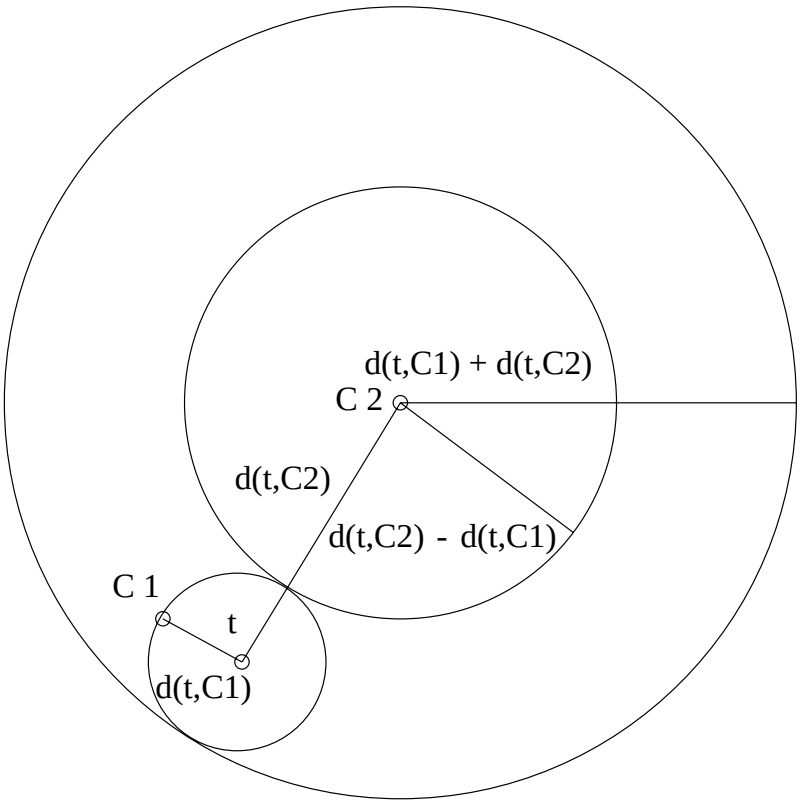

Figure 3.2: Geometric diagram for Criteria 2 and 3 of triangular inequality elimination

Step 1: Compute $d(X, C_i)$, C_i is a selected codeword and X is a data vector. Set $U = \{C_i\}$ and $n = i$. Here n is codeword index of the current nearest codeword and U is a set of used codewords.

Step 2: Eliminate codeword C_j if $d(C_j, C_i) \geq 2d(X, C_i)$.

Step 3: If all codewords are eliminated or used, then terminate the program; otherwise, $s = \mathrm{argmin}_p(\sum_{C_l \in U} |d(C_p, C_l) - d(X, C_l)|)$. C_p is an unused and non-eliminated codeword.

Step 4: Calculate $d(X, C_s)$ and $U = U \cup \{C_s\}$.

Step 5: Find the current nearest codeword index $n = \mathrm{argmin}_{p \in \{n,s\}} d(X, C_p)$. If $n = s$, then $Q = U$; otherwise, $Q = \{C_s\}$.

Step 6: Eliminate the unused and non-eliminated codeword C_j if $d(C_j, C_q) \geq d(X, C_q) + d(X, C_n)$ or $d(C_j, C_q) \leq d(X, C_q) - d(X, C_n)$, where $C_q \in Q$. Go to step 3.

In step 0, the distortions for every possible pair of codewords are calculated off line. Steps 1 and 2 are initializations of this algorithm. Criterion 1 of the triangular inequality elimination used in step 2 is not efficient because the value of $d(X, C_i)$ may be large, since C_i is selected randomly. The tentative matching codeword is found in step 3. In step 4, the distortion between the data vector and the tentative matching codeword is calculated. The nearest codeword is updated in step 5. The potential for matching impossible codewords is eliminated using Criteria 2 and 3 of the triangular inequality elimination in the last step. The main effect of this algorithm is to find an efficient tentative matching codeword and then Criteria 2 and 3 of triangular inequality elimination are applied to eliminate impossible codeword matching. Here, the tentative matching codeword is a non-eliminated and unused codeword which satisfies Eq. 3.22.

$$C_s = \min_{C_p}\left(\sum_{C_l \in U} |d(C_p, C_l) - d(X, C_l)|\right). \tag{3.22}$$

This tentative matching codeword is the approximation of the nearest to the intersection of all hyperspheres with radius $d(X, C_l)$, $\forall C_l \in U$.

3.1.6 Minimax Method

The minimax method (Cheng *et al.*, 1984) is to take the codeword with the minimum value of the maximum dimension-distortion as the tentative match and then use the hypercube approach and the partial distortion search (PDS). The minimax method is depicted as follows :

Step 1: For the given test vector X and codebook C, calculate the absolute error $e_{ij} = |x^i - c_j^i|$, $i = 1, 2, ..., k$, $j = 1, 2, ..., N$.

Step 2: Find the maximum component of each error vector, that is to find $\max_i e_{ij}$ for each codeword. For convenience, interchange the maximum component of error vector with e_{1j}.

Step 3: Find the minimum neighbour l =arg $\min_j \max_i e_{ij}$.

Step 4: Find the squared Euclidean distortion $D_{min} = \sum_{i=1}^{k} e_{il}^2$.

Step 5: Use the hypercube approach, i.e., if $\max_i e_{ij} \geq \sqrt{D_{min}}$, then c_j will not be the nearest neighbour to X. Use the PDS to delete the rest of the codewords.

3.1.7 Previous Vector Candidate

For speech data, the classification result of the present vector is usually the same as or close to the classified result of the previous vector. The nearest codeword of the previous vector can be used as the tentative match called previous vector candidate which is first proposed by (Pan, 1988; Chen & Pan, 1989).

In vector quantization of images, data are first divided into subsequent blocks of size $k = M \times M$. The previous vector candidate has also been applied to image coding (Huang & Chen, 1990) by taking the advantage of high correlation between contiguous subimages. Let $C(m, n)$ denote the nearest codeword of the block image $X(m, n)$. As shown in Fig. 3.3, the nearest codewords of the four adjacent blocks are used as the tentative matching codewords, i.e., calculate the distortions between the data vector $X(i, j)$ and codewords $C(i-1, j)$,$C(i-1, j-1)$,$C(i, j-1)$ and $C(i+1, j-1)$. The codeword with the minimum distortion is chosen as the candidate. Then the Criterion 1 of the triangular inequality elimination is used to eliminate impossible codeword matching. Partial distortion search (PDS) is used as the last stage to calculate the distortion for the rest of the codewords. The previous vector candidate has also been applied to image coding using vector quantization by (Ngwa-Ndifor & Ellis, 1991). Only one codeword $C(i-1, j)$ is used as the tentative match and only the partial distortion search (PDS) is applied in this algorithm. The previous vector candidate, Criterion 1 of the triangular inequality elimination and the partial distortion search were also applied to Manhattan (Chebyshev) metric for VQ image coding by (Nyeck *et al.*, 1992).

3.1.8 Subcodebook Search Algorithm

A subcodebook search (SCS) algorithm (Lo & Cham, 1993) was developed for efficient VQ encoding of images. This algorithm also takes the advantage of high correlation between two adjacent blocks. The control codeword is one of the four nearest codewords of the four adjacent blocks which has the smallest distortion to the current encoding block. In the training phase, the decision distortion for each control codeword is decided from Eq. 3.23.

$$D_i = \sum_{j=1}^{N} P_{C_i}(C_j) D(C_j, C_i), \tag{3.23}$$

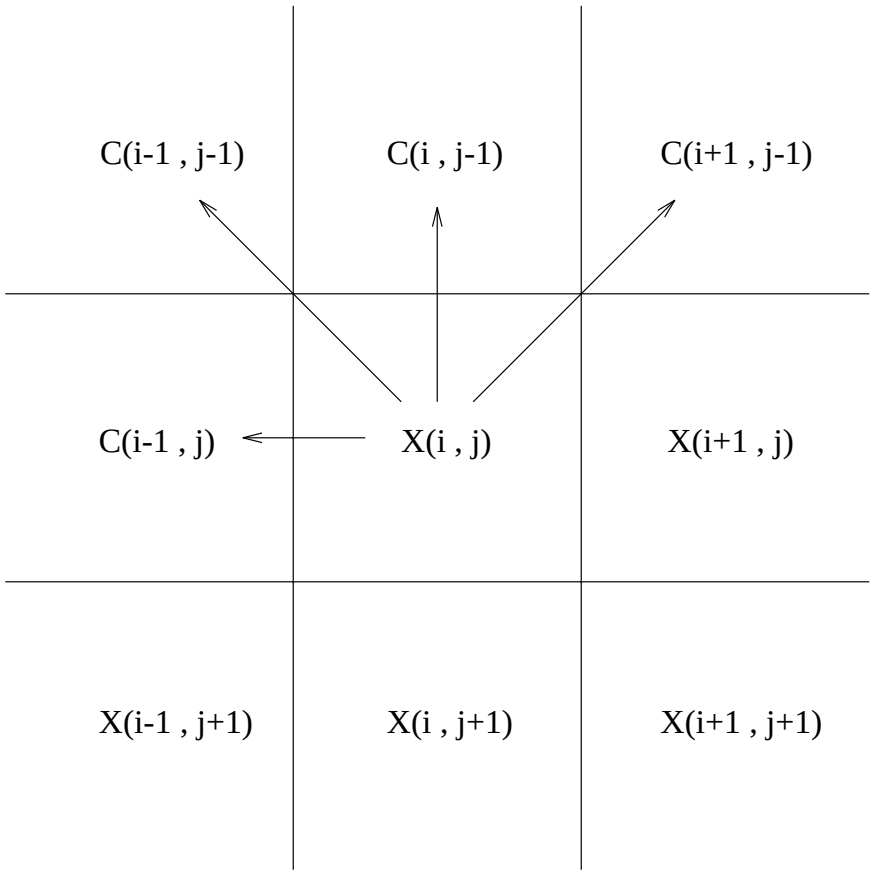

Figure 3.3: Diagram of four adjacent codewords for image coding

where $1 \leq i \leq N$, $P_{C_i}(C_j)$ is the probability that the best match to the training data vectors is the codeword C_j when the control codeword is C_i and the squared Euclidean distortion measure is applied.

The subcodebooks can be constructed by grouping those codewords having distortion to the control codeword C_i smaller or equal to $4D_i$. The additional memory requirements for SCS algorithm are two tables: the decision distortion for each subcodebook and the mapping codeword indices for each subcodebook.

In the encoding phase, the control codeword C_i is determined first. Compare the distortion $D(X, C_i)$ and the decision distortion D_i, here X is the data vector. If $D(X, C_i) \geq D_i$, then search the whole codebook; otherwise, search the corresponding subcodebook. In searching the whole codebook or subcodebook, the partial distortion search and hypercube approach are applied.

3.1.9 Fast Sliding Search Algorithm

The fast sliding codeword search algorithm was presented by (Koh & Kim, 1988). This algorithm uses the codeword with the most similar sum of components to the data vector as the tentative matching codeword, i.e., find a codeword C_i such that

$$i = \mathrm{argmin}_l \left| \sum_{j=1}^{k} x^j - \sum_{j=1}^{k} c_l^j \right|. \tag{3.24}$$

In the training phase, the sum of all dimensions for each codeword is calculated first and these values are sorted in increasing order or decreasing order. In the encoding phase, the tentative matching codeword is obtained by using Eq. 3.24. Then p codewords are searched from $C_{i-\frac{p}{2}}$ to $C_{i+\frac{p}{2}-1}$ which is illustrated in Fig. 3.4. If $i - \frac{p}{2} \leq 1$, then search the codeword from C_1 to C_p. If $i + \frac{p}{2} - 1 \geq N$, then search the codeword from C_{N-p+1} to C_N. This algorithm is an approximate search algorithm.

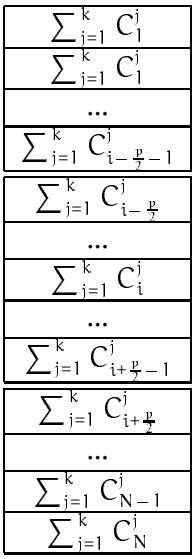

Figure 3.4: Search strategy of fast sliding search algorithm

3.1.10 Equal-average hyperplane partitioning method

The equal-average hyperplane partitioning method for vector quantization of image was proposed by (Guan & Kamel, 1992). This method utilizes hyperplanes orthogonal to the central line **l** to partition the search space. Any point on **l** has the same value for every dimension. As explained in (Lee & Chen, 1994), each point on a fixed hyperplane **H**, which is orthogonal to

the central line **l** and intersects **l** at point $L_H = (m_H, m_H, ..., m_H)$, will have the same mean value m_H, such a hyperplane is called an *equal average hyperplane*.

In the training phase of the equal-average hyperplane partitioning method, the sum of all dimensions for each codeword is calculated and divided by k first. These values are sorted in increasing order. It is similar to the *fast sliding search* algorithm. In the encoding phase, the mean of the data vector is calculated as

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

Then the tentative matching codeword is found first by using the same method as the *fast sliding search* algorithm, i.e., calculate Eq. 3.25.

$$i = \arg\min_l |m_x - \frac{1}{k}\sum_{j=1}^{k} c_l^j|. \tag{3.25}$$

Compute the distortion between this data vector X and the tentative matching codeword C_i,

$$d_i = \sqrt{\sum_{j=1}^{k} (x^j - c_i^j)^2}.$$

Any other codeword which is closer to the data vector X than the tentative matching codeword C_i will be located inside the hypersphere centred at X with radius d_i. Projecting the hypersphere on **l**, two boundary projection points $L_{max} = (m_{max}, m_{max}, ..., m_{max})$ and $L_{min} = (m_{min}, m_{min}, ..., m_{min})$ on **l** can be found, where

$$m_{max} = m_x + \frac{d_i}{\sqrt{k}} \tag{3.26}$$

and

$$m_{min} = m_x - \frac{d_i}{\sqrt{k}}. \tag{3.27}$$

Hence, only the codewords with mean value from m_{min} to m_{max} are searched. The equal-average hyperplane partitioning method uses the mean value to eliminate unlikely codewords and hence much computation time is saved. This algorithm is further improved by introducing the following formula (Lee & Chen, 1994):

$$\text{if} \qquad |V_X - V_{C_p}| \geq d_i, \tag{3.28}$$

$$\text{then} \qquad d_p \geq d_i, \tag{3.29}$$

where

$$V_X = \sqrt{\sum_{j=1}^{k} (x^j - m_x)^2} \tag{3.30}$$

and

$$V_{C_p} = \sqrt{\sum_{j=1}^{k} (c_p^j - m_{C_p})^2}. \tag{3.31}$$

If Eq. 3.28 is met, then C_p will not be the nearest codeword to data vector X. By testing Eq. 3.28 first, if it is not satisfied, then check if

$$m_p \geq m_{max} \tag{3.32}$$

or

$$m_p \leq m_{min}. \tag{3.33}$$

If they are still not met, then calculate the distortion between X and C_p. Note that memory size of $N(k+2)$ is needed for this algorithm compared with $N(k+1)$ for the equal-average hyperplane partitioning method.

3.1.11 Fast Full Search Equivalent Encoding Algorithm

The fast full search equivalent encoding algorithms (Huang *et al.*, 1992) utilize the minimum mean distance as the tentative matching approach, then apply the three criteria of triangular inequality elimination to reject unlikely codeword matching. The first algorithm uses the minimum mean distance as the tentative matching codeword and Criterion 1 of the triangular inequality elimination as the elimination method which is described as follows:

Step 1: Compute $r_x = \sum_{j=1}^{k} x^j$.

Step 2: Find codeword C_i, such that $i = \text{argmin}_l |\sum_{j=1}^{k} x^j - \sum_{j=1}^{k} c_l^j|$.

Step 3: Calculate the L_1 distortion $d_{min} = d(X, C_i) = \sum_{j=1}^{k} |x^j - c_i^j|$.

Step 4: Check the termination of this program. If $\frac{d(C_p, C_i)}{2} \geq d_{min}$, then omit the distortion calculation of codeword C_p, set $p = p + 1$ and goto step 4; otherwise, goto next step.

Step 5: Calculate $d(X, C_p)$, update $d_{min} = min\{d(X, C_p), d_{min}\}$ and $i = argmin_l\{d(X, C_l)\}$, set $p = p + 1$ and goto step 4.

From Criteria 2 and 3, codeword C_p can be eliminated if it does not satisfy the following inequality:

$$d(X, Z) - d(X, C_i) \leq d(Z, C_p) \leq d(X, Z) + d(X, C_i), \tag{3.34}$$

where $d(X, Z)$ is the L_1 distortion between data vector X and any other vector Z. By setting the vector Z to the origin, this inequality can be rewritten as

$$\sum_{j=1}^{k} |x^j| - d(X, C_i) \leq \sum_{j=1}^{k} |c_p^j| \leq \sum_{j=1}^{k} |x^j| + d(X, C_i). \tag{3.35}$$

If Eq. 3.35 is not met, then eliminate codeword C_p. Combining Eq. 3.35 with the minimum mean distance as the tentative matching approach is the algorithm 2 in (Huang *et al.,* 1992). Combining Criterion 1 of triangular inequality elimination, Eq. 3.35 and the minimum mean distance as the tentative matching approach is the algorithm 3. In terms of the total number of mathematical operations, algorithm 1 outperforms the other two algorithms. In terms of the number of multiplications, algorithm 3 is superior to the other two algorithms.

3.1.12 Adaptive Fast Encoding Algorithm

From Eq. 3.34, the vector Z can be set to any value. This inequality will be very efficient if small values of $d(X, Z)$ and $d(X, C_i)$ are selected. Eq. 3.34 provides different constraints on the test codewords for different values of vector Z. In previous work (Salari & Li, 1994), three values of vector Z are selected such that three smaller $d(X, Z)$ are provided. Hence codeword C_p can be eliminated if C_p cannot satisfy any of the following three inequalities:

$$d(X, Z_0) - d(X, C_i) \leq d(Z_0, C_p) \leq d(X, Z_0) + d(X, C_i),$$

$$d(X, Z_1) - d(X, C_i) \leq d(Z_1, C_p) \leq d(X, Z_1) + d(X, C_i),$$

$$d(X, Z_2) - d(X, C_i) \leq d(Z_2, C_p) \leq d(X, Z_2) + d(X, C_i),$$

where $d(Z_m, C_p)$ can be calculated off line, m=1,2 and 3, p=1,2,...,N. Z_0 is set to origin and codewords are sorted in ascending order of $d(Z_0, C_p)$. Three tables A_0, A_1 and A_2 are built to store ascending ordered values of $d(Z_0, C_p)$, $d(Z_1, C_p)$ and $d(Z_2, C_p)$, respectively. Two index tables B_1 and B_2 are used to store the codeword indices corresponding to the ordered $d(Z_1, C_p)$ and $d(Z_2, C_p)$ tables. In the encoding phase, only the codewords C_p satisfying these three inequalities simultaneously are needed to compute the distortion, i.e., the final subset of codewords is

$$\{C_p : l_1 \le B_2(p) \le l_2, n_1 \le B_1(p) \le n_2, m_1 \le p \le m_2\},$$

where m_1 is the index of the first element of A_0 whose value exceeds $d(X, Z_0) - d(X, C_i)$ and m_2 is the the index of the last element of A_0 whose value is smaller than $d(X, Z_0) + d(X, C_i)$; n_1 is the index of the first element of A_1 whose value exceeds $d(X, Z_1) - d(X, C_i)$ and n_2 is the the index of the last element of A_1 whose value is smaller than $d(X, Z_1) + d(X, C_i)$; l_1 is the index of the first element of A_2 whose value exceeds $d(X, Z_2) - d(X, C_i)$ and l_2 is the index of the last element of A_2 whose value is smaller than $d(X, Z_2) + d(X, C_i)$. If any codeword cannot be eliminated using these three inequalities, then calculate the distortion of $d(X, C_p)$ and update the current nearest codeword and the current minimum distortion.

3.1.13 Fast MMSE Encoding Technique

The fast minimum mean squared error ($MMSE$) encoding technique (Soleymani & Morgera, 1989) assumes codewords C_i, $i = 1, 2, ..., N$, partition the space R^k into N regions S_i, $i = 1, 2, ..., N$, such that

$$S_i = \{X : \sum_{p=1}^{k} (x^p - c_i^p)^2 \le \sum_{p=1}^{k} (x^p - c_j^p)^2, \quad \text{all } j\}.$$

For each codeword C_i, let $r_i = \sqrt{D_i}$ where

$$D_i = \max_{X \in S_i} \sum_{p=1}^{k} (x^p - c_i^p)^2. \tag{3.36}$$

For any given input data vector X, if

$$|x^p - c_i^p| > r_i, \tag{3.37}$$

for some $p \epsilon \{1, 2, ..., k\}$, then C_i will not be the nearest codeword to X. Combine Eq. 3.37 with the hypercube approach such that the codeword C_i can be rejected if

$$|x^p - c_i^p| > r_i^{'}, \tag{3.38}$$

where $r_i^{'} = min\{r_i, \sqrt{D_{min}}\}$ and D_{min} is the current minimum distortion found before checking the codeword C_i.

From the training data, calculate r_i for each codeword C_i, $i = 1, 2, ..., N$ and sort r_i in increasing order and also sort the codebook accordingly. Hence, after finding some codeword C_l such that

$$\sum_{p=1}^{k} (x^p - c_l^p)^2 = D_{min} \leq r_l^2, \tag{3.39}$$

there is no need to compare $\sqrt{D_{min}}$ with r_i for $i > l$. The fast MMSE encoding technique can be depicted as follows:

Step 1: Set $i = 1$ and $j = 1$.

Step 2: While $i < N$ (N is the number of codewords), calculate step 3 to step 6.

Step 3: Calculate $e_{ij} = |x^j - c_i^j|$.

Step 4: If $e_{ij} > r_i$, set $i = i + 1$, $j = 1$ and go to step 2; otherwise, if $j < k$ (k is the number of dimensions), set $j = j + 1$ and go to step 3.

Step 5: Calculate $dm = r_i^2$, $d = e_{i1} * e_{i1}$. Set $j = 2$.

Step 6: If $j > k$, set $m = i + 1$, $D_{min} = d$ and go to step 7. Calculate $d = d + e_{ij} * e_{ij}$, if $d > dm$, set $i = i + 1$ and go to step 2; otherwise, set $j = j + 1$ and go to step 6.

Step 7: Set $j = 1$. While $m < N$, calculate step 8 to step 11.

Step 8: Calculate $e_{mj} = |x^j - c_m^j|$.

Step 9: If $e_{mj} > \sqrt{D_{min}}$, set $m = m + 1$ and go to step 7. otherwise, if $j < k$, set $j = j + 1$ and go to step 8.

Step 10: Calculate $d = e_{m1} * e_{m1}$. Set $j = 2$.

Step 11: Calculate $d = d + e_{mj} * e_{mj}$, if $d > D_{min}$, set $m = m + 1$ and go to step 7; otherwise, set $j = j + 1$ and go to step 11.

Step 12: Set $D_{min} = d$, $m = m + 1$, record C_m as the current nearest codeword to vector X and go to step 7.

In this algorithm, the hypercube approach and PDS are combined with r_i from step 1 to step 6; the hypercube approach and PDS are also combined with the current minimum distortion D_{min} from step 7 to step 12. The main idea of this algorithm is to create r_i from the training data and sort r_i in increasing order.

In addition, t_i, the maximum dimension-distortion for the input data vector in S_i, can be used instead of r_i, defined as

$$t_i = max_{X \epsilon S_i} max_p |x^p - c_i^p|. \tag{3.40}$$

Since the maximum dimension-distortion is less than the square-root of the total distortion, using t_i instead of r_i may result in a more efficient algorithm. Note that the fast MMSE encoding technique is an approximate search algorithm and occasional encoding errors will happen in the nearest neighbour assignment. In order to reduce the encoding errors, a small value added to r_i or t_i is needed.

3.1.14 Projection Method

For the projection method of codeword search for vector quantization (Cheng *et al.*, 1984), Eq. 3.41 and Eq. 3.42 are computed from the training data.

$$T_{ij1} = max_{X \epsilon C_j} x^i. \tag{3.41}$$

$$T_{ij0} = min_{X \epsilon C_j} x^i. \tag{3.42}$$

Sort T_{ijm} in the increasing order for each dimension, where $i = 1, 2, ..., k$, $j = 1, 2, ..., N$ and $m = 0 or 1$. There are $2N - 1$ contiguous intervals for each dimension. For each dimension, create a table where the lth column indicates whether c_l is a candidate and the pth row indicates that x_i is located in the pth interval. The entry of this table can be 1 or 0 to express candidate or

non-candidate. It can also be the index of the codevector for the candidate. In the encoding stage, the interval for each dimension is determined first. The possible candidates are the candidates for the intersection of these k tables given the row for every table. This method is an approximate search algorithm. Occasional encoding errors will occur in the codeword search. In order to reduce the encoding error, Eq. 3.41 and Eq. 3.42 can be modified as follows:

$$T_{ij1} = max_{X \in C_j} x^i + \delta_{j1}, \tag{3.43}$$

$$T_{ij0} = min_{X \in C_j} x^i - \delta_{j0}, \tag{3.44}$$

where δ_{j1} and δ_{j0} are small values and $j = 1, 2, ..., k$. The values of δ_{j1} and δ_{j0} can be decided from experiments.

3.2 Improved Absolute Error Inequality Criterion

The improved absolute error inequality criterion (IAEI) (Pan *et al.*, 1995a; Pan *et al.*, 1995b) is a special case of the bound for Minkowski metric (Pan *et al.*, 1996b). IAEI criterion can be depicted as follows:

$$\text{if} \qquad \sum_{i=1}^{s} |x^i - c_j^i| \geq \sqrt{hD_{min}}, \tag{3.45}$$

$$\text{then} \qquad \sum_{i=1}^{k} (x^i - c_j^i)^2 \geq D_{min}, \tag{3.46}$$

where $s \leq h \leq k$.

The IAEI criterion can also be proved as follows:

Let $a_i = |x^i - c_j^i|$ and $s \leq h \leq k$. Then

$$0 \leq \sum_{i=1}^{h} (a_i - \sum_{j=1}^{h} \frac{a_j}{h})^2 = \sum_{i=1}^{h} a_i^2 - \frac{2}{h} \sum_{i=1}^{h} a_i \sum_{j=1}^{h} a_j + \sum_{i=1}^{h} (\sum_{j=1}^{h} \frac{a_j}{h})^2 = \sum_{i=1}^{h} a_i^2 - \frac{1}{h} (\sum_{i=1}^{h} a_i)^2. \tag{3.47}$$

Hence

$$\sum_{i=1}^{k} a_i^2 \geq \sum_{i=1}^{h} a_i^2 \geq \frac{1}{h} (\sum_{i=1}^{h} a_i)^2 \geq \frac{1}{h} (\sum_{i=1}^{s} a_i)^2. \tag{3.48}$$

Hence if $\sum_{i=1}^{s} a_i \geq \sqrt{hD_{min}}$,

then Eq. 3.48 becomes $\sum_{i=1}^{k} a_i^2 \geq \frac{1}{h}(\sqrt{hD_{min}})^2 = D_{min}$.

The main difference between the IAEI criterion and the AEI criterion is that s and h are used instead of k in Eq. 3.45. Since h can be adapted with s which can be set to a smaller value than k, the IAEI criterion provides a tighter bound than the AEI criterion.

3.2.1 Fast Algorithms Using IAEI

The fast algorithm is generated using the codeword with the minimum value of the maximum dimension-distortion as the tentative match and applying the improved absolute error inequality (IAEI) criterion and partial distortion search (PDS). This new fast codeword searching algorithm is described as follows :

Step 1: For the given test vector X and codebook C, calculate the absolute error $e_{ij} = |x^i - c_j^i|$, $i = 1, 2, ..., k$, $j = 1, 2, ..., N$.

Step 2: Find the maximum component of each error vector, that is to find $max_i e_{ij}$ for each codeword. For convenience, interchange the maximum component of error vector with e_{1j}.

Step 3: Find the minimum neighbour l =arg $min_j max_i e_{ij}$.

Step 4: Find the square Euclidean distortion $D_{min} = \sum_{i=1}^{k} e_{il}^2$.

Step 5: If $\sum_{i=1}^{s} e_{ij} \geq \sqrt{hD_{min}}$, then c_j will not be the nearest neighbour to X, where $s \leq h \leq k$. Use the PDS to delete the rest of the codewords.

In this new fast codeword search algorithm, for $s = h = 1$, it is the same as the hypercube approach in step 5 of the minimax method. By adapting the values of s and h from 1 to k, this algorithm eliminates a very large number of multiplications.

3.2.2 Minimax method with AEI approach

In this section, the new fast codeword search algorithm using IAEI described in the previous sub-section is compared with the minimax method as well as the minimax method including the absolute error inequality criterion. The approach of the minimax method including AEI is described as follows :

Step 1: For the given test vector X and codebook C, calculate the absolute error $e_{ij} = |x^i - c_j^i|$, $i = 1, 2, ..., k$, $j = 1, 2, ..., N$.

Step 2: Find the maximum component of each error vector, that is to find $max_i e_{ij}$ for each codeword. For convenience, interchange the maximum component of error vector with e_{1j}.

Step 3: Find the minimum neighbour $l = \arg\ min_j max_i e_{ij}$.

Step 4: Find the square Euclidean distortion $D_{min} = \sum_{i=1}^{k} e_{il}^2$.

Step 5: Use the hypercube approach, i.e., if $max_i e_{ij} \geq \sqrt{D_{min}}$, then delete the codeword c_j. Use the AEI criterion, i.e., if $\sum_{i=1}^{s} |x^i - c_j^i| \geq \sqrt{kD_{min}}$, then c_j will not be the nearest neighbour to X, where $s \leq k$. Use the PDS to delete the rest of the codewords. Here the AEI criterion is applied by adapting s from 1 to k.

3.2.3 Experiments

The test materials for these experiments consisted of two hundred words recorded from one male speaker. The speech was sampled at a rate of 16 kHz and 13-dimensional cepstrum coefficients with inverse variance weighting were computed over 20 ms-wide frames with a 5 ms frame shift. The purpose of inverse variance weighting is to equalize the importance of every cepstrum coefficient. A total of 20,030 analyzed frames were used in the codeword searching experiments. Codebooks of size 64, 256 and 1,024 codewords with Euclidean distortion measure are used in these experiments.

Experiments were carried out to test the performance of the minimax method; the minimax method with absolute error inequality elimination rule; and the new fast search algorithm described above. The bounds for IAEI were separated into four sections. These four sections were to set $h = 1$ to check the first dimension-difference, $h = 4$ for the sum from the first dimension-difference to the fourth, $h = 9$ for the sum from the first dimension-difference to the ninth and $h = 13$ for the sum of all dimension-differences. The choice of $h = 4$ and $h = 9$ allows the expression $\sqrt{hD_{min}}$ in the elimination test (Eq. 3.45) to be evaluated using only additions, once $\sqrt{D_{min}}$ has been computed, since $\sqrt{4D_{min}} = 2\sqrt{D_{min}}$ and $\sqrt{9D_{min}} = 3\sqrt{D_{min}}$.

Fig. 3.5 illustrates the experimental results for the elimination probability of IAEI at each

feature dimension for 16, 64, 256 and 1024 codewords, respectively. For 1024 codewords, 92.7% of impossible codewords matches will be eliminated by using the IAEI criterion in the first dimension. Only 0.65% of codewords cannot be eliminated using the IAEI criterion. The numbers of eliminations at each dimension for 8, 32, 128 and 512 codewords are shown in Table 3.1. No codeword can be eliminated in the second or fifth dimension and only a few codewords are eliminated in the tenth dimension because the bounds of the IAEI criterion are separated into four sections and h is set to 1, 4, 9 and 13. If h is set to i at the ith dimension, then significant multiplication overhead is needed in the computation of $\sqrt{hD_{min}}$. The statistics of the elimination probability for IAEI criterion at each feature dimension for 16, 64, 256 and 1024 codewords, respectively, is depicted in Fig. 3.6 where h is set to i at the ith dimension. Table 3.2 shows the number of eliminations at each dimension for 8, 32, 128 and 512 codewords, where h is also set to i at the ith dimension. The elimination efficiency for h set to i at the ith dimension is better than h set to 1, 4, 9 and 13 but significant multiplication overhead is needed if h is set to i at the ith dimension. Experimental data relating to computational complexity are depicted

	number of codewords			
dimension	8	32	128	512
1	36,876	336,053	1,911,039	9,023,539
2	0	0	0	0
3	3,450	13,445	34,107	73,228
4	13,800	46,887	112,603	228,730
5	0	0	0	0
6	1,057	5,866	14,536	32,387
7	4,530	20,270	53,834	122,545
8	9,355	32,694	81,887	175,040
9	13,071	36,575	86,146	172,657
10	181	1,074	3,507	7,646
11	3,865	12,788	34,829	71,259
12	8,355	21,331	50,946	99,451
13	10,533	26,944	56,529	94,848

Table 3.1: Number of eliminations at each dimension (h is set to 1, 4, 9 and 13)

in Tables 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9 and 3.10. Table 3.10 shows that this new fast codeword search algorithm saves more than 77% and 21% multiplication operations compared with the minimax method and the minimax method with AEI criterion respectively for 1,024 codewords. Although several digital signal processing chips exist that can implement addition

	number of codewords			
dimension	8	32	128	512
1	36,876	336,053	1,911,039	9,023,539
2	9,025	32,074	75,067	154,884
3	7,246	21,708	49,707	99,302
4	5,364	19,394	48,129	97,968
5	4,521	16,657	42,762	94,449
6	5,605	19,125	47,840	102,112
7	4,731	18,720	48,622	106,165
8	5,035	17,731	44,523	94,010
9	6,486	17,634	41,062	81,403
10	5,831	15,699	38,525	75,705
11	5,299	14,786	35,937	68,688
12	5,245	12,760	30,023	56,990
13	5,446	14,530	31,330	52,575

Table 3.2: Number of eliminations at each dimension (h is set to i for the ith dimension)

and multiplication in approximately the same time, multipliers take up much larger chip areas than adders. Also since the multiplication operation is more expensive than the comparison and addition operations for general processors (Leibson, 1993), this new fast algorithm is better than the other two algorithms.

This fast algorithm is implemented by using the IAEI, setting h to 1, 4, 9 and 13, adapting and comparing Eq. 3.45 for s from 1 to 13. Another possible approach is to adapt s from 1 to 13 but only compare Eq. 3.45 at s = 1, 4, 9 and 13. As shown in Table 3.11, this approach will decrease the number of comparisons as well as the total number of operations at the expense of more additions. In terms of the total number of mathematical operations, this approach is a little better than the minimax method but drastically reduces the number of multiplications for 1024 codewords.

A fast codeword search algorithm must include two key elements: a good tentative matching approach and a powerful elimination criterion. The IAEI is a powerful elimination criterion. An efficient algorithm can therefore be implemented by combining IAEI with another tentative matching approach, such as the previous vector candidate (Pan, 1988; Pan *et al.*, 1996c; Chen & Pan, 1989).

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
Minimax	1.023	2,833	2,973	6,829
Minimax_AEI	721	3,675	3,815	8,211
IAEI_Euclidean	700	3,480	3,679	7,859

Table 3.3: Computational complexity of codeword search for 8 codewords on Euclidean metric

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
Minimax	1,575	5,535	5,515	12,625
Minimax_AEI	985	7,084	7,064	15,133
IAEI_Euclidean	932	6,662	6,708	14,302

Table 3.4: Computational complexity of codeword search for 16 codewords on Euclidean metric

3.3 Improvement in Partial Distortion Search

The PDS algorithm (Bei & Gray, 1985) has been shown to be an efficient and simple codeword search algorithm. This method is always used in the last stage when the other elimination criterion cannot delete impossible codeword matching. As shown previously in section 3.1, subsection 3.1.1, this reduces to $(k - s)$ multiplications and $2(k - s)$ additions at the expense of s comparisons. This algorithm is suitable for computer architectures in which the complexity of comparisons is negligible with respect to that of multiplications. However, PDS is less suited to processor architectures in which comparisons are computationally expensive. An improvement of the partial distortion search algorithm using dynamic programming (DP) procedure (Fissore *et al.*, 1993) is called DPPDS. Here a new improved PDS method (Pan *et al.*, 1994b) is proposed by determining which dimension is suitable to start inserting comparisons for every codeword assessed from the training data.

Let r be the cost ratio of the comparison computation time to dimension-distortion computation

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
Minimax	2,037	10,390	10,049	22,476
Minimax_AEI	1,130	12,673	12,332	26,135
IAEI_Euclidean	1,045	11,921	11,650	24,616

Table 3.5: Computational complexity of codeword search for 32 codewords on Euclidean metric

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
Minimax	2,865	20,035	19,053	41,953
Minimax_AEI	1,400	23,581	22,600	47,581
IAEI_Euclidean	1,256	22,262	21,353	44,871

Table 3.6: Computational complexity of codeword search for 64 codewords on Euclidean metric

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
Minimax	3,827	38,750	36,487	79,064
Minimax_AEI	1,644	43,926	41,663	87,233
IAEI_Euclidean	1,422	41,765	39,577	82,764

Table 3.7: Computational complexity of codeword search for 128 codewords on Euclidean metric

time. The improved partial distortion search (improved PDS) algorithm can be described as follows:

Step 1: Set $l = 1$.

Step 2: Set $i = 1$ and $d_{min} = \infty$.

Step 3: Calculate the distortion d for the ith codeword to the lth training vector. Compute the saving dimension-distortion number M_{jl}^i and the induced comparison number C_{jl}^i at the insertion from the jth dimension for the ith codeword. Set $d_{min} = Min(d_{min}, d)$.

Step 4: If $i < N$, set $i = i + 1$ and go to step 3. Here N is the number of codewords.

Step 5: If $l < T$, set $l = l + 1$ and go to step 2; otherwise, set $c_j^i = T^{-1} \sum_{l=1}^{T} C_{jl}^i$ and $m_j^i = T^{-1} \sum_{l=1}^{T} M_{jl}^i$. Here $l = 1$ to T and T is the number of training vectors. The comparison starts from $I(i)$ for the ith codeword if $I(i) = argMax_j(m_j^i - rc_j^i)$.

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
Minimax	5,088	75,640	70,813	151,541
Minimax_AEI	1,901	83,135	78,308	163,344
IAEI_Euclidean	1,584	79,700	74,949	156,233

Table 3.8: Computational complexity of codeword search for 256 codewords on Euclidean metric

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
Minimax	6,492	148,517	138,562	293,571
Minimax_AEI	2,115	158,799	148,844	309,758
IAEI_Euclidean	1,708	153,730	143,852	299,290

Table 3.9: Computational complexity of codeword search for 512 codewords on Euclidean metric

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
Minimax	7,569	292,892	272,682	573,143
Minimax_AEI	2,133	305,783	285,573	593,489
IAEI_Euclidean	1,671	299,002	278,865	579,538

Table 3.10: Computational complexity of codeword search for 1024 codewords on Euclidean metric

A more efficient algorithm can be developed by combining this improved PDS algorithm with the dynamic programming in the PDS method. It is referred to as improved DPPDS. The difference between the improved DPPDS and the DPPDS algorithm (Fissore *et al.*, 1993) is that in the improved algorithm the dimensions at which comparisons are performed are determined separately for each codeword instead of being the same for all the codewords. Assume S_j^i is the number of successful comparisons for the ith codeword at position j for T data vectors. Hence the number of dimension-distortion computations for inserting comparison operations in position j for the ith codeword can be expressed as

$$N_d^i(j, k) = T.j + (T - S_j^i)(k - j), \tag{3.49}$$

codeword no.	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
8	700	2,796	3,720	7,216
16	932	5,443	6,795	13,170
32	1,045	10,221	11,805	23,071
64	1,256	19,711	21,613	42,580
128	1,422	38,205	39,975	79,602
256	1,584	74,784	75,548	151,916
512	1,708	147,301	144,707	293,716
1024	1,671	291,360	279,969	573,000

Table 3.11: Computational complexity for comparison inserted only in s =1, 4, 9 and 13

where k is the number of dimensions, i=1,2,...,N, N is the number of codewords.

If the previous comparison is performed in position j, the number of dimension-distortion computations for inserting comparisons in position t for the ith codeword is as follows:

$$N_d^i(j, t) = T.j + (T - S_j^i)(t - j) + (T - S_t^i)(k - t). \tag{3.50}$$

The computational advantage for the ith codeword is

$$V^i(j, t) = [N_d^i(j, k) - N_d^i(j, t)] - r(T - S_t^i) = (S_t^i - S_j^i)(k - t) - r(T - S_t^i). \tag{3.51}$$

The dynamic programming technique and Eq. 3.51 are applied to Eq. 3.52, i.e., find suitable inserting positions j to maximize Eq. 3.52.

$$A^i(t) = A^i(j) + V^i(j, t), \tag{3.52}$$

where $t = 1, 2, \ldots, k, j < t$.

The speech databases used in training and test experiments consist of one hundred words recorded from five male speakers separated into three sets. The sampling rate used is 16 kHz and 12-dimensional cepstrum coefficients are computed over 20 ms-wide frames with a 5 ms frame shift. The first data set recorded from two speakers is used to generate the codebook. The inserting dimension of comparison to every codeword for the improved PDS algorithm is computed from the codebook using the second data set recorded from two other speakers. The third data set recorded from the fifth speaker is used to test the performance of these approaches.

The 12-dimensional cepstrum coefficients with variance weighting and 256 codewords are used in the experiment of PDS, improved PDS, improved DPPDS and dynamic programming in PDS referred as DPPDS. The purpose of variance weighting is to equalize the importance of every cepstrum coefficient. The experimental results are shown in Table 3.12. The performance is compared with the standard PDS. These results show that the performance of the improved PDS is almost the same as using DP to improve the performance of PDS. General speaking, if the cost ratio of the computer architecture defined as the comparison computation time divided by the dimension distortion computation time is smaller or equal to 1.2, it is better to use the

improved PDS than DP in PDS for the cepstrum coefficients with variance weighting. The improved DPPDS is superior to the other algorithms.

cost ratio	DP in PDS	improved PDS	improved DPPDS
0.1	0 %	0.82 %	0.95%
0.2	0.5 %	2.2 %	4.6%
0.3	2.9 %	4.0 %	5.5%
0.4	5.7 %	5.8 %	8.2%
0.5	7.9 %	7.9 %	10.9%
0.6	10.1 %	10.0 %	13.3%
0.7	12.4 %	12.1 %	15.7%
0.8	14.5 %	14.2 %	17.8%
0.9	16.3 %	16.1 %	19.7%
1.0	17.3 %	18.1 %	21.7%
1.1	19.1 %	19.8 %	23.5%
1.2	20.8 %	21.5 %	25.2%
1.3	24.5 %	23.1 %	26.7%
1.4	26.2 %	24.6 %	27.8%
1.5	27.7 %	26.0 %	29.2%
1.6	29.1 %	27.3 %	31.0%
1.7	30.4 %	28.6 %	32.3%
1.8	31.6 %	29.7 %	33.6%
1.9	32.7 %	30.9 %	34.8%
2.0	33.8 %	31.7 %	35.9%
2.5	38.1 %	36.8%	40.7%
3.0	42.6 %	40.5 %	44.5%
3.5	45.9 %	44.5%	47.5%
4.0	48.5 %	47.9 %	49.9%
4.5	50.6 %	50.7%	52.0%

Table 3.12: The performance of DP in PDS, improved PDS and improved DPPDS (percentage improvement on standard PDS)

3.4 Improvement in Extended Partial Distortion Search

The extended partial distortion search (EPDS) algorithm (Chen & Pan, 1989; Pan, 1988) is a modified version of PDS which optimizes the calculation in terms of the number of multiplications for a minor overhead in data sorting. It can be used in vector encoding and word recognition. The EPDS algorithm for the frame-distortion accumulation in word recognition is stated as follows:

Step 1: Let $L_j = 1$ and calculate the distortion Df_j between the first feature frame X_1 and the jth codebook, for j = 1 to V. Here V is the number of codebooks and L_j is the L_jth frame for the jth codebook.

Step 2: Find $Df_s = Min_j Df_j$ and $s = argMin_j Df_j$.

Step 3: If $L_s = T$, then set the sth codebook to be the best match and terminate the program; otherwise, set $L_s = L_s + 1$, calculate the encoding distortion of X_{l_s} frame by using the sth codebook and add it to Df_s, and go to step 2. Here T is the number of frames.

Assume a recognition system including 10 words (10 codebooks) and one test word including 9 frames. As shown in Table 3.13, Cb_i is the ith codebook (word) and f_i is the ith frame of the test word. The value of each entry is the accumulated frame-distortions for the codebook. The underline used in this example means the last calculated frame-distortion for the corresponding codebook. The calculation of many frame-distortions can be omitted. Hence the EPDS algorithm is very suitable for word recognition. For vector encoding, this computes the dimension-

	Cb_1	Cb_2	Cb_3	Cb_4	Cb_5	Cb_6	Cb_7	Cb_8	Cb_9	Cb_{10}
f_1	8	7	9	6	5	11	2	2	4	5
f_2	15	15	14	12	18	17	3	6	8	10
f_3	22	<u>24</u>	<u>25</u>	17	<u>27</u>	22	6	11	12	11
f_4	<u>25</u>	35	29	22	32	<u>28</u>	9	17	18	21
f_5	26	36	32	<u>28</u>	35	32	13	19	<u>26</u>	<u>23</u>
f_6	30	44	34	37	39	38	17	<u>25</u>	28	31
f_7	36	47	38	39	42	40	19	26	31	33
f_8	38	53	41	43	47	44	20	27	33	35
f_9	39	58	43	46	51	47	<u>23</u>	29	36	38

Table 3.13: Diagram of distortion calculation for EPDS in word recognition

distortion for the first dimension of the input vector to the first dimension of all codewords, then sorts the dimension-distortion to obtain the nearest codeword. The distortion for the input vector to the nearest codeword in the second dimension is calculated and added to the previous distortion of the same codeword. The dimension-distortions are sorted again to obtain the nearest codeword. The procedure continues until the last dimension-distortion is calculated and the distortion is smallest.

Assume the number of dimensions and the number of codewords are 10 and 8, respectively.

Table 3.14 illustrates an example of EPDS algorithm in vector encoding. The value of each entry is the accumulated dimension-distortions for the codeword. The underline used in this example means the last calculated dimension-distortion for the corresponding codeword. The calculation of many dimension-distortions can be omitted. The EPDS algorithm is an optimal PDS algorithm in the sense of reducing the number of multiplications. The detailed algorithm of the EPDS in vector encoding is described as follows:

Step 1: Let $l_i = 1$ and calculate the distortion $D_i = (x_1 - c_{i1})^2$ between the first dimension x_1 of the input vector X and the first dimension c_{i1} of the ith codeword C_i, for $i = 1$ to N. Here N is the number of codewords and l_i is the l_ith dimension for the ith codeword.

Step 2: Find $D_s = Min_i D_i$ and $s = arg Min_i D_i$.

Step 3: If $l_s = k$, then set the sth codeword to be the best match and terminate the program; otherwise, set $l_s = l_s + 1$, calculate the encoding distortion of x_{l_s} by using the sth codeword and add it to D_s, and go to step 2. Here k is the dimension of the input vector and codewords.

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
x_1	2	3	2	1	5	4	8	7
x_2	6	5	4	2	8	7	12	12
x_3	7	9	11	4	12	9	<u>15</u>	<u>16</u>
x_4	9	11	13	5	<u>16</u>	13	18	19
x_5	<u>14</u>	13	<u>18</u>	6	18	<u>14</u>	21	22
x_6	17	<u>15</u>	21	8	20	16	25	23
x_7	18	16	22	10	21	17	27	24
x_8	20	17	25	11	23	18	28	27
x_9	22	18	28	12	24	20	30	29
x_{10}	23	19	30	<u>14</u>	26	21	35	32

Table 3.14: Diagram of distortion calculation for EPDS in vector encoding

The EPDS algorithm is suitable for computer architectures in which the complexity of comparisons is negligible with respect to that of the multiplications, such as Intel 80486 processor. However, EPDS is less suited to some DSP processors, such as the TMS320 series of processors in which comparisons are computationally expensive. An improvement of the extended partial distortion search approach is proposed here. It involves inserting the sorting operation from

a suitable dimension to minimize the EPDS search cost for any computer architecture. Here sorting means that the comparisons are performed to find the codeword which has the minimum distortion at the present stage.

In this improved algorithm, the sorting of the accumulated distortions to find the minimum D_s is performed only after the first j dimensions' distortion terms have been accumulated for every codeword, where j is chosen to minimize the total computation. Let r be the cost ratio of the sorting time to dimension-distortion computation time. To insert the sorting to dimension-distortion accumulation at the jth dimension, the cost of sorting is $m_j r$, but there is a decrease of $N - m_j$ dimension-distortion computations. Here N is the number of codewords and m_j is the average number of codewords whose distortion computation cannot be omitted at the sorting insertion of the jth dimension. From the above description, the following two equations are satisfied.

$$m_j \geq 1, \quad j = 1, ..., k-1 \tag{3.53}$$

$$m_j \geq m_{j+1}, \quad j = 1, ..., k-2 \tag{3.54}$$

Let $A(j)$ be the global advantage function of inserting the sorting from the jth dimension onwards. The advantage can be expressed in terms of N, k, m_j and r as follows.

$$A(k) = 0 \tag{3.55}$$

$$A(j) = A(j+1) + V(j), \quad j = 1, ..., k-1 \tag{3.56}$$

where $V(j)$ is the local advantage due to sorting at the jth dimension, given by

$$V(j) = (N - m_j) - m_j r = N - (r+1)m_j \tag{3.57}$$

From Eq. 3.54 and 3.57,

$$V(i) \geq V(j) \quad \text{if} \quad i > j \tag{3.58}$$

Hence there is some t ($1 \leq t \leq k$) such that

$$V(j) \geq 0 \quad \text{for} \quad j = t, \ldots, k-1 \tag{3.59}$$

$$V(j) \leq 0 \quad \text{for} \quad j = 1, \ldots, t-1 \tag{3.60}$$

and so

$$A(t) \geq A(j), \quad j = 1, \ldots, k, \quad t \neq j \tag{3.61}$$

This value t is the optimal sorting insertion dimension for the given value of the cost ratio r.

From Eq. 3.57, 3.59 and 3.60, the cost interval r_t corresponding to the sorting insertion dimension t can be derived.

$$0 \leq r_t \leq \frac{N}{m_t} - 1, \quad t = 1 \tag{3.62}$$

$$\frac{N}{m_{t-1}} - 1 \leq r_t \leq \frac{N}{m_t} - 1, \quad t = 2, \ldots, k-1 \tag{3.63}$$

From the training data, calculate the cost interval r_j, $j = 1$ to $k-1$. The optimal sorting insertion is from the jth dimension if the cost ratio of the computer architecture lies in the cost interval r_j. For the conventional exhaustive full search method, Nk dimension-distortions are computed corresponding to the computation time of Nk multiplications, $N(2k-1)$ additions, and $(N-1)$ comparisons. One dimension-distortion computation involves approximately the computation time of one multiplication and two additions. The sorting time is $N-1$ comparisons for the basic sorting method. The computation time of EPDS and improved EPDS are $Nk - A(1)$ and $Nk - A(t)$. The performance of EPDS, improved EPDS, and the improvements of the improved EPDS are as follows.

$$\text{EPDS performance} = \frac{Nk - A(1)}{Nk} \tag{3.64}$$

$$\text{Improved EPDS performance} = \frac{Nk - A(t)}{Nk} \tag{3.65}$$

$$\text{Improvement} = \frac{A(t) - A(1)}{Nk - A(1)} \tag{3.66}$$

The speech databases used in training and test experiments consist of one hundred words recorded from five male speakers separated into three sets. The sampling rate used is 16 kHz and 12-dimensional cepstrum coefficients are computed over 20 ms-wide frames with a 5 ms frame shift. The first data set recorded from two speakers is used to generate the codebook. Cost intervals for the improved EPDS algorithm are computed from the codebook using the second data set recorded from two other speakers. The third data set recorded from the fifth speaker is used to test the performance of these approaches.

Table 3.15 illustrates cost intervals of 16 codewords and 128 codewords. From these cost intervals and the cost ratio of sorting time to dimension-distortion computation time for a given computer architecture, the dimension of sorting insertion can be decided. For example, the inserting should be from the third dimension if the cost ratio of the computer architecture is 6 for 128 codewords. The performance comparison of improved EPDS and EPDS is shown in Table 3.16 and Table 3.17. These efficiencies can be calculated from Eq. 3.64, 3.65 and 3.64 by using the maximum cost ratio from Table 3.15. For 128 codewords, if the cost ratio is 7.55, the performance of EPDS is 67%, but that of improved EPDS will be 50% for inserting from the third dimension, it improves 26%. This technique can also be applied to frame-distortion accumulation in a word recognition system.

inserting dimension	cost intervals of 16 codewords	cost intervals of 128 128 codewords
1	[0.00 , 1.56]	[0.00 , 2.03]
2	[1.56 , 3.29]	[2.03 , 4.99]
3	[3.29 , 4.32]	[4.99 , 7.55]
4	[4.32 , 6.12]	[7.55 , 13.4]
5	[6.12 , 7.43]	[13.4 , 19.5]
6	[7.43 , 8.92]	[19.5 , 29.3]
7	[8.92 , 10.1]	[29.3 , 39.0]
8	[10.1 , 11.6]	[39.0 , 52.9]
9	[11.6 , 12.6]	[52.9 , 67.7]
10	[12.6 , 13.4]	[67.7 , 84.2]
11	[13.4 , 14.3]	[84.2 , 104]

Table 3.15: Cost intervals of 16 codewords and 128 codewords

inserting dimension	EPDS performance	improved EPDS performance	improvement
1	41 %	41 %	0 %
2	63 %	58 %	8 %
3	76 %	66 %	13 %
4	99 %	77 %	22 %
5	116 %	84 %	28 %
6	135 %	90 %	34 %
7	150 %	93 %	38 %
8	170 %	97 %	43 %
9	182 %	98 %	46 %
10	192 %	99 %	48 %
11	203 %	100 %	51 %

Table 3.16: The performance of 16 codewords

inserting dimension	EPDS performance	improved EPDS performance	improvement
1	29 %	29 %	0 %
2	49 %	42 %	15 %
3	67 %	50 %	26 %
4	107 %	60 %	44 %
5	148 %	68 %	54 %
6	215 %	78 %	64 %
7	281 %	84 %	70 %
8	376 %	90 %	76 %
9	478 %	95 %	80 %
10	590 %	98 %	83 %
11	724 %	100 %	86 %

Table 3.17: The performance of 128 codewords

3.5 Fast Algorithm for Approximate Search

Multiplication operations are far more expensive compared with comparison and addition operations for general processors (Leibson, 1993). In this section, an efficient approximate search algorithm which can dramatically reduce the number of multiplication operations is presented. This algorithm is based on the modification of the Chebyshev metric or Manhattan metric. Assume the training data and codewords are $X_p = \{x_p^1, x_p^2, ..., x_p^k\}$ and $C_i = \{c_i^1, c_i^2, ..., c_i^k\}$, respectively, $p = 1, 2, ..., T$, $i = 1, 2, ..., N$. T, k and N are the total number of training data

vectors, the number of dimensions and the number of codewords, respectively. The distortion between data vector X_m and codeword C_l can be expressed as follows:

$$d(l, p) = \sum_{j=1}^{k} (x_p^j - c_l^j)^2.$$

The codeword with the minimum value of the maximum dimension-distortion is

$$n_p = \mathrm{argmin}_i \mathrm{max}_j |x_p^j - c_i^j|,$$

and

$$d(n_p, p) = \sum_{j=1}^{k} (x_p^j - c_n^j)^2.$$

Separate all codewords C_i into two sets for every training data vectors X_p.

$$\mathrm{First\ \ set:} \qquad A_p = \{i | d(i, p) \geq d(n_p, p)\}.$$

$$\mathrm{Second\ \ set:} \qquad B_p = \{i | d(i, p) < d(n_p, p)\}.$$

Calculate the parameter $rate$ using Eq. 3.67 and 3.68

$$rate_p = \frac{\mathrm{max}_{l \in B_p} \mathrm{max}_j |c_l^j - x_p^j|}{\mathrm{max}_j |c_{n_p}^j - x_p^j|}. \tag{3.67}$$

for each training data vector X_p.

$$rate = \mathrm{max}_p rate_p + \delta. \tag{3.68}$$

where δ is a small value. After the parameter $rate$ is obtained, a new codeword elimination criterion is developed as follows:

$$\mathrm{if} \qquad \mathrm{max}_j |x_m^j - c_l^j| \geq rate . \mathrm{max}_j |x_m^j - c_{n_p}^j|, \tag{3.69}$$

$$\mathrm{then} \qquad \sum_{j=1}^{k} (x_m^j - c_l^j)^2 \geq \sum_{j=1}^{k} (x_m^j - c_{n_p}^j)^2, \tag{3.70}$$

where $n_p = \mathrm{argmin}_i \mathrm{max}_j |x_m^j - c_i^j|$.

method	mul.	cmp.	add.	sum	average distortion
minimax	502,231	3,376,691	3,587,363	7,466,285	0.940086
rate_1.0	0	3,099,888	3,129,984	6,229,872	0.959554
rate_1.1	97,216	3,332,328	3,189,985	6,619,529	0.945844
rate_1.2	142,396	3,345,996	3,230,290	6,718,682	0.941630
rate_1.3	178,404	3,355,566	3,262,337	6,796,307	0.940600
rate_1.4	209,539	3,362,509	3,289,937	6,861,985	0.940210
rate_1.5	237,772	3,367,516	3,314,977	6,920,265	0.940139
rate_1.6	262,278	3,371,061	3,336,813	6,970,152	0.940096
rate_1.7	282,651	3,373,216	3,355,062	7,010,929	0.940093
rate_1.8	301,284	3,374,809	3,371,852	7,047,945	0.940088
rate_1.9	316,990	3,375,730	3,386,098	7,078,818	0.940086
rate_2.0	330,482	3,376,189	3,398,439	7,105,110	0.940086
rate_2.1	341,306	3,376,384	3,408,397	7,126,087	0.940086
rate_2.2	352,263	3,376,497	3,418,520	7,147,280	0.940086
rate_2.3	361,709	3,376,520	3,427,279	7,165,508	0.940086
rate_2.4	370,395	3,376,539	3,435,332	7,182,266	0.940086
rate_2.5	378,380	3,376,542	3,442,745	7,197,667	0.940086

Table 3.18: Performance comparison of minimax method and fast approximate algorithm for 8 codewords

The efficiency of the codeword search depends on the value of the parameter $rate$. The smaller value the $rate$ is, the more efficient this algorithm gets. In the extreme, $rate = 1$, it is Chebyshev metric or Manhattan metric. For this metric, the number of multiplications, comparisons and additions are 0, $N.(k-1) + (N-1)$ and $N.k$, respectively. Parameter $rate$ can be reduced to a smaller value if the increased distortion is small. The test materials for these experiments consisted of two hundred words recorded from two male speakers. The speech was sampled at a rate of 16 kHz and 13-dimensional cepstrum coefficients were computed over 20 ms-wide frames with a 5 ms frame shift. A total of 20,030 analyzed frames used as the training data were recorded from one male speaker. The test data includes 30,096 analyzed frames recorded from the other speaker. Codebooks of size 8, 256 and 1,024 codewords with Euclidean distortion measure are used in these experiments.

Tables 3.18, 3.19 and 3.20 illustrate the performance of the minimax method and this new efficient approximate search algorithm with different $rate$. The training $rate$s are 1.874833, 2.028345 and 2.231797 for 8, 256 and 1024 codewords, respectively. Here δ is set to 0. The average distortion is the same as the minimax method if the training $rate$s are used. Obviously,

method	mul.	cmp.	add.	sum	average distortion
minimax	2,710,965	109,599,202	102,346,066	214,656,233	0.346460
rate_1.0	0	100,129,392	100,159,488	200,288,880	0.388426
rate_1.1	428,683	107,992,358	10,0517,026	208,938,067	0.364018
rate_1.2	770,084	108,202,247	10,0818,487	209,790,818	0.353663
rate_1.3	1,087,047	108,424,879	101,091,793	210,603,719	0.349195
rate_1.4	1,388,239	108,647,096	101,344,105	211,379,440	0.347424
rate_1.5	1,675,786	108,862,034	101,576,736	212,114,556	0.346771
rate_1.6	1,932,988	109,051,837	101,777,160	212,761,985	0.346557
rate_1.7	2,157,441	109,213,439	101,945,392	213,316,272	0.346482
rate_1.8	2,337,348	109,338,814	102,075,174	213,751,336	0.346468
rate_1.9	2,471,616	109,429,695	102,168,902	214,070,213	0.346461
rate_2.0	2,563,560	109,490,166	102,231,336	214,285,062	0.346460
rate_2.1	2,619,365	109,525,849	102,268,176	214,413,390	0.346460
rate_2.2	2,650,193	109,544,662	102,288,120	214,482,975	0.346460
rate_2.3	2,665,490	109,553,760	102,297,907	214,517,157	0.346460
rate_2.4	2,672,197	109,557,447	102,302,218	214,531,862	0.346460
rate_2.5	2,674,831	109,558,755	102,303,955	214,537,541	0.346460

Table 3.19: Performance comparison of minimax method and fast approximate algorithm for 256 codewords

the parameter $rate$ can be set to a small value if the codebook size is small. For 8 codewords, the number of multiplications will be reduced by 80% with only 0.6% increased distortion if the $rate$ is 1.1.

3.6 Efficient Search Algorithm for Image Coding

The mean-distance-ordered search (MPS) algorithm (Ra & Kim, 1993) takes advantage of the fact that the nearest codeword is usually in the neighbourhood of the minimum squared mean distance. The basic inequality of this approach is as follows:

$$\text{if} \qquad |\sum_{i=1}^{k} x^i - \sum_{i=1}^{k} c_j^i| \geq \sqrt{kD_{min}}, \tag{3.71}$$

$$\text{then} \qquad \sum_{i=1}^{k} (x^i - c_j^i)^2 \geq D_{min}.$$

This means C_j will not be the nearest neighbour to X if Eq. 3.71 is satisfied. In the MPS algorithm, the sum of all dimensions for each codeword is calculated first and these values are sorted

method	mul.	cmp.	add.	sum	average distortion
minimax	6,457,314	436,016,497	405,649,633	848,123,444	0.271357
rate_1.0	0	400,607,856	400,637,952	801,245,808	0.310621
rate_1.1	577,887	431,680,772	401,129,451	833,388,110	0.288017
rate_1.2	1,110,517	432,055,772	401,599,403	834,765,692	0.278229
rate_1.3	1,684,421	432,507,813	402,095,215	836,287,449	0.274065
rate_1.4	2,307,142	433,010,905	402,619,302	837,937,349	0.272393
rate_1.5	2,960,568	433,533,163	403,152,623	839,646,354	0.271760
rate_1.6	3,620,279	434,047,396	403,672,576	841,340,251	0.271497
rate_1.7	4,255,817	434,526,434	404,154,916	842,937,167	0.271393
rate_1.8	4,827,987	434,942,906	404,573,078	844,343,971	0.271366
rate_1.9	5,301,655	435,275,810	404,907,048	845,484,513	0.271360
rate_2.0	5,654,318	435,515,429	405,147,304	846,317,051	0.271360
rate_2.1	5,890,136	435,671,173	405,303,503	846,864,812	0.271360
rate_2.2	6,031,703	435,762,281	405,394,910	847,188,894	0.271360
rate_2.3	6,106,298	435,808,957	405,441,729	847,356,984	0.271360
rate_2.4	6,139,254	435,829,116	405,461,953	847,430,323	0.271360
rate_2.5	6,152,092	435,836,743	405,469,710	847,458,545	0.271360

Table 3.20: Performance comparison of minimax method and fast approximate algorithm for 1024 codewords

in the increasing or decreasing order. In the encoding stage, the sum of all dimensions of the data vector is computed and one codeword called the tentative matching codeword required for the minimization of the left hand side of Eq. 3.71, mean distortion (MD), is found. The squared Euclidean distortion between the data vector and this tentative matching codeword referred to here as D_{min} is calculated. Then Eq. 3.71 is applied to eliminate impossible codeword matching. Codewords C_j for which $\sum_{i=1}^{k} c_j^i \geq \sum_{i=1}^{k} x^i + \sqrt{kD_{min}}$ or $\sum_{i=1}^{k} c_j^i \leq \sum_{i=1}^{k} x^i - \sqrt{kD_{min}}$ can be eliminated. Otherwise, the PDS is applied to calculate the distortion and update D_{min}.

The efficiency of the MPS algorithm depends on the distortion of the tentative matching codeword. If the distortion of the tentative matching codeword is small, then the MPS algorithm is very efficient. Unfortunately, some data vectors may have small mean distortion (MD) but the squared Euclidean distortion is significant, such as one data vector (200, 200, 0, 0) and one codeword (0, 0, 200, 200). In order to improve performance, a new algorithm is proposed from the extension of the bound for Minkowski metric (Pan *et al.*, 1996b). This bound is as follows:

$$\text{if} \qquad \sum_{i=1}^{s} |x^i - c_j^i|^p \geq \sqrt[\frac{n}{p}]{h^{\frac{n}{p}-1} D_{min}} \tag{3.72}$$

$$\text{then} \qquad \sum_{i=1}^{k} |x^i - c_j^i|^n \geq D_{min} \tag{3.73}$$

where $s \leq h \leq k$ and $p \leq n$.

This is an improved absolute error inequality (IAEI) criterion (Pan *et al.*, 1996b) by setting $n = 2$ and $p = 1$. Hence the IAEI criterion is expressed as follows:

$$\text{if} \qquad \sum_{i=1}^{s} |x^i - c_j^i| \geq \sqrt{hD_{min}}, \tag{3.74}$$

$$\text{then} \qquad \sum_{i=1}^{k} (x^i - c_j^i)^2 \geq D_{min}, \tag{3.75}$$

where $s \leq h \leq k$.

$$\text{Because} \qquad \sum_{i=1}^{s} |x^i - c_j^i| \geq |\sum_{i=1}^{s} x^i - \sum_{i=1}^{s} c_j^i|, \tag{3.76}$$

hence a new inequality is derived as follows:

$$\text{if} \qquad |\sum_{i=1}^{s} x^i - \sum_{i=1}^{s} c_j^i| \geq \sqrt{hD_{min}}, \tag{3.77}$$

$$\text{then} \qquad \sum_{i=1}^{k} (x^i - c_j^i)^2 \geq D_{min},$$

where $s \leq h \leq k$.

This new inequality (Eq. 3.77) is the generalized form of the the basic inequality (Eq. 3.71) of the MPS algorithm. By using this new inequality, the codeword can be separated into two vectors. The first vector is composed of the first half of the elements, the other elements belong to the second vector. Using these two separated vectors, the sum of the elements for these separated codewords can be calculated first. Therefore Eq. 3.77 can be applied to eliminate impossible codeword matching for these two separated vectors. Because the sum of the first part is considered as well as the sum of the second part, this approach overcomes the inefficiency

of the unsuitable tentative matching codeword using mean distortion (MD). By combining the MPS algorithm with Eq. 3.77 for the separated codewords, the improved algorithm is computed as follows:

Step 1: $FCode\_sum_i = \sum_{j=1}^{k/2} c_i^j$, $SCode\_sum_i = \sum_{j=\frac{k}{2}+1}^{k} c_i^j$ and $TCode\_sum_i = FCode\_sum_i + SCode\_sum_i$ are calculated for each codeword, $i = 1, 2, ..., N$, N is the number of codewords. A sorting list is computed according to the increasing order of the $TCode\_sum_i$.

Step 2: $FData\_sum = \sum_{j=1}^{k/2} x^j$, $SData\_sum = \sum_{j=\frac{k}{2}+1}^{k} x^j$ and $TData\_sum = FData\_sum + SData\_sum$ are calculated.

Step 3: Calculate the tentative matching codeword i using $argMin_i |TData\_sum - TCode\_sum_i|$.

Step 4: Calculate the squared Euclidean distortion D_{min} for the tentative matching codeword. Set l to be the nearest uncalculated codeword to the tentative matching codeword in the sorting list.

Step 5: Check the termination of this program. Test Eq. 3.71 for the neighbour codewords in a back-and-forth manner as in paper (Ra & Kim, 1993), if it is satisfied, delete impossible codeword matching, set l to be the nearest uncalculated codeword to the tentative matching codeword in the sorting list and goto step 5; Otherwise, goto next step.

Step 6: If $|FData\_sum - FCode\_sum_l| \geq \sqrt{\frac{k}{2} D_{min}}$ or $|SData\_sum - SCode\_sum_l| \geq \sqrt{\frac{k}{2} D_{min}}$, then eliminate this codeword; otherwise use the PDS to the codeword search and update the D_{min}. Set l to be the nearest uncalculated codeword to the tentative matching codeword in the sorting list and goto step 5.

The training material for these experiments was a LENA image. It consists of 512 x 512 pels with 8 bits/pel resolution. Codebook sizes of 64, 128, 256, 512 and 1024 are generated by the well known LBG algorithm. The vector dimension k is 16. An AIRPLANE image was used as the test material. Experiments were carried out to test the performance of the MPS algorithm and the proposed new algorithm. The performance is measured in terms of the number of calculated distortions. As shown in Table 3.21, Table 3.22, Table 3.23, 3.24 and 3.25, the new algorithm reduces 29%, 34%, 38%, 42% and 44% calculated distortions compared with the MPS algorithm for 64, 128, 256, 512 and 1024 codewords, respectively. In terms of the number

of multiplications, this new algorithm reduces 27.57 % compared with the MPS algorithm for 1024 codewords. In terms of the total number of operations, 10.81%, 13.31% and 14.94% operations are reduced for 256, 512 and 1024 codewords, respectively. Actually, this algorithm can be further improved by using IAEI (Pan *et al.*, 1996b) instead of PDS. Note that $(k + 3)N$ memory is needed for this improved algorithm compared with $(k + 1)N$ memory for the MPS algorithm. From the experiments, the performance of the proposed algorithm is significantly better than the MPS algorithm. This improved algorithm can be extended by separating the codevector into several sub-vectors.

method	mul.	add.	cmp.	sum	distortion no.
MPS	938,360	2,068,161	820,387	3,826,908	69,088
New	841,071	1,918,504	826,195	3,585,770	49,116

Table 3.21: Performance comparison of MPS and New algorithm for 64 codewords, MSE=168

method	mul.	add.	cmp.	sum	distortion no.
MPS	1,417,101	3,006,633	1,372,334	5,796,068	134,886
New	1,203,841	2,721,519	1,373,637	5,298,997	89,061

Table 3.22: Performance comparison of MPS and New algorithm for 128 codewords, MSE=138

method	mul.	add.	cmp.	sum	distortion no.
MPS	2,236,464	4,626,455	2,323,753	9,186,672	259,460
New	1,798,428	4,082,844	2,312,421	8,193,693	161,573

Table 3.23: Performance comparison of MPS and New algorithm for 256 codewords, MSE=115

3.7 Fast Search Algorithm for Quadratic Metric

In chapter 2, subsection 2.9.1, the bound for quadratic metric (Pan *et al.*, 1996b) is derived. Assume that

$$D_{min} = D(X, C_m) = (X - C_m)^t W^{-1} (X - C_m) = \sum_{i=1}^{k} |E_m^t V_i|^2. \qquad (3.78)$$

$$\text{If} \qquad \sum_{i=1}^{s} |E_j^t V_i| \geq \sqrt{hD_{min}}, \tag{3.79}$$

$$\text{then} \qquad D(X, C_j) \geq D_{min} \tag{3.80}$$

where $E_m = X - C_m$, $W^{-1} = LL^t$, $L = [V_1 V_2 V_3 ... V_k]$, and $s \leq h \leq k$.

For speech or image data, the classification result of the present vector is usually the same as or close to the classified result of the previous vector. The nearest codeword of the previous vector can be used as the tentative match called previous vector candidate (Pan, 1988; Pan *et al.*, 1996c; Chen & Pan, 1989). A fast search algorithm for the quadratic metric is proposed by using the *previous vector candidate* as the tentative match, then the bound for quadratic metric is applied to eliminate impossible codeword match. This fast search algorithm is depicted as follows:

Step 1: Compute the nearest neighbour for the first frame X_1. For the other frame X_p, use the nearest neighbour of X_{p-1} (previous vector candidate) as a tentative match and so find the initial value of D_{min}.

Step 2: For every codeword C_j, calculate steps 3 to 7.

Step 3: For every dimension (i from 1 to k), calculate steps 4 to 6.

Step 4: Calculate the error vector component $e_{ij} = (x^i - c_j^i)$ and $|E_j^t V_i| = \sum_{r=1}^{i} e_{rj} l_{ir}$.

Step 5: If $\sum_{m=1}^{i} |E_j^t V_m| \geq \sqrt{hD_{min}}$, $h \geq i$, then C_j will not be the nearest neighbour to the frame X_p, therefore go to step 3 for the next codeword.

Step 6: Calculate $|E_j^t V_i|^2$. If $\sum_{m=1}^{i} |E_j^t V_m|^2 \geq D_{min}$, then C_j will not be the nearest neighbour to the frame X_p, therefore go to step 3 for the next codeword.

Step 7: If $\sum_{m=1}^{k} |E_j^t V_m|^2 < D_{min}$, set $D_{min} = \sum_{m=1}^{k} |E_j^t V_m|^2$ and record C_j as the nearest neighbour to X_p.

The test materials for these experiments consisted of 99 words recorded from one male speaker. The speech was sampled at a rate of 16 kHz and 13-dimensional cepstrum coefficients were computed over 20 ms-wide frames with a 5 ms frame shift. The total number of frames is

9,391. Codebooks with 256, 512 and 1,024 codewords with quadratic metric are used in these experiments.

The conventional exhaustive method, the fast codeword search algorithm without tentative match approach (i.e. with C_1 as the tentative candidate) and the fast codeword search algorithm with quadratic metric were tested in these experiments. The conventional exhaustive method is referred to as "$conventional$". The fast codeword search algorithm without tentative match approach and the fast codeword search algorithm are referred to as "$No-quadratic$" and "$Pre-quadratic$", respectively. The bounds for quadratic metric are separated into four sections ($h = 1, 4, 9, 13$).

The experimental results are shown in Tables 3.26, 3.27 and 3.28. For 1,024 codewords, 91.6% of the number of multiplications are saved, as well as considerable saving in the number of additions. The increase in the number of comparisons is moderate.

A modified method can be applied to previous fast algorithm by preprocessing $C_m^t L$ first, then XL can be operated outside the loop of the codeword search. This modified method is more efficient than previous one. Assume z_{mi} is the element of the vector $C_m^t L$, $1 \leq m \leq N$, $1 \leq i \leq k$. The modified algorithm is described as follows:

Step 1: Compute the nearest neighbour for the first frame X_1. For the other frame X_p, use the nearest neighbour of X_{p-1} (previous vector candidate) as a tentative match and so find the initial value of D_{min}.

Step 2: Calculate $X_p{}^t L = (y_1, y_2, ..., y_k)$.

Step 3: For every codeword C_j, calculate steps 3 to 7.

Step 4: For every dimension (i from 1 to k), calculate steps 4 to 6.

Step 5: Calculate $|E_j^t V_i| = \sum_{r=1}^{i} |y_r - z_{jr}|$.

Step 6: If $\sum_{m=1}^{i} |E_j^t V_m| \geq \sqrt{hD_{min}}$, $h \geq i$, then C_j will not be the nearest neighbour to the frame X_p, therefore go to step 3 for the next codeword.

Step 7: Calculate $|E_j^t V_i|^2$. If $\sum_{m=1}^{i} |E_j^t V_m|^2 \geq D_{min}$, then C_j will not be the nearest neighbour to the frame X_p, therefore go to step 3 for the next codeword.

Step 8: If $\sum_{m=1}^{k} |E_j^t V_m|^2 < D_{min}$, set $D_{min} = \sum_{m=1}^{k} |E_j^t V_m|^2$ and record C_j as the nearest neighbour to X_p.

The same materials are used to test this modified method. Experimental results is shown in Table 3.29. In terms of the total number of mathematic operations, the modified version can reduce by more than 50 % computation complexity. No extra memory is needed if the same matrix *W* is used throughout. Hence the original codewords need not be stored, but can be replaced completely by the transformed codewords $C_m^t L$.

method	mul.	add.	cmp.	sum	distortion no.
MPS	3,500,725	7,136,625	3,823,283	14,460,633	487,045
New	2,644,173	6,118,165	3,773,868	12,536,206	281,031

Table 3.24: Performance comparison of MPS and New algorithm for 512 codewords, MSE=92

method	mul.	add.	cmp.	sum	distortion no.
MPS	6,352,245	12,818,199	7,156,645	26,327,098	962,662
New	4,600,745	10,770,471	7,022,970	22,394,186	544,073

Table 3.25: Performance comparison of MPS and New algorithm for 1024 codewords, MSE=85

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
$Conventional$	437,545	2,395	435,141	875,081
$No-quadratic$	82,691	26,354	92,776	201,821
$Pre-quadratic$	50,685	18,630	57,003	126,318

Table 3.26: Computational complexity of codeword search for 256 codewords on quadratic metric

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
$Conventional$	875,091	4,799	870,283	1,750,173
$No-quadratic$	142,364	47,619	160,012	349,995
$Pre-quadratic$	86,963	33,569	97,912	218,444

Table 3.27: Computational complexity of codeword search for 512 codewords on quadratic metric

method	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
$Conventional$	1,750,182	9,607	1,740,566	3,500,355
$No-quadratic$	246,729	86,226	277,687	610,642
$Pre-quadratic$	147,768	59,737	166,366	373,871

Table 3.28: Computational complexity of codeword search for 1024 codeword on quadratic metric

number of codewords	mul.($\times 10^3$)	cmp.($\times 10^3$)	add.($\times 10^3$)	sum($\times 10^3$)
256	10,926	18,630	27,957	57,513
512	18,837	33,569	49,433	101,839
1,024	32,555	59,737	86,882	179,174

Table 3.29: Computational complexity of modified method

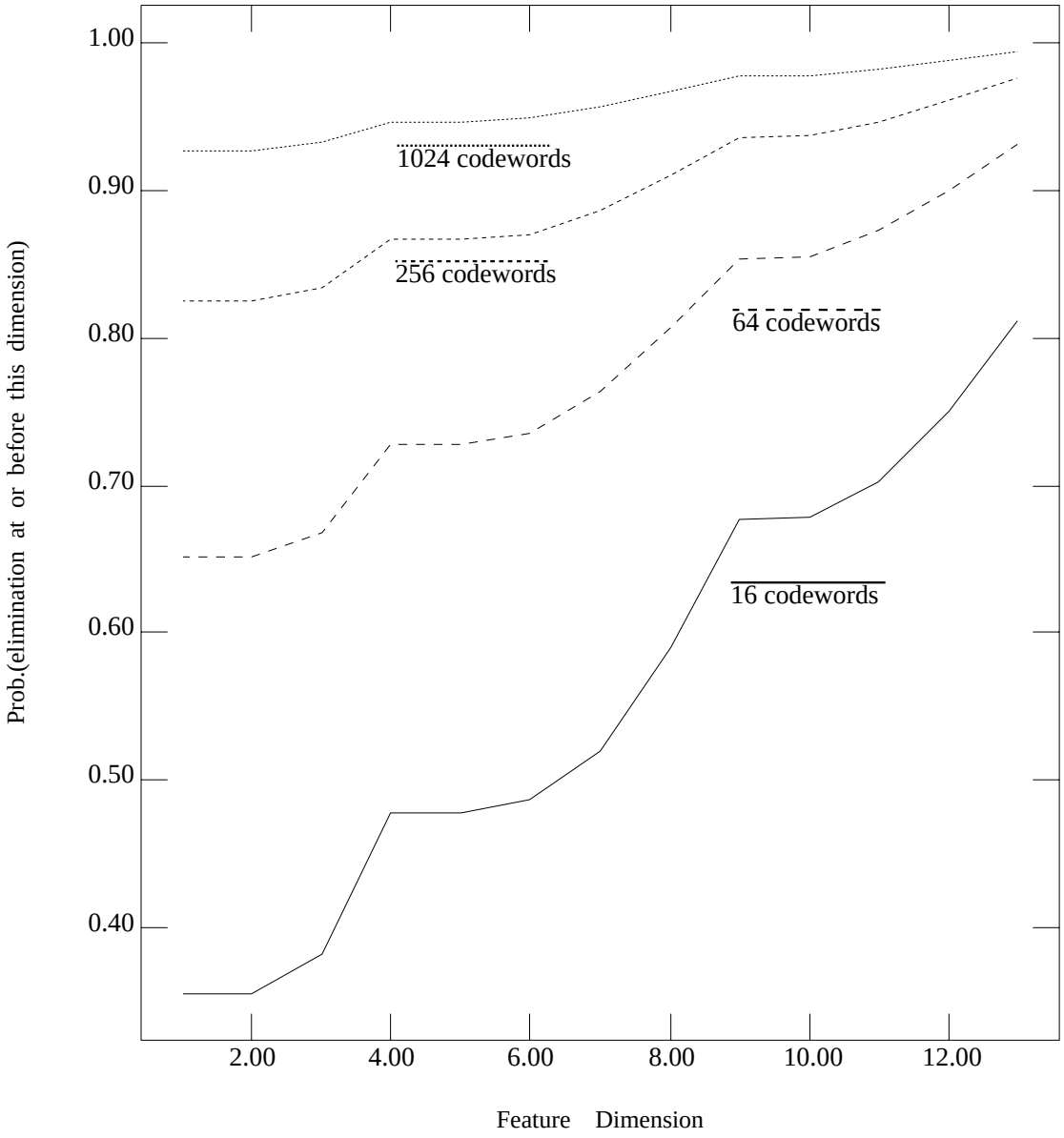

Figure 3.5: Experimental results for the elimination probability of IAEI at each feature dimension (h is set to 1, 4, 9 and 13)

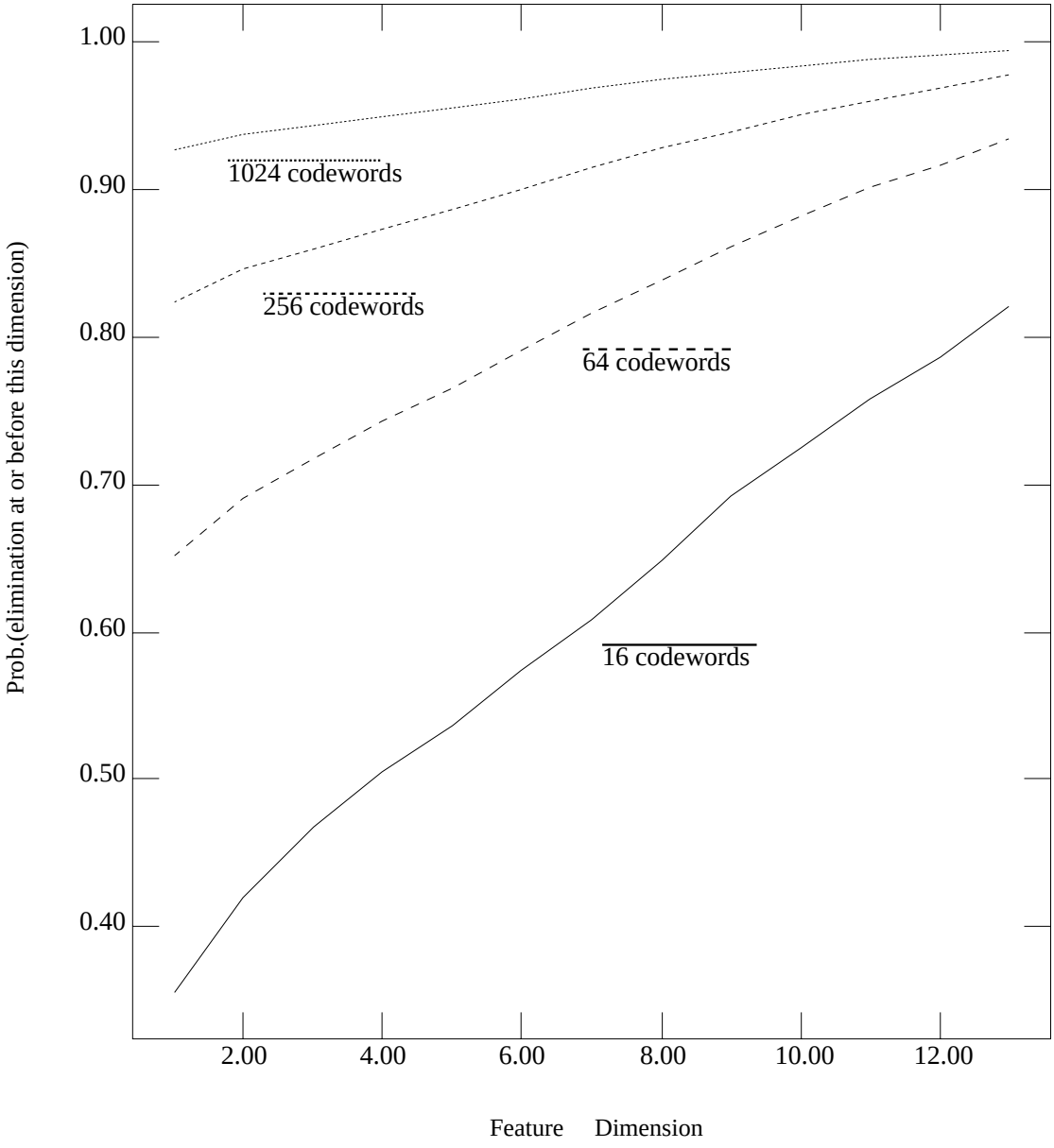

Figure 3.6: Experimental results for the elimination probability of IAEI at each feature dimension (h is set to i for the ith dimension)

Chapter 4

Fast Clustering Algorithms

4.1 Introduction

Vector quantization (VQ) is a source coding procedure that can achieve improved data compression ratios compared to linear approaches combined with a scalar quantizer such as predictive coding or transform coding. The encoder of VQ encodes a given set of k-dimensional data vectors $X=\{X_j | X_j \in R^k; j = 1, ..., T\}$ with a much smaller set of codewords $C=\{C_i | C_i \in R^k; i = 1, ..., N\}(N \ll T)$. Only the index i is sent to the decoder. The decoder has the same codebook as the encoder, and decoding is operated by table look-up procedure. The performance of data compression depends on a good codebook of representative vectors.

The LBG algorithm (Linde *et al.*, 1980) is an efficient VQ clustering algorithm. This algorithm is based either on a known probabilistic model or on a long training sequence of data. The main idea of this algorithm is the iterative application of a codebook modification operation where a distortion measure D is used to compute the cost $D(X_j, C_i)$ of reproducing the data vector X_j as the codeword C_i. Usually the Euclidean distortion measure is used to compute the cost. The iteration is terminated if the average distortion $D(X, C)$ converges. The iterative procedure is time consuming and it is difficult to apply the VQ clustering procedure for real time operation.

The computational complexity of the LBG algorithm can be significantly reduced if an efficient codeword search algorithm is applied to the partitioning of the data vectors. Many fast algorithms have been proposed to increase the speed of codeword search. Fischer and Patrick (Fischer & Patrick, 1970) presented a preprocessing algorithm to reorder the design sample such that a large

number of distance computations could be eliminated. Fukunaga and Narendra (Fukunaga & Narendra, 1975) proposed a branch and bound (BAB) algorithm for computing some nearest neighbours. BAB algorithm is a tree search algorithm using a hierarchical decomposition of the sample set of known patterns. They used the criterion of triangular inequality elimination to develop two rules to eliminate the distance computation in the tree classifier. Kamgar-Parsi and Kanal (Kamgar-Parsi & Kanal, 1985) added another two rules to the BAB algorithm to improve the computation time. Niemann and Goppert (Niemann & Goppert, 1988) combined these four rules into one and used a hierarchical partition of pattern sample algorithm to get more efficient computation time. Jiang and Zhang (Jiang & Zhang, 1993) developed a more efficient BAB tree search algorithm for finding the nearest neighbour to a new data vector in the codebook. All these efficient search methods described above are however not suitable to apply to VQ clustering algorithms due to the overhead of preprocessing required.

Bei and Gray (Bei & Gray, 1985) proposed the partial distortion search (PDS) algorithm to reduce computational complexity. PDS is a simple and efficient codeword search algorithm which has no extra storage or preprocessing requirements. The minimax method was proposed by Cheng et al. to derive a tentative match and improve the search efficiency (Cheng *et al.*, 1984). Vidal (Vidal, 1986) presented the approximating and eliminating search algorithm (AESA) in which the computation time is approximately constant for codeword search in a large codebook size. AESA is a very efficient algorithm to reduce multiplication operations for large codebook size but it needs a large number of comparison operations. Soleymoni and Morgera (Soleymani & Morgera, 1987b) proposed the absolute error inequality (AEI) elimination criterion to improve the speed of VQ search. Chen and Pan (Chen & Pan, 1989) applied the triangular inequality elimination (TIE) on VQ-based recognition of isolated words taking advantage of the high correlation characteristics between data vectors of adjacent speech frames. In this chapter, several fast clustering approaches based on the LBG algorithm (Linde *et al.*, 1980) are presented and compared.

4.2 Experimental Materials

The cepstrum of a signal is defined as the Fourier transform of the log of the signal spectrum. Cepstrum coefficients are used as the test features in the clustering experiments because they are commonly used in speech coding, speech synthesis, speech recognition and speaker recognition. The test materials for these experiments consist of two hundred words recorded from one male speaker. The speech is sampled at a rate of 16 kHz and 13-dimensional cepstrum coefficients with pre-emphasis value 0.98 are computed over 20 ms-wide frames with a 5 ms frame shift. A total of 20,030 analyzed frames are used in the VQ clustering experiments. Normally, this data set size is used to train up to 1024 codewords. Here, it is used in the VQ clustering experiments for 8 codewords to 1024 codewords.

4.3 LBG Algorithm

All of the fast VQ clustering algorithms (Pan *et al.*, 1994a; Pan *et al.*, 1996c) described in this chapter are based on the LBG algorithm (Linde *et al.*, 1980). This algorithm starts by assigning all the training data vectors to a single cluster, and proceeds by binary splitting until the desired number of clusters is achieved. After each splitting of the clusters there is an iterative procedure in which the cluster centroids are re-estimated and the data vectors are re-classified until the average distortion between the centroids and their classified vectors converges. The classification at each stage uses the full-search algorithm to find the nearest centroid to each vector. The detail algorithm based on unknown distribution is as follows:

Step 1: Set $m = 1$. Calculate centroid $C_1 = \frac{1}{T}\sum_{j=1}^{T} X_j$, where T is the total number of data vectors.

Step 2: Divide each centroid C_i into two close vectors $C_{2i-1} = C_i * (1+\delta)$ and $C_{2i} = C_i * (1-\delta)$, $1 \leq i \leq m$. Here δ is a small fixed perturbation scalar. Let $m = 2m$. Set $n = 0$, here n is the iterative times.

Step 3: Find the nearest neighbour to each data vector. Put X_j in the partitioned set P_i if C_i is the nearest neighbour to X_j.

Step 4: After obtaining the partitioned sets $P = (P_i; 1 \leq i \leq m)$, set $n = n + 1$. Calculate the overall average distortion $D_n = \frac{1}{T} \sum_{i=1}^{m} \sum_{j=1}^{T_i} D(X_j^{(i)}, C_i)$, where $P_i = \{X_1^{(i)}, X_2^{(i)}, ..., X_{T_i}^{(i)}\}$.

Step 5: Find centroids of all disjoint partitioned sets P_i by $C_i = \frac{1}{T_i} \sum_{j=1}^{T_i} X_j^{(i)}$.

Step 6: If $(D_{n-1} - D_n)/D_n > \epsilon$, go to step 3; otherwise go to step 7. Here ϵ is a small distortion threshold.

Step 7: If $m = N$, then take the codebook C_i as the final codebook; otherwise, go to step 2. Here N is the codebook size.

4.4 Previous Vector Candidate and Previous Partitioned Centre

In the VQ clustering procedure, speech data has the property that the classification result for the present vector is usually the same as or close to the classified result of the previous vector (Chen & Pan, 1989). Moreover, most of the vectors which are re-estimated in a full-search actually remain in the same partitioned set as for the previous re-estimation. With binary codeword splitting, the most probable partition to which data vectors belong can be chosen from the separated centres of the partitioned set. The previous vector candidate and previous partitioned centre can be used as tentative matches in the VQ clustering algorithm. Fig. 4.1 illustrates the relationship between the number of codewords and the probability that data vectors remain in the same partitioned set after re-estimation in full-search. For the fixed data vectors, the more codewords being generated, the larger is the probability that the data vectors belong to the same (previous) partitioned set. The probability is up to 0.949 for 1024 codewords. These results are averages across the re-estimation and re-classification iterations when $\delta = 0.01$ and $\epsilon = 0.005$.

4.5 Codebook Reorder Method

The codebook reorder method (Pan, 1988) is to reorder the codewords so as to increase the search efficiency. For speech encoding, it chooses the nearest codeword of the previous frame as a tentative match to encode the present frame. From training data, it is possible to calculate the probability of these codewords to be encoded and arrange these codewords in the order

of decreasing probability. The codeword search is operated from the most probable codeword to the least probable. It is simple and efficient to create a state table where these elements are indices of codewords and arranged in the increasing order of distortion between the most probable codeword and the other codewords. In the VQ clustering procedure, the previous vector candidate or previous partitioned centre can be chosen as the most probable codeword so as to create the state table. The computational complexity is $O(N^2 \log_2 N)$ using Heapsort (Press *et al.*, 1986) to establish the state table.

4.6 Fast Clustering Algorithms

4.6.1 APV-type clustering algorithm

An efficient clustering algorithm must include two key elements, i.e., a good tentative match and a powerful codeword elimination criterion. The previous vector candidate as the tentative match with AEI and PDS to improve the conventional clustering algorithm is proposed. This algorithm is called APV-type algorithm. It is described as follows.

step 1: Set $m = 1$. Calculate centroid $C_1 = \frac{1}{T} \sum_{j=1}^{T} X_j$, where T is the total number of data vectors.

step 2: Divide each centroid C_i into two close vectors $C_{2i-1} = C_i * (1+\delta)$ and $C_{2i} = C_i * (1-\delta)$, $1 \leq i \leq m$. Here δ is a small fixed perturbation scalar. Let $m = 2m$. Set $n = 0$, here n is the iterative times.

step 3: Compute the nearest neighbour for the first data vector X_1. For data vector X_j, use the nearest neighbour of X_{j-1} (previous vector candidate) as a tentative match and apply AEI with a PDS to find the nearest neighbour to each data vector. Put X_j in the partitioned set P_i if C_i is the nearest neighbour to X_j.

step 4: After obtaining the partitioned sets $P = (P_i; 1 \leq i \leq m)$, set $n = n + 1$. Calculate the overall average distortion $D_n = \frac{1}{T} \sum_{i=1}^{m} \sum_{j=1}^{T_i} D(X_j^{(i)}, C_i)$, where $P_i = \{X_1^{(i)}, X_2^{(i)}, ..., X_{T_i}^{(i)}\}$.

step 5: Find centroids of all disjoint partitioned sets P_i by $C_i = \frac{1}{T_i} \sum_{j=1}^{T_i} X_j^{(i)}$.

step 6: If $(D_{n-1} - D_n)/D_n > \epsilon$, go to step 3; otherwise go to step 7. Here ϵ is a small distortion threshold.

step 7: If $m = N$, then take the codebook C_i as the final codebook; otherwise, go to step 2. Here N is the codebook size.

4.6.2 APC-type Clustering Algorithm

The previous vector candidate is a very efficient tentative match for word recognition (Chen & Pan, 1989). It is not powerful compared with the previous partitioned centre in clustering algorithm because some adjacent data vectors are uncorrelated. It is possible to modify this clustering algorithm using the previous partitioned centre as a tentative match with AEI and PDS elimination criteria. This algorithm is referred to as APC-type algorithm and it is depicted as follows.

step 1: Set $m = 1$. Calculate centroid $C_1 = \frac{1}{T}\sum_{j=1}^{T} X_j$, where T is the total number of data vectors.

step 2: Divide each centroid C_i into two close vectors $C_{2i-1} = C_i * (1+\delta)$ and $C_{2i} = C_i * (1-\delta)$, $1 \leq i \leq m$. Here δ is a small fixed perturbation scalar. Let $m = 2m$. Set $n = 0$, here n is the iterative times.

step 3: For each data vector X_j, set $D_{min} = MIN(D(X_j, C_{2i-1}), D(X_j, C_{2i}))$, C_{2i-1} and C_{2i} are split from C_i associated to the partitioned set P_i to which X_j previously belonged. Choose C_{2i-1} or C_{2i} as the previous partitioned centre which is the nearest neighbour to X_j.

step 4: Use the previous partitioned centre as a tentative match and apply AEI with PDS to find the nearest neighbour to each data vector. Put X_j in the partitioned set P_i if C_i is the nearest neighbour to X_j.

step 5: After obtaining the partitioned sets $P = (P_i; 1 \leq i \leq m)$, set $n = n + 1$. Calculate the overall average distortion $D_n = \frac{1}{T}\sum_{i=1}^{m}\sum_{j=1}^{T_i} D(X_j^{(i)}, C_i)$, where $P_i = \{X_1^{(i)}, X_2^{(i)}, ..., X_{T_i}^{(i)}\}$.

step 6: Find centroids of all disjoint partitioned sets P_i by $C_i = \frac{1}{T_i}\sum_{j=1}^{T_i} X_j^{(i)}$.

step 7: If $(D_{n-1} - D_n)/D_n > \epsilon$, take C_i as the previous partitioned centre for each $X_j \in P_i$ and go to step 4; otherwise go to step 8. Here ϵ is a small distortion threshold.

step 8: If $m = N$, then take the codebook C_i as the final codebook; otherwise, go to step 2. Here N is the codebook size.

Fig. 4.2 shows the statistics for the elimination probability of APC-type using AEI criterion at each feature dimension. The previous partitioned centre is used as the initial codeword in this experiment. For 1024 codewords, 61.6% of impossible codeword matches will be eliminated by using AEI at the first dimension and only 0.5% codewords cannot be eliminated using AEI criterion.

4.6.3 APCH-type Clustering Algorithm

The hypercube approach provides the tighter bound than AEI for $s = 1$. The APC-type algorithm can be further improved by adding the hypercube approach to step 4, as an APCH-type algorithm. Use the previous partitioned centre as a tentative match. Check Eq. 3.5 to eliminate impossible codeword match. Apply AEI to eliminate the codeword which cannot be eliminated using hypercube approach. A PDS scheme is used for the codeword which cannot be eliminated using hypercube approach and AEI criterion. The detail of this algorithm is stated as follows:

step 1: Set $m = 1$. Calculate centroid $C_1 = \frac{1}{T}\sum_{j=1}^{T} X_j$, where T is the total number of data vectors.

step 2: Divide each centroid C_i into two close vectors $C_{2i-1} = C_i * (1+\delta)$ and $C_{2i} = C_i * (1-\delta)$, $1 \leq i \leq m$. Here δ is a small fixed perturbation scalar. Let $m = 2m$. Set $n = 0$, here n is the iterative times.

step 3: For each data vector X_j, set $D_{min} = MIN(D(X_j, C_{2i-1}), D(X_j, C_{2i}))$, C_{2i-1} and C_{2i} are split from C_i associated to the partitioned set P_i to which X_j previously belonged. Choose C_{2i-1} or C_{2i} as the previous partitioned centre which is the nearest neighbour to X_j.

step 4: Use the previous partitioned centre as a tentative match and apply AEI, hypercube approach and PDS to find the nearest neighbour to each data vector. Put X_j in the

partitioned set P_i if C_i is the nearest neighbour to X_j.

step 5: After obtaining the partitioned sets $P = (P_i; 1 \leq i \leq m)$, set $n = n + 1$. Calculate the overall average distortion $D_n = \frac{1}{T} \sum_{i=1}^{m} \sum_{j=1}^{T_i} D(X_j^{(i)}, C_i)$, where $P_i = \{X_1^{(i)}, X_2^{(i)}, ..., X_{T_i}^{(i)}\}$.

step 6: Find centroids of all disjoint partitioned sets P_i by $C_i = \frac{1}{T_i} \sum_{j=1}^{T_i} X_j^{(i)}$.

step 7: If $(D_{n-1} - D_n)/D_n > \epsilon$, take C_i as the previous partitioned centre for each $X_j \in P_i$ and go to step 4; otherwise go to step 8. Here ϵ is a small distortion threshold.

step 8: If $m = N$, then take the codebook C_i as the final codebook; otherwise, go to step 2. Here N is the codebook size.

Fig. 4.3 shows the statistics for the elimination probability of APCH-type using the hypercube approach at the first feature dimension and AEI criterion at the other feature dimension. The previous partitioned centre is used as the initial codeword in this experiment. For 1024 codewords, 88.9% of impossible codeword matches will be eliminated by using the hypercube approach at the first dimension and only 0.45% codewords cannot be eliminated using hypercube approach and AEI criterion. For 8 codewords and 64 codewords, only 8.3% and 2.9% codewords cannot be eliminated using hypercube approach and AEI criterion.

4.6.4 IPC-type Clustering Algorithm

The hypercube approach and AEI criterion are special cases of the improved AEI (IAEI) criterion. Here, the improved AEI criterion is adopted to increase the efficiency for the clustering algorithm in step 4. This algorithm is referred to as IPC-type algorithm. By applying Eq. 3.45, this criterion can be separated into several sections. For 13-dimensional cepstrum coefficients, it is possible to separate the improved AEI criterion into four sections. These four sections are to set h=1 to check the first dimension-difference, h=4 for the sum from the first dimension-difference to the fourth, h=9 for the sum from the first dimension-difference to the ninth and h=13 for the sum of all dimension-differences. A PDS scheme is used for the codeword which cannot be eliminated using the improved AEI criterion. The detail algorithm is stated as follows:

step 1: Set $m = 1$. Calculate centroid $C_1 = \frac{1}{T}\sum_{j=1}^{T} X_j$, where T is the total number of data vectors.

step 2: Divide each centroid C_i into two close vectors $C_{2i-1} = C_i * (1+\delta)$ and $C_{2i} = C_i * (1-\delta)$, $1 \leq i \leq m$. Here δ is a small fixed perturbation scalar. Let $m = 2m$. Set $n = 0$, here n is the iterative times.

step 3: For each data vector X_j, set $D_{min} = MIN(D(X_j, C_{2i-1}), D(X_j, C_{2i}))$, C_{2i-1} and C_{2i} are split from C_i associated to the partitioned set P_i to which X_j previously belonged. Choose C_{2i-1} or C_{2i} as the previous partitioned centre which is the nearest neighbour to X_j.

step 4: Use the previous partitioned centre as a tentative match and apply IAEI with PDS to find the nearest neighbour to each data vector. Put X_j in the partitioned set P_i if C_i is the nearest neighbour to X_j.

step 5: After obtaining the partitioned sets $P = (P_i; 1 \leq i \leq m)$, set $n = n + 1$. Calculate the overall average distortion $D_n = \frac{1}{T}\sum_{i=1}^{m}\sum_{j=1}^{T_i} D(X_j^{(i)}, C_i)$, where $P_i = \{X_1^{(i)}, X_2^{(i)}, ..., X_{T_i}^{(i)}\}$.

step 6: Find centroids of all disjoint partitioned sets P_i by $C_i = \frac{1}{T_i}\sum_{j=1}^{T_i} X_j^{(i)}$.

step 7: If $(D_{n-1} - D_n)/D_n > \epsilon$, take C_i as the previous partitioned centre for each $X_j \in P_i$ and go to step 4; otherwise go to step 8. Here ϵ is a small distortion threshold.

step 8: If $m = N$, then take the codebook C_i as the final codebook; otherwise, go to step 2. Here N is the codebook size.

Fig. 4.4 shows the statistics for the elimination probability of IPC-type using IAEI criterion at each feature dimension. The previous partitioned centre is used as the initial codeword in this experiment. For 1024 codewords, 88.9% of impossible codeword matches will be eliminated by using IAEI at the first dimension and only 0.38% codewords cannot be eliminated. For 8 codewords and 64 codewords, only 5.4% and 2.2% codewords cannot be eliminated using IAEI criterion.

4.6.5 TPC-type, ATPC-type and TPCR-type Clustering Algorithms

The triangular inequality elimination (TIE) criteria can also be applied to step 4 of the clustering algorithm. TPC-type is the clustering algorithm combining previous partitioned centre, TIE and PDS. An ATPC-type algorithm is the addition of AEI to the TPC-type algorithm, i.e., if the codeword cannot be eliminated using TIE, then apply AEI and PDS. A TPCR-type algorithm is the addition of a codebook reorder method to the TPC-type algorithm, i.e., reorder the codewords in increasing order of distortion between previous partitioned centre and these codewords before applying TIE. Fig. 4.5 illustrates the elimination probability using TIE combined with previous partitioned centre in VQ clustering procedure. For 1024 codewords, the elimination probability is 0.949.

4.7 Experiments and Results

The test materials used in the VQ clustering experiments are described in the Section 4.2. To verify these fast algorithms, the mathematical operations (multiplications, comparisons, and additions) are used to calculate the computational efficiency. The experiments are carried out by setting the small fixed perturbation scalar to 0.01 and the small distortion threshold to 0.005. Twelve approaches are compared in the VQ clustering procedure. The conventional exhaustive method is referred to as CVT-type. P-type and T-type are approaches using PDS and TIE in codebook design. TP-type is the approach using TIE to eliminate unlikely codeword matches, then applying PDS to the codeword search. TPC-type is the algorithm using the previous partitioned centre as the most probable matching with TIE and PDS to reduce the clustering time. TPCR-type is the TPC-type with codebook reorder method. It is called an APC-type if the previous partitioned centre is used as the tentative match with AEI and PDS to accelerate the clustering speed. Using the previous vector candidate instead of the previous partitioned centre in an APC-type is called the APV-type. The ATPC-type is an algorithm combining TIE, AEI, PDS and the previous partitioned centre. The APCH-type is the addition of the hypercube approach to the APC-type. IPC-type is an algorithm combining the previous partitioned centre, improved AEI and PDS. The combination of previous partitioned centre, hypercube approach

and partial distortion search is referred to as the PCH-type.

The experimental results for 8 codewords to 1024 codewords are shown in Table 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8. For a general processor architecture, the multiplication operation is more expensive than the comparison operation and the addition operation. It is better to use an IPC-type algorithm for large codebook size and a TPC-type algorithm or TPCR-type algorithm for small codebook size. Table 4.1 to Table 4.8 also illustrate the number of total mathematical operations. In terms of the total number of operations, TPC-type outperforms all of the above algorithms. It needs extra computation time to generate the distortion table for TIE approach and that is why the total number of multiplications in ATPC-type, TPC-type and TPCR-type are larger than IPC-type, APC-type, APV-type, PCH-type and APCH-type for 1024 codewords. The codebook reorder method is not very efficient in the VQ clustering algorithm owing to the overhead of the sorting procedure. In small codebook size (such as 8 codewords), TPCR-type is excellent. It is not however superior compared with IPC-type, APCH-type, APC-type, APV-type, ATPC-type and TPC-type for large codebook size. For other codebook sizes between 8 codewords and 1024 codewords, these fast VQ clustering algorithms are also very efficient in computation.

The comparison of elimination probability of APC-type, APCH-type and IPC-type algorithms for 16 codewords is shown in Fig. 4.6. For 16 codewords, the elimination probability of APCH-type algorithm and APC-type algorithm are 0.85 and 0.37 at the first dimension. This means that the hypercube approach is efficient. The elimination probability of the APCH-type algorithm is the same as the IPC-type algorithm at the first feature dimension. At other feature dimensions, the elimination probability of the IPC-type algorithm is higher than the APCH-type algorithm. The $IAEI$ criterion is superior to the AEI criterion with hypercube approach in the VQ clustering algorithm. Fig. 4.7 and 4.8 illustrate the saving in the number of multiplications at each iteration of IPC-type, PCH-type, TPC-type and ATPC-type algorithms for 128 and 1024 codewords. Fig. 4.9 and 4.10 illustrate the saving in the total number of mathematical operations at each iteration of IPC-type, PCH-type, TPC-type and ATPC-type algorithms for 128 and 1024

codewords. The comparative efficiency of these algorithms are only influenced a little by the number of iterations.

To sum up, the IPC-type algorithm, which is a combination of the previous partitioned centre, the improved absolute error inequality criterion and the partial distortion search, is judged to be the best VQ clustering algorithm approach for general processors. In contrast, the TPC-type algorithm, which is a combination of the previous partitioned centre, the triangular inequality elimination and the partial distortion search is judged to be the most suitable approach for DSP processors.

method	mul($\times 10^6$)	cmp($\times 10^6$)	add($\times 10^6$)	sum($\times 10^6$)	saving sum	saving mul
IPC	5.72	2.92	18.0	26.6	61.4 %	73.5 %
APCH	5.82	3.57	18.5	27.9	59.6 %	73.1 %
APC	5.88	6.87	25.0	37.8	45.2 %	72.8 %
APV	5.90	8.53	26.1	40.5	41.3 %	72.7 %
ATPC	5.87	5.87	21.3	33.0	52.2 %	72.8 %
PCH	5.72	1.94	15.8	23.5	65.9 %	73.5 %
TPC	5.63	2.30	15.0	22.9	66.8 %	73.9 %
TPCR	5.63	2.30	15.0	22.9	66.8 %	73.9 %
TP	13.6	11.1	30.4	55.1	20.1 %	37.0 %
P	14.0	13.8	30.9	58.7	14.9 %	54.3 %
T	16.2	2.31	35.7	54.2	21.4 %	33.3 %
CVT	21.6	1.36	46.0	69.0	0 %	0 %

Table 4.1: Computational complexity of VQ clustering for 8 codewords

method	mul($\times 10^6$)	cmp($\times 10^6$)	add($\times 10^6$)	sum($\times 10^6$)	saving sum	saving mul
IPC	8.59	6.98	30.0	45.6	71.1 %	83.1 %
APCH	8.77	8.45	31.3	48.5	69.3 %	82.7 %
APC	8.87	15.2	45.0	69.1	56.3 %	82.5 %
APV	9.41	18.4	47.8	75.6	52.2 %	81.5 %
ATPC	8.86	12.8	35.9	57.6	63.5 %	82.6 %
PCH	9.13	4.90	25.4	39.4	75.1 %	82.0 %
TPC	8.91	5.59	23.0	37.5	76.3 %	82.5 %
TPCR	8.91	5.60	23.0	37.5	76.3 %	82.5 %
TP	28.1	25.8	60.0	114	27.8 %	44.7 %
P	29.3	29.0	61.3	120	24.1 %	42.3 %
T	35.2	5.73	74.3	115	27.2 %	30.7 %
CVT	50.8	3.47	104	158	0 %	0 %

Table 4.2: Computational complexity of VQ clustering for 16 codewords

method	mul($\times 10^6$)	cmp($\times 10^6$)	add($\times 10^6$)	sum($\times 10^6$)	saving sum	saving mul
IPC	12.5	17.2	52.1	81.8	78.8 %	90.1 %
APCH	12.8	20.6	55.8	89.2	76.9 %	89.8 %
APC	13.0	34.6	84.6	132	65.8 %	89.7 %
APV	14.7	40.9	91.8	147	61.9 %	88.3 %
ATPC	13.0	28.4	61.6	103	73.3 %	89.7 %
PCH	15.0	12.7	42.3	70.0	81.9 %	88.1 %
TPC	14.5	14.1	35.6	64.2	83.4 %	88.5 %
TPCR	14.4	14.1	35.6	64.1	83.4 %	88.6 %
TP	60.2	61.1	123	244	36.8 %	52.2 %
P	63.8	63.6	127	254	34.2 %	49.4 %
T	81.0	14.7	165	260.7	32.5 %	35.7 %
CVT	126	9.1	251	386	0 %	0 %

Table 4.3: Computational complexity of VQ clustering for 32 codewords

method	mul($\times 10^6$)	cmp($\times 10^6$)	add($\times 10^6$)	sum($\times 10^6$)	saving sum	saving mul
IPC	15.9	32.0	78.3	126	82.9 %	93.4 %
APCH	16.3	37.9	85.4	140	81.0 %	93.3 %
APC	16.5	61.0	133	211	71.4 %	93.2 %
APV	19.7	71.8	147	239	67.6 %	91.9 %
ATPC	16.7	48.6	88.3	154	79.1 %	93.1 %
PCH	21.6	24.7	62.2	109	85.2 %	91.1 %
TPC	20.3	26.4	48.0	94.7	87.2 %	91.6 %
TPCR	20.2	26.8	48.0	95.0	87.1 %	91.7 %
TP	99.4	107	199	405	45.1 %	58.9 %
P	107	107	207	421	43.0 %	55.8 %
T	147	28.4	294	469	36.4 %	39.3 %
CVT	242	17.9	478	738	0 %	0 %

Table 4.4: Computational complexity of VQ clustering for 64 codewords

method	mul($\times 10^6$)	cmp($\times 10^6$)	add($\times 10^6$)	sum($\times 10^6$)	saving sum	saving mul
IPC	20.0	63.3	127	210	86.4 %	96.1 %
APCH	20.5	74.4	141	236	84.7 %	96.0 %
APC	20.7	115	225	361	76.6 %	95.9 %
APV	26.6	135	255	417	72.9 %	94.8 %
ATPC	21.8	87.7	131	241	84.4 %	95.7 %
PCH	33.2	51.3	99.2	184	88.1 %	93.5 %
TPC	29.7	53.0	67.0	149.7	90.3 %	94.2 %
TPCR	29.7	54.7	67.0	151.4	90.2 %	94.2 %
TP	163	188	318	669	56.6 %	68.0 %
P	181	181	337	699	54.6 %	64.4 %
T	287	59.3	565	911	40.8 %	43.6 %
CVT	509	38.2	993	1540	0 %	0 %

Table 4.5: Computational complexity of VQ clustering for 128 codewords

method	mul($\times 10^6$)	cmp($\times 10^6$)	add($\times 10^6$)	sum($\times 10^6$)	saving sum	saving mul
IPC	24.4	122	209	355	88.7 %	97.7 %
APCH	25.0	141	234	400	87.3 %	97.6 %
APC	25.2	210	381	616	80.4 %	97.6 %
APV	35.2	249	441	752	76.1 %	96.6 %
ATPC	29.7	154	191	375	88.1 %	97.1 %
PCH	51.4	103	163	317	89.9 %	95.1 %
TPC	44.6	103	95.8	243	92.3 %	95.7 %
TPCR	44.5	111	95.6	251	92.0 %	95.7 %
TP	256	316	486	1058	66.3 %	75.4 %
P	295	295	526	1116	64.5 %	71.7 %
T	546	120	1065	1731	44.9 %	47.6 %
CVT	1042	79.1	2021	3142	0 %	0 %

Table 4.6: Computational complexity of VQ clustering for 256 codewords

method	mul($\times 10^6$)	cmp($\times 10^6$)	add($\times 10^6$)	sum($\times 10^6$)	saving sum	saving mul
IPC	28.5	218	335	582	90.2 %	98.6 %
APCH	29.2	250	376	655	89.0 %	98.5 %
APC	29.4	362	616	1007	83.1 %	98.5 %
APV	44.8	431	731	1207	79.7 %	97.7 %
ATPC	45.8	256	275	577	90.3 %	97.7 %
PCH	77.6	190	265	533	91.0 %	96.1 %
TPC	69.6	186	143	399	93.3 %	96.5 %
TPCR	69.5	220	143	433	92.7 %	96.5 %
TP	385	503	713	1601	73.1 %	80.5 %
P	458	458	783	1699	71.4 %	76.8 %
T	967	223	1877	3067	48.4 %	51.1 %
CVT	1976	151	3817	5944	0 %	0 %

Table 4.7: Computational complexity of VQ clustering for 512 codewords

method	mul($\times 10^6$)	cmp($\times 10^6$)	add($\times 10^6$)	sum($\times 10^6$)	saving sum	saving mul
IPC	33.0	427	594	1054	91.5 %	99.2 %
APCH	33.8	481	663	1178	90.5 %	99.2 %
APC	34.1	680	1082	1796	85.5 %	99.2 %
APV	58.6	812	1323	2194	82.2 %	98.6 %
ATPC	105	463	465	1033	91.6 %	97.4 %
PCH	127	384	479	990	92.0 %	96.9 %
TPC	143	366	283	792	93.6 %	96.5 %
TPCR	143	539	282	964	92.2 %	96.5 %
TP	645	871	1165	2681	78.3 %	84.3 %
P	777	777	1259	2813	77.2 %	81.1 %
T	1887	453	3649	5989	51.5 %	54.1 %
CVT	4109	315	7922	12346	0 %	0 %

Table 4.8: Computational complexity of VQ clustering for 1024 codewords

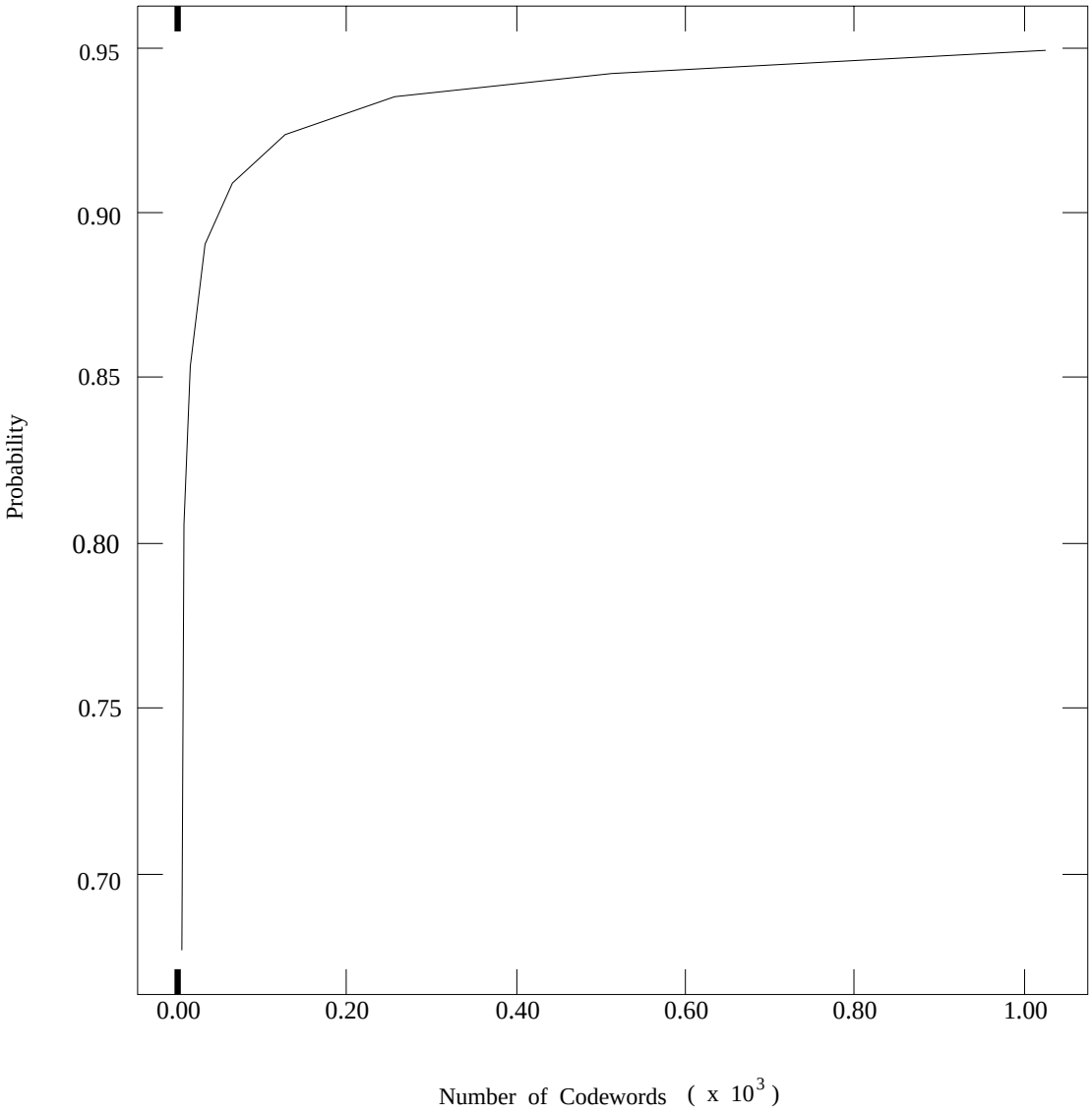

Figure 4.1: Relationship between the number of codewords and the probability of the data vectors belonging to the previous partitioned set

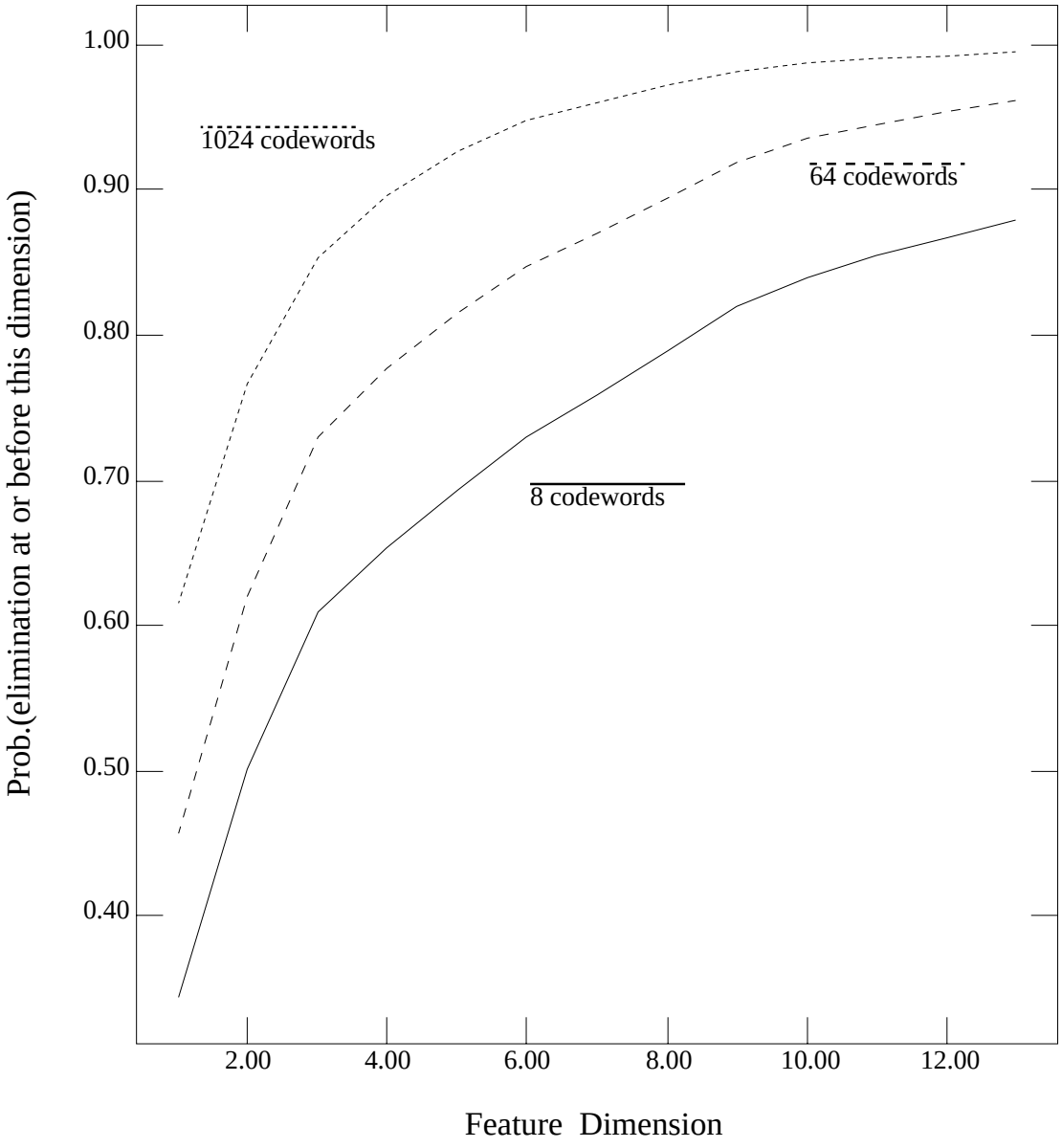

Figure 4.2: The elimination probability of APC-type using AEI at each feature dimension

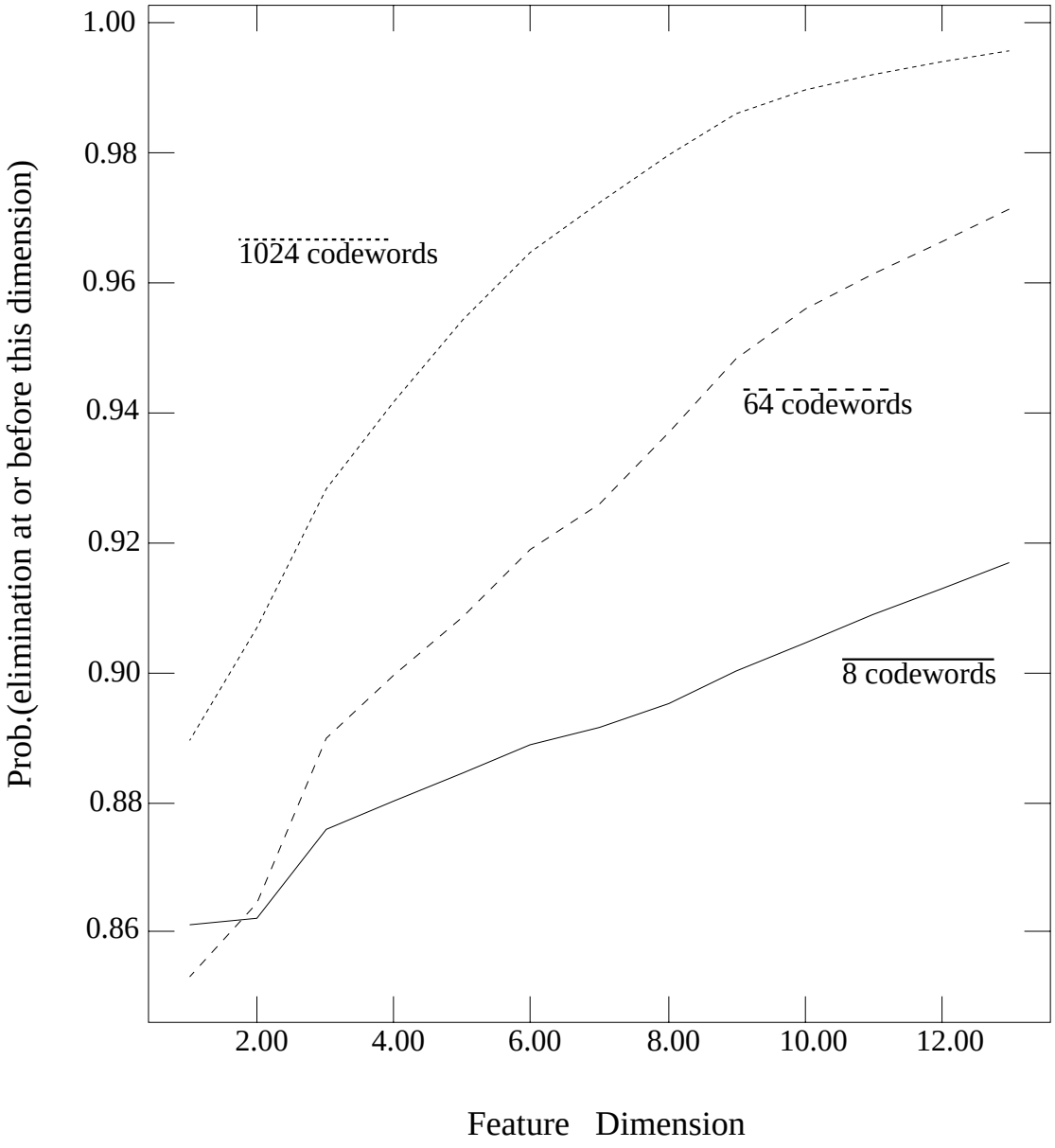

Figure 4.3: The elimination probability of APCH-type at each feature dimension

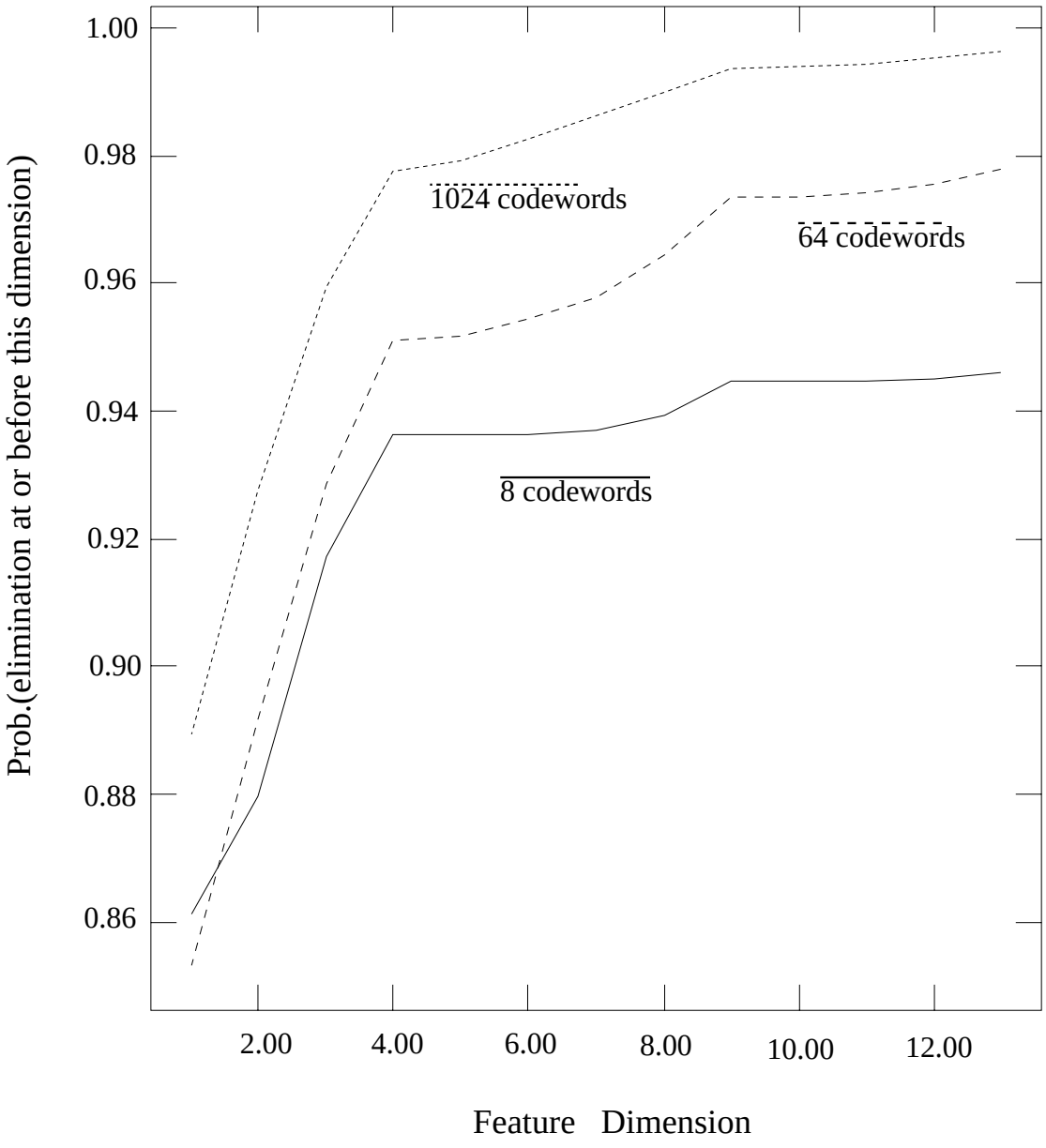

Figure 4.4: The elimination probability of IPC-type using IAEI at each feature dimension

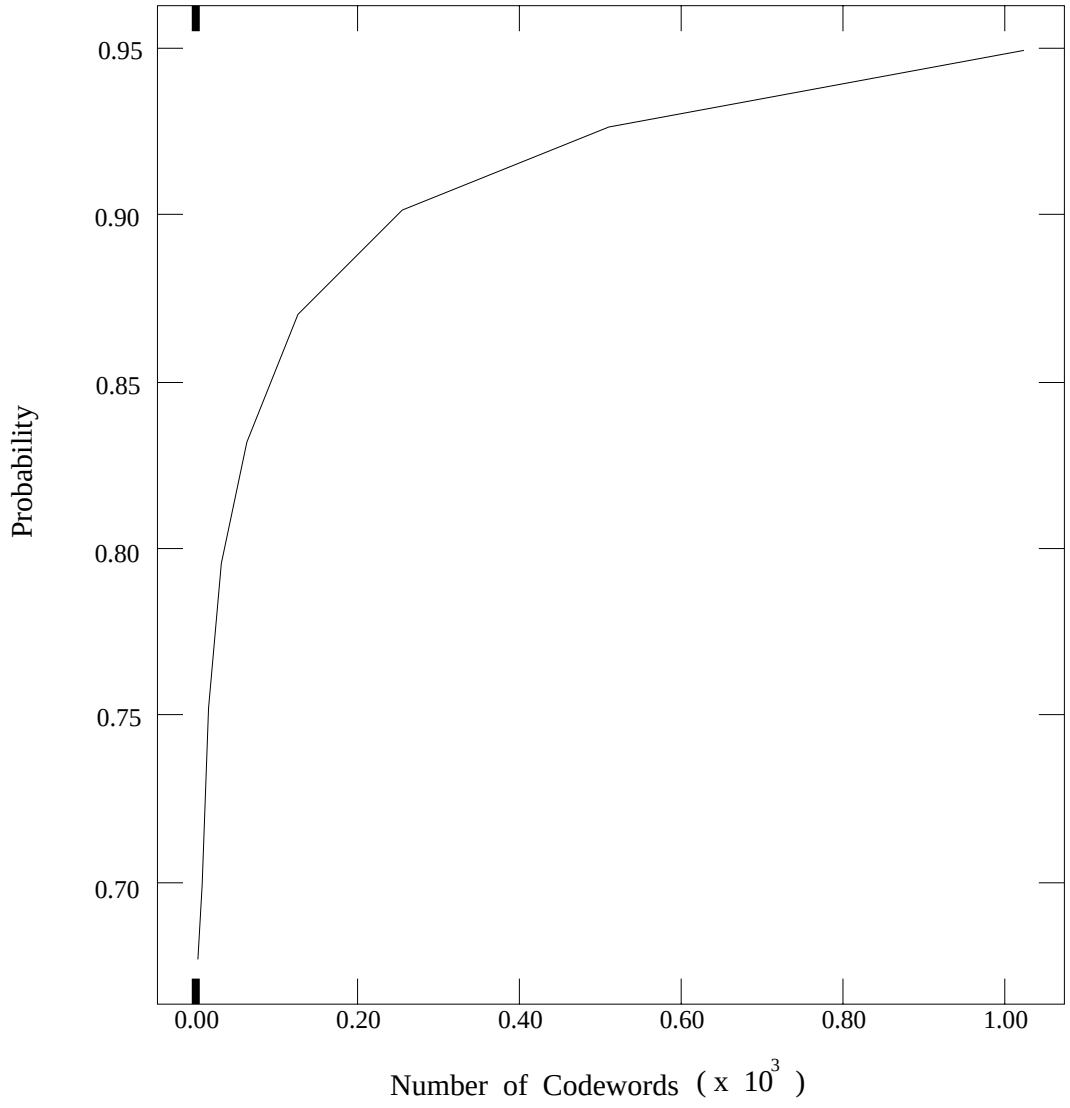

Figure 4.5: Relationship between the number of codewords and the elimination probability using TIE

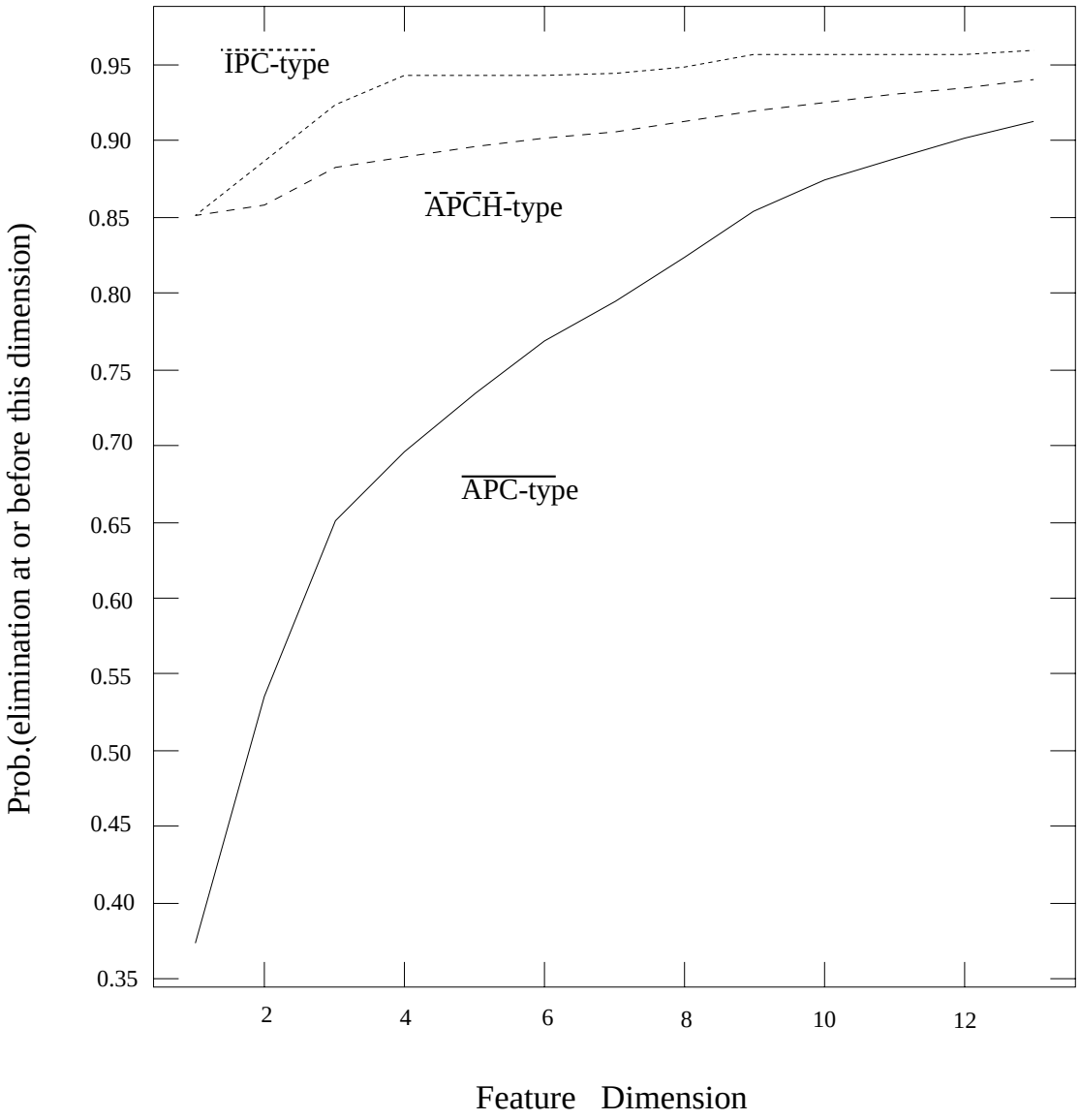

Figure 4.6: Comparison of elimination probability for 16 codewords

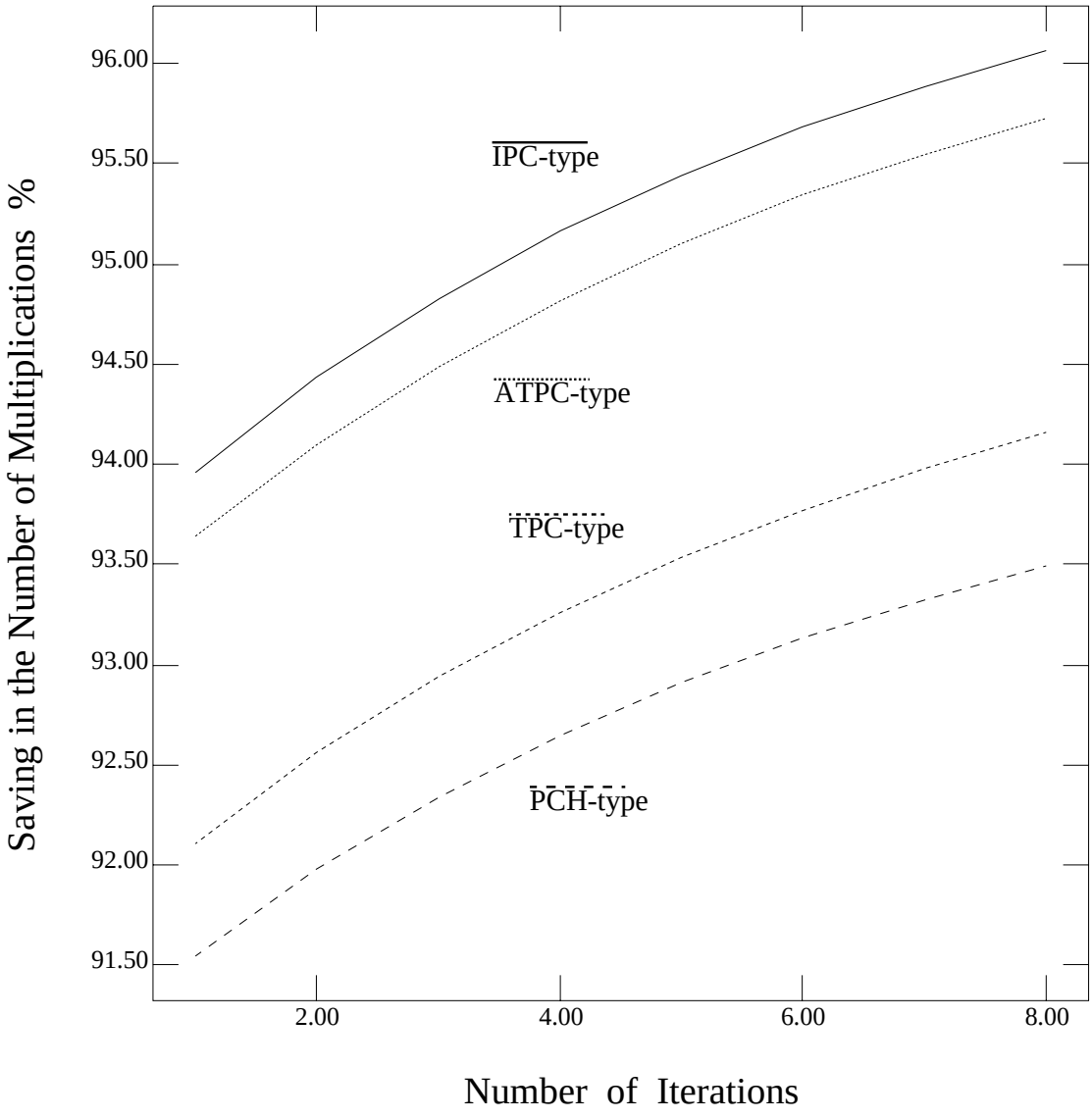

Figure 4.7: Saving in the number of multiplications at each iteration for 128 codewords

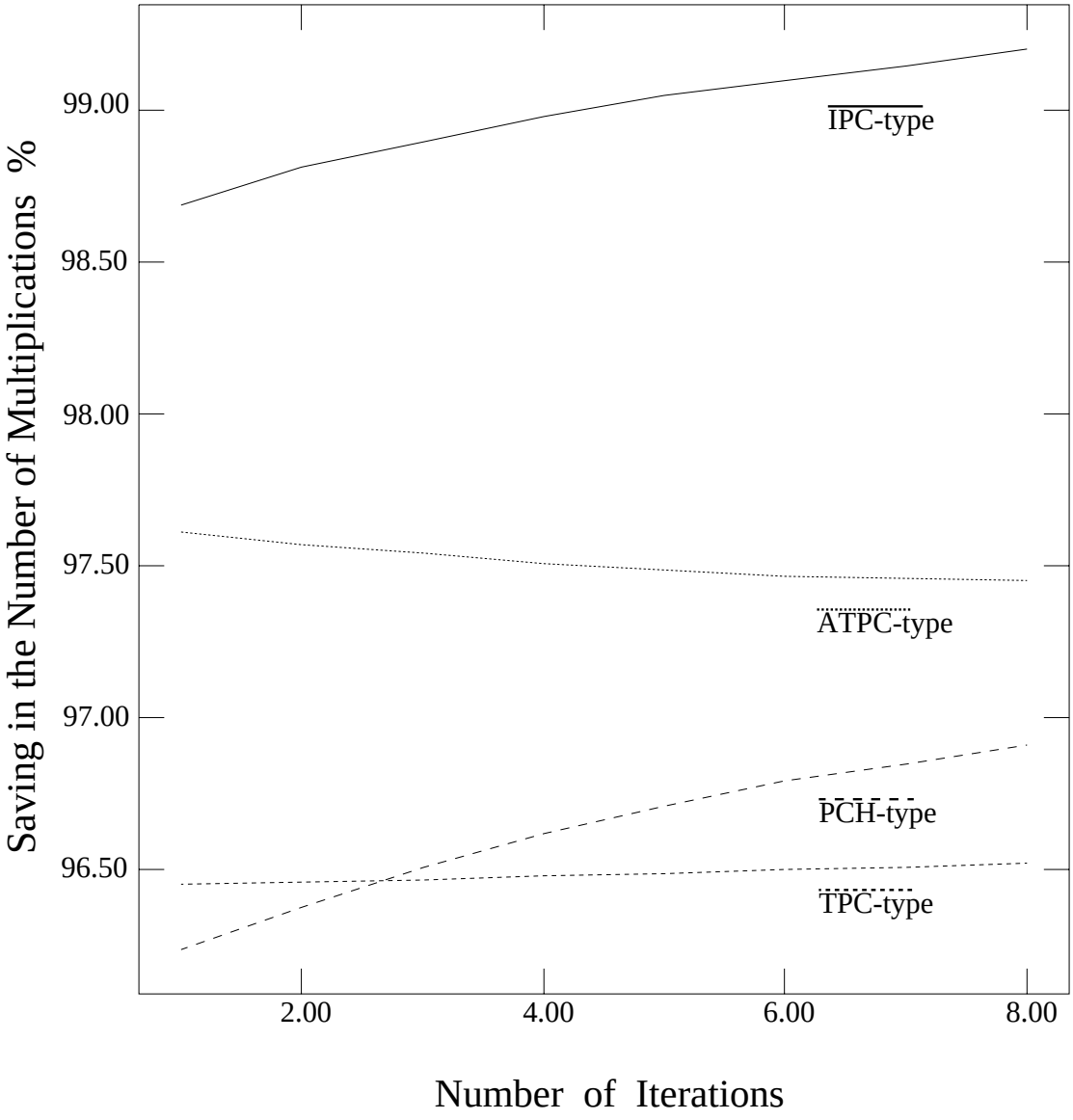

Figure 4.8: Saving in the number of multiplications at each iteration for 1024 codewords

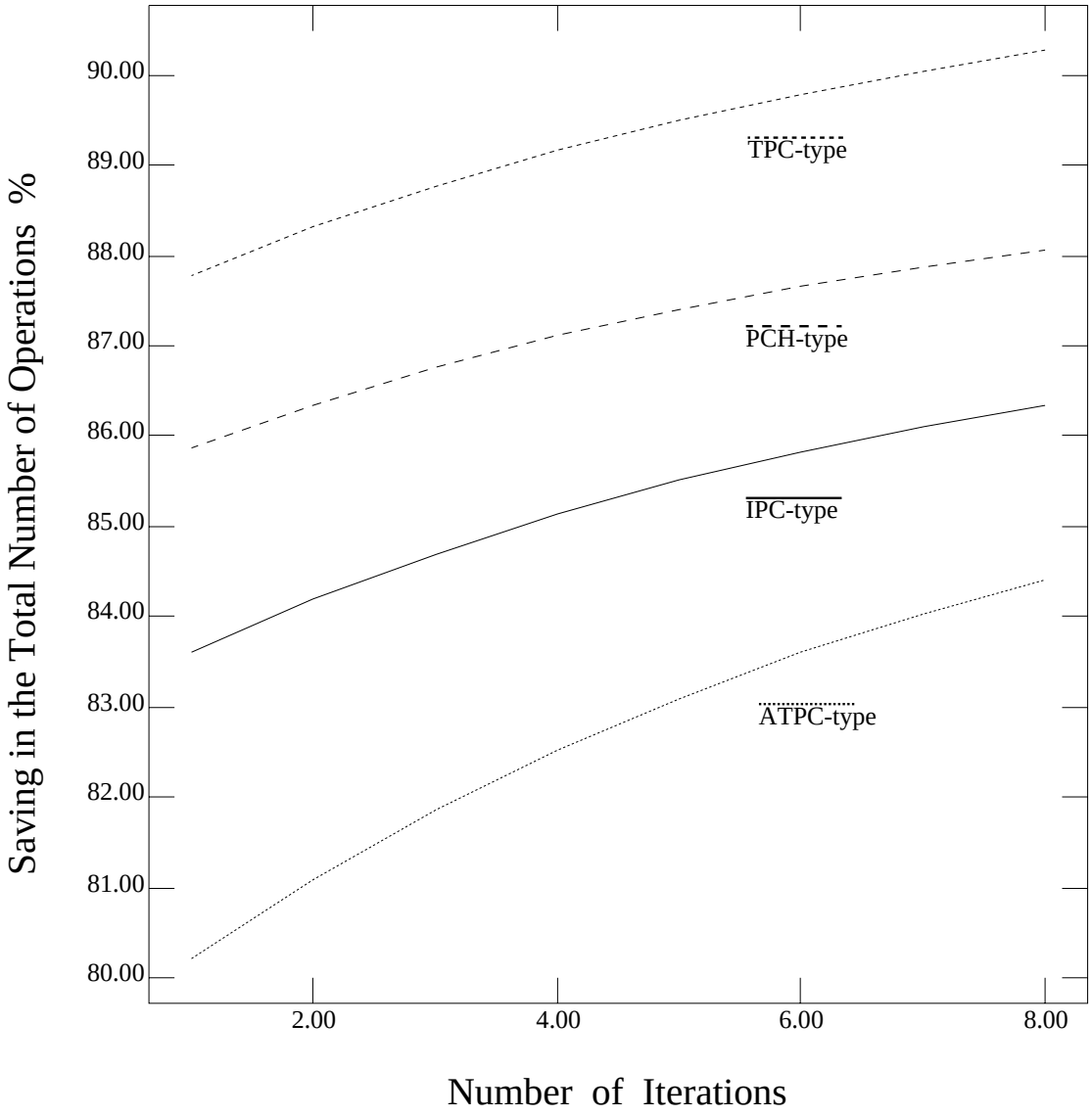

Figure 4.9: Saving in the total number of mathematical operations at each iteration for 128 codewords

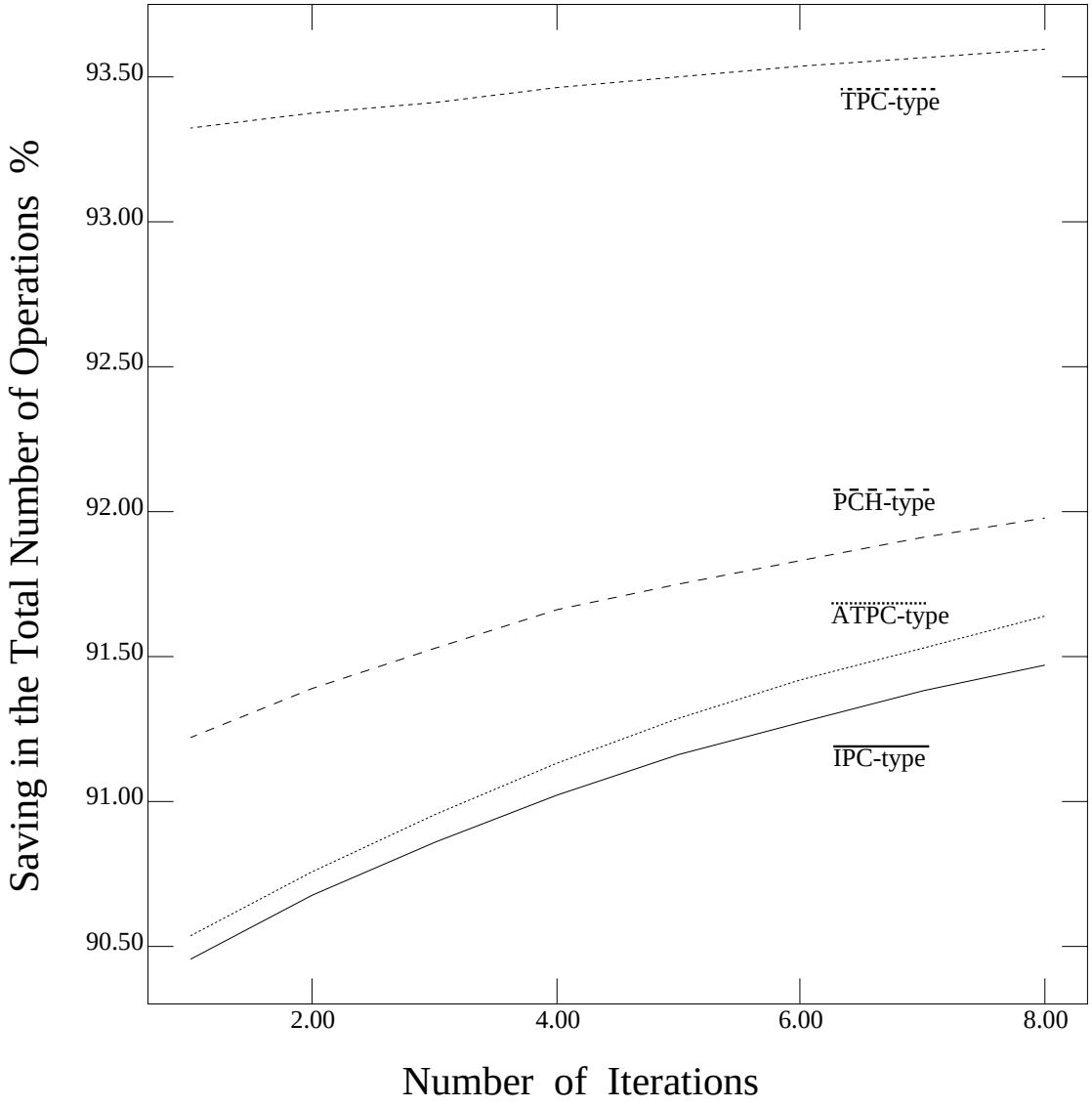

Figure 4.10: Saving in the total number of mathematical operations at each iteration for 1024 codewords

Chapter 5

Improved Algorithms for VQ Codebook Design

5.1 Introduction

Data compression using vector quantization (VQ) has received great attention because of its promising compression ratio and simple implemented structure. For the simplest VQ implementation, it separates the signal into several sections and compresses each section into one vector. Each vector of the signal to be compressed is compared to the codevectors of a codebook. The address of the codevector most similar to the signal vector is sent to the receiver. At the receiver, the decoder accesses a codevector from an identical codebook, thus an approximation of the original signal is reconstructed. Compression is obtained by sending the index of the particular codevector thereby requiring fewer bits than sending the signal vector. The key to VQ data compression is a good codebook design.

Suppose that there are T training data vectors X_j, $j = 1, 2, ..., T$ and N codevectors C_i, $i = 1, 2, ..., N$, are generated from these training data vectors. The training data vectors are partitioned into N sets S_i and C_i is the centroid of the training data vectors in the partitioned set S_i. The criterion of the VQ codebook design can be formulated as the following mathematical form:

$$\text{minimize} \qquad f(W, X, C) = \sum_{j=1}^{T} \sum_{i=1}^{N} w_{ij} D(X_j, C_i), \tag{5.1}$$

subject to the following constraints:

$$\sum_{i=1}^{N} w_{ij} = 1, \quad 1 \le j \le T \tag{5.2}$$

$$w_{ij} = 0 \text{ or } 1, \tag{5.3}$$

where $X = \{X_1, X_2, ..., X_T\}$, $X_j \equiv$ jth training data vector, $C = \{C_1, C_2, ..., C_N\}$, $C_i \equiv$ ith centroid vector, $W \equiv$ a $N \times T$ matrix,

$$w_{ij} = \begin{cases} 1, & \text{if } X_j \in S_i \\ 0, & \text{if } X_j \notin S_i \end{cases}$$

$T \equiv$ the total number of training data vectors, $N \equiv$ the number of codevectors $D(X_j, C_i) \equiv$ the distortion between the data vector X_j and the codevector C_i.

If the squared Euclidean distortion measure is applied, then the criterion of the VQ codebook design can be expressed as

$$\text{minimize} \qquad f(W, X, C) = \sum_{j=1}^{T} \sum_{i=1}^{N} w_{ij} \sum_{p=1}^{k} (x_j^p - c_i^p)^2, \tag{5.4}$$

where k is the number of dimensions, $C_i = \{c_i^1, c_i^2, ..., c_i^k\}$ and $X_j = \{x_j^1, x_j^2, ..., x_j^k\}$. The matrix W can be considered as the partitioned results of the training data vectors and from the matrix W, the codevector can be obtained as

$$C_i = \frac{1}{|S_i|} \sum_{j=1}^{T} w_{ij} X_j, \tag{5.5}$$

where $|S_i|$ denotes the number of training data vectors in the partitioned set S_i or the number of non-zero w_{ij}, $j = 1, 2, ..., T$. The number of possible codebooks generated from these training

data vectors (Anderberg, 1973) is

$$\frac{1}{N!}\sum_{i=0}^{N}(-1)^{N-i}\binom{N}{i}i^{T}, \tag{5.6}$$

If all the possible codebooks are tested, then a globally optimal codebook can be obtained. Unfortunately, such computation is normally prohibitive, making any kind of exhaustive search unrealistic even for the most powerful computers with relatively small values of codebook size N and the number of training data vectors T. In order to overcome this difficulty, many algorithms were applied to codebook design to produce sub-optimal codebook designs, such as the K-means algorithm, ISODATA clustering algorithm, GLA algorithm, pairwise nearest neighbour (PNN) algorithm, fuzzy C-means clustering algorithm, simulated annealing method, stochastic relaxation approach, continuation method and deviation reduction algorithm which will be described in the following subsections. New codebook design procedures using genetic algorithms and genetic algorithms coupled with the stochastic relaxation approach will be presented and experimental comparison of these algorithms with GLA will be presented in section 5.3.

5.1.1 K-means and ISODATA Clustering Algorithms

The K-means algorithm (MacQueen, 1967) is a well known iterative procedure for the clustering problem. It is also known as the C-means algorithm or basic ISODATA clustering algorithm. This algorithm can also be applied to VQ codebook design, and the K-means algorithm can be depicted as follows:

Step 1: Randomly select N training data vectors as the initial codevectors C_i, $i = 1, 2, ..., N$ from T training data vectors.

Step 2: For each training data vector X_j, $j = 1, 2, ..., T$, assign X_j to the partitioned set S_i if

$$i = \mathrm{argmin}_l D(X_j, C_l). \tag{5.7}$$

Step 3: Compute the centroid of the partitioned set (codevector) using

$$C_i = \frac{1}{|S_i|} \sum_{X_j \in S_i} X_j \tag{5.8}$$

where $|S_i|$ denotes the number of training data vectors in the partitioned set S_i. If there is no change in the clustering centroids, then terminate this program; otherwise, go to step 2.

The ISODATA clustering algorithm (Ball & Hall, 1967) is a highly interactive version of the K-means algorithm. This algorithm is characterized by the addition of several heuristics to eliminate, aggregate and/or split clusters based on several predefined parameters (Ball & Hall, 1967).

5.1.2 GLA Algorithm

An iterative nonvariational technique for the design of scalar quantizer has been reported (Lloyd, 1982). Linde, Buzo and Gray (LBG) extended Lloyd's (Linde *et al.*, 1980) basic approach to the general case of vector quantizer. It is called the LBG algorithm and it is also known as the Generalised Lloyd algorithm (GLA). The GLA algorithm is a well known codebook design algorithm and has been described in Section 4.3 where the basic idea of finding the centroids of partitioned sets and the minimum distortion partitions is the same as the K-means algorithm. In the K-means algorithm, the initial centroids are selected randomly from the training vectors and the training vectors are added to the training procedure one at a time. The training procedure terminates when the last vector is incorporated. In contrast in the GLA algorithm, the initial centroids are generated from all of the training data by applying the splitting procedure and all the training vectors are incorporated to the training procedure at each iteration. Normally, the K-means algorithm is used to group data and the groups can change with time; the GLA algorithm is applied to generate the centroids and the centroids cannot change with time.

5.1.3 Pairwise Nearest Neighbour Algorithm

An *agglomerative* clustering approach is a process in which each training data is placed in its own cluster and these atomic clusters are gradually merged into larger and larger clusters until the desired objective is attained. A *divisive* clustering approach reverses the process of *agglomerative* clustering approach by starting with all training data in one cluster and subdividing into several smaller clusters. The GLA algorithm starts from one cluster and then separates this cluster to two clusters, four clusters, and so on until N clusters are generated, where N is the desired number of clusters or codebook size. Therefore the GLA algorithm is a *divisive* clustering approach. The pairwise nearest neighbour (PNN) algorithm (Equitz, 1989; Equitz, 1987) is an *agglomerative* clustering approach. PNN is actually identical to Ward's hierarchical clustering method (Bottemiller, 1992; Ward, 1963) published in 1963.

The PNN algorithm begins with a separate cluster for each vector in the training set and merges together two clusters at a time until the desired number of codevectors is achieved. At the start of this algorithm, there are T training data vectors and each data vector corresponds to a codevector, i.e., the codebook size is T. Then, these T clusters are converted to $T - 1$ clusters by merging together into a single cluster the two closest clusters. This merging process is repeated until the number of clusters is equal to the desired number of codevectors or the average distortion is greater than the predefined maximum average distortion.

The pairwise nearest neighbour algorithm can be depicted as follows:

Step 1: Set the current number of codevectors $\nu = T$ and the codevector C_i which belongs to the partitioned set S_i, is the training data vector, $i = 1, 2, ..., T$. T is the total number of training data vectors.

Step 2: Calculate the pair distortion $d(C_i, C_j)$ between the codevectors C_i and C_j, $1 \leq i \leq \nu$, $i < j \leq \nu$.

Step 3: Merge two partitioned sets S_i and S_j to S_i and calculate new codevector $C_i = \frac{n_i C_i + n_j C_j}{n_i + n_j}$ if $d(C_i, C_j) = \{\min d(C_m, C_n), 1 \leq n \leq v, n < m \leq v\}$. Set $C_j = C_v$, $S_j = S_v$ and $v = v - 1$. Here, n_i and n_j are the number of training data vectors in the partitioned sets S_i and S_j.

Step 4: Calculate the average distortion for the partitioned sets.

Step 5: Terminate the program if $v = N$ or the average distortion is greater than the predefined maximum average distortion, where N is the desired codebook size.

The pairwise nearest neighbour algorithm is an *agglomerative* clustering approach in which pairs of clusters are progressively merged together. The key to efficient execution of this algorithm is to find the closest pairs of centroids quickly among all centroids. The obvious way is to explicitly find each centroid's nearest neighbour, but this requires at least a $\log_2 T$ search (Equitz, 1989) and leads to a complexity of $O(T \log T)$ for each merge. In order to reduce the computation complexity, a K-d tree structure (Bentley, 1975; Friedman *et al.*, 1977) can be applied to reduce the complexity of the entire PNN algorithm to $O(T \log_2 T)$ and it is independent of the number of codevectors.

5.1.4 Simulated Annealing Method

Simulated annealing (SA) (Kirkpatrick *et al.*, 1983; Bohachevsky *et al.*, 1986) is a random search method which has been presented for optimization of NP-hard problems. Vechi and Kirkpatrick applied a simulated annealing method to the optimization of a wiring problem (Vecchi & Kirkpatrick, 1983). Gamal et al. also used the method of simulated annealing to construct good source codes, error-correcting codes and spherical codes (Gamal *et al.*, 1987). Cetin and Weerackody first proposed the method of simulated annealing in vector quantizer design (Cetin & Weerackody, 1988). There are also many algorithms involving simulated annealing for codebook design (Vaisey & Gersho, 1988; Flanagan *et al.*, 1989; Lu & Morrell, 1991). The basic algorithm of simulated annealing for codebook design can be stated as follows:

Step 1: The training data X_j, $j = 1, 2, ..., T$, is partitioned into the partitioned set S_i, $i = 1, 2, ..., N$ randomly. Set $n = 0$ and calculate the codevector

$$C_i = \frac{1}{|S_i|} \sum_{X_j \in S_i} X_j \tag{5.9}$$

where $|S_i|$ denotes the number of training data vectors in the partitioned set S_i.

Step 2: The codebook is perturbed by randomly selecting a data vector and moving this data vector from its current partitioned set to the different randomly selected partitioned set. Calculate the new centroids.

Step 3: The change in distortion $\triangle D$ is defined as the distortion of current codebook minus the distortion of previous codebook. The perturbation is accepted if

$$e^{-\frac{\triangle D}{\hat{T}_n}} > r, \tag{5.10}$$

where r is a random value generated uniformly on the interval [0,1].

Step 4: If the distortion of the current codebook reaches the desired value or the iterative number n reaches the predetermined value, then terminate the program; otherwise, set $n = n + 1$ and go to step 2.

This algorithm starts with an initial temperature $\hat{T}_0$. The temperature sequence $\hat{T}_0$, $\hat{T}_1$, $\hat{T}_2$,... are positive numbers which is called an annealing schedule where

$$\hat{T}_0 \geq \hat{T}_1 \geq \hat{T}_2 ... \tag{5.11}$$

and

$$\lim_{n \to \infty} \hat{T}_n = 0. \tag{5.12}$$

If the resulting codebook decreases the distortion, the movement of the data is accepted. If the distortion is increased, it is accepted with the condition as in Eq. 5.10. Obviously, the perturbation is accepted easily for the earlier temperature and it is difficult to be accepted at

the final temperature. By accepting the perturbation for positive $\triangle D$ in some probability gives the opportunity for jumping off the local optimum. The performance of the codebook design depends on the annealing schedule.

5.1.5 Stochastic Relaxation Approach

In the previous subsection, the basic algorithm of simulated annealing in codebook design is to perturb codevectors by moving the training data vector from its current partitioned set to a different partitioned set. At each iteration of the simulated annealing algorithm, the perturbation is performed if and only if Eq. 5.10 is satisfied. This is called a stochastic relaxation algorithm (Zeger & Gersho, 1989; Zeger *et al.*, 1992) if the perturbation is applied by adding some values to the codevectors definitely for each iteration. The stochastic relaxation algorithm is depicted as follows:

Step 1: Select initial codevectors $C_i^{(1)}$ randomly, $i = 1, 2, ..., N$. Set iterative number $m = 1$ and $D_0 = \infty$.

Step 2: Assign the data vector X_l to partitioned set S_i if $d(X_l, C_i) \leq d(X_l, C_j)$, $i \neq j$, $j = 1, 2, ..., N$. Calculate the overall distortion D_m.

Step 3: If $\frac{|D_{m-1} - D_m|}{D_m} < \epsilon$, then terminate the program; otherwise, set $m = m + 1$.

Step 4: Compute the centroid for each partitioned set,

$$C_i = \frac{1}{|S_i|} \sum_{X_j \in S_i} X_j \tag{5.13}$$

where $|S_i|$ denotes the number of training data vectors in the partitioned set S_i.

Step 5: Perturb the codevector using

$$C_i^m = C_i^m + S_i(T_m). \tag{5.14}$$

Go to step 2.

$S_i(T_m)$ is a perturbation function in which the value of the temperature T_m decreases with the increase of the iterative number m. In previous work (Zeger & Gersho, 1989), $S_i(T_m)$ is a uniform distribution with zero-mean and T_m is the range.

5.1.6 Fuzzy C-means Clustering Algorithm

The GLA algorithm, PNN algorithm, simulated annealing method and stochastic relaxation approach in codebook design assign each training data vector to one and only one cluster, i.e., the training data vectors are partitioned into disjoint sets. However, each training data vector can be assigned a membership function indicating the degree of its "belongingness" to each cluster rather than assign it to only one cluster because some clusters are not compacted and well separated (Dunn, 1974; Bezdek, 1973). Assume that T and N are the number of training data vectors and the number of codevectors. The object function of the fuzzy C-means (FCM) clustering algorithm (Bezdek, 1973) is to

minimize

$$J(U, C) = \sum_{i=1}^{T} \sum_{j=1}^{N} u_{ij}^2 D_{ij}, \tag{5.15}$$

subject to

$$\sum_{j=1}^{N} u_{ij} = 1, \qquad 1 \le i \le T, \tag{5.16}$$

where D_{ij} is the squared Euclidean distortion between the training data vector X_i and the centroid C_j, u_{ij} is the value of membership for the training data vector X_i belonging to the cluster j, $U = \{u_{ij}\}$ is $T \times N$ matrix and $C = \{C_1, C_2, ..., C_N\}$ is the codebook.

The fundamental FCM clustering algorithm can be stated as follows:

Step 1: Set $m = 1$ and select the membership function $U(1)$. Here m is the current number of iterations.

Step 2: Calculate $C(m)$ by using

$$C_j(m) = \frac{\sum_{i=1}^{T} u_{ij}^2 X_i}{\sum_{i=1}^{T} u_{ij}^2}, \quad 1 \le j \le N. \tag{5.17}$$

Step 3: If $D_{in}(m) = 0$, then $u_{in}(m+1) = 1$ and $u_{ij}(m+1) = 0$ for $j \neq n$; otherwise, calculate $U(m+1)$ by using

$$u_{ij}(m+1) = \frac{1}{\sum_{n=1}^{N} \frac{D_{ij}(m)}{D_{in}(m)}}, \quad 1 \le i \le T, 1 \le j \le N. \tag{5.18}$$

Step 4: Terminate the program and take $C(m) = \{C_1(m), C_2(m), ..., C_N(m)\}$ as the final codebook if $max|u_{ij}(m) - u_{ij}(m+1)| < \varepsilon$, $1 \le i \le T$, $1 \le j \le N$; otherwise, set $m = m + 1$ and go to step 2. Here ε is a small predefined value.

5.1.7 Path-following Approach

The path-following approach, also known as continuation method for vector quantizer (CMVQ) design was proposed by (Chung *et al.*, 1993). Suppose that there are T input vectors in the training set $S = \{x_p | p = 1, 2, ..., T\}$ and N codevectors in the codebook $C = \{C_i | i = 1, 2, ..., N\}$, where $T \gg N$. The sum of the squared errors within the partitioned sets is

$$D(S, C) = \sum_{p=1}^{T} \sum_{X_p \epsilon S_i} d(X_p, C_i), \tag{5.19}$$

where $d(X_p, C_i)$ is the squared Euclidean distortion between X_p and C_i, $S_i = \{X_p \epsilon S | d(X_p, C_i) \le d(X_p, C_j), \forall i \neq j\}$ and $1 \le i \le N$. The centroid computation step of the GLA algorithm is found by evaluating $\frac{\partial D(S,C)}{\partial C_i} = 0$, i.e.,

$$\sum_{X_p \epsilon S_i} X_p - |S_i|.C_i = 0, \tag{5.20}$$

where $|S_i|$ is the number of training vectors belonging to the partitioned set S_i. If a training set S^N consists of only N vectors randomly chosen from S, by evaluating $\frac{\partial D(S^N,C)}{\partial C_i} = 0$, the following equation is obtained:

$$\sum_{X_p \epsilon S_i^N} X_p - |S_i^N|.C_i = 0, \tag{5.21}$$

where $S_i^N = \{X_p \epsilon S^N | d(X_p, C_i) \leq d(X_p, C_j) \forall i \neq j\}$. The homotopy function (Stonick & Alexander, 1992; Richter & DeCarlo, 1983) for VQ codebook design can be defined as

$$h(C_i, t) = (1-t)[\sum_{X_p \epsilon S_i^N} X_p - |S_i^N|.C_i] + t[\sum_{X_p \epsilon S_i} X_p - |S_i|.C_i]. \tag{5.22}$$

By setting Eq. 5.22 to zero, the iterative algorithm for the codevector is derived as

$$C_i = \frac{1}{|S_i^N| + t\{|S_i| - |S_i^N|\}}(\sum_{X_p \epsilon S_i^N} X_p + t \sum_{X_p \epsilon \{S_i - S_i^N\}} X_p). \tag{5.23}$$

The homotopy parameter t is a weighting factor in this centroid computation step. It can be set to a linear homotopy parameter sequence $\{t_n = n.\triangle t | n = 0, 1, ..., \frac{1}{\triangle t}\}$. If $n = 0$, it is the initial step as in Eq. 5.21. If $n = \frac{1}{\triangle t}$, it is the final result as in Eq. 5.20. By adapting n from 0 to $n = \frac{1}{\triangle t}$, the final codevectors are generated.

5.1.8 Deviation Reduction Algorithm

The deviation reduction (DR) algorithm was proposed by (Chen *et al.*, 1995). This algorithm generates N codevectors from the training data $X = \{X_1, X_2, ..., X_T\}$ of k-dimensional vectors with $T \gg N$. These training data are grouped into N clusters first using K-d tree with N buckets which is generated from the greatest co-ordinate variance (Bentley, 1975). Each cluster is represented by the number of data n_i belonging to this cluster and the centroid C_i of this cluster, $i = 1, 2, ..., N$. The weighted distances defined in Eq. 5.24 are calculated.

$$d_{i,j} = \frac{n_i n_j |C_i - C_j|^2}{n_i + n_j}, \tag{5.24}$$

where $i = 1, 2, ..., N$ and $j = i+1, i+2, ..., N$.

The mean of these $\frac{N(N-1)}{2}$ weighted distances among all clusters is

$$\bar{d} = \frac{2}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1}^{N} d_{i,j}. \tag{5.25}$$

The difference between the $d_{i,j}$ and $\bar{d}$ indicates the deviation of the cluster C_i and C_j from the mean. Based on the assumption that the small value of deviation leads to the better optimum, the codevector C_i can be generated iteratively by

$$C_i = C_i + \alpha(d_{i,j} - \bar{d})(C_j - C_i), \tag{5.26}$$

where $i = 1, 2, ..., N$, $j = 1, 2, ..., N$ and α is a small positive constant. Eq. 5.26 is applied to adapt codevector so as to reduce the deviation.

5.2 Codebook Design Using Genetic Algorithms

It has been shown (Lloyd, 1982) that two conditions are necessary but not sufficient for the existence of an optimal minimum mean squared error (MSE) quantizer :

(1) the codewords should be the centroids of the partitions of the vector space.

(2) the centroid is the nearest neighbour (NN) for the data vectors in the partitioned set.

These conditions have been applied to codebook design by Linde et al. in the generalized Lloyd algorithm (GLA) (Linde *et al.*, 1980). Since these conditions are necessary but not sufficient, there is no guarantee that the resulting codebook is optimal. The generalized Lloyd algorithm is widely used in codebook generation for vector quantization. It is a descent algorithm in the sense that at each iteration the average distortion is reduced. For this reason, GLA tends to get trapped in local minima. The performance of the GLA is dependent on the number of minima and on the choice of the initial conditions.

Genetic algorithms refer to a model introduced and investigated by Holland (Holland, 1975) and by students of Holland. They are computer search methods whose mechanics are based on those of natural genetics. A genetic algorithm is any population-based model that uses selection

and recombination operators to generate new sample points in a search space. This evolutionary procedure yields an effective search in a broad range of problems. Genetic algorithms (Goldberg, 1989; Davis, 1991) have been proven to be powerful methods in search, optimization and machine learning. They encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures in order to achieve optimization. Genetic algorithms have been used in VLSI layout, communication network design, medical imaging, automatic control and machine learning, facility layout problem and the optimization of generalised assignment problem.

This chapter describes the GA-GLA1 and GA-GLA2 algorithms (Pan *et al.*, 1995c; Pan *et al.*, 1996d) derived by applying genetic algorithms to codebook design to produce better optimum VQ codebook vectors. The four main steps involved in genetic algorithms are evaluation, selection, crossover and mutation. It is referred to as GA-GLA1 algorithm if the evaluation, selection and crossover are adopted in combination with GLA to produce a superior codebook design algorithm.

The fitness of genetic algorithms can be represented by the mean squared error (MSE). In the VQ operation, a chromosome is designated as the centroid of the cluster. The individual of the population is the codebook. As shown in Fig. 5.1, the proposed GA-GLA1 algorithm consists of the following steps:

Step 1: Initialization – Calculate the central chromosome (centroid) G_0 from the training vectors X_i (i=1,2,...,T). Select N chromosomes G_j (j=1,2,...,N) for every member of the population using random number generator. Here N is the codebook size, so that each codebook consists of N single-vector chromosomes. P sets of N chromosomes are generated in this step, P is the population size.

Step 2: Update – GLA is used to update N chromosomes for every member of the population.

Step 3: Evaluation – Fitness (or MSE) of every member of the population is evaluated in this step.

Step 4: Selection – The survivors of the current population are decided from the survival rate P_s. A random number generator is used to generate random numbers whose values are between 0 and 1. If the random number is smaller than P_s, this codebook survives; otherwise, it does not survive. The best fitness of the population always survives. Pairs of parents are selected from these survivors and undergo a subsequent crossover operation to produce the child chromosomes that form a new population in the next generation.

Step 5: Crossover – The chromosomes of each survivor are sorted in decreasing order according to the squared error between the chromosome G_j of the current population and the central chromosome G_0. Without sorting here, it is difficult to jump out of the local minima. The 1-point or 2-point crossover technique (Goldberg, 1989) is used to produce the next generation from the selected parents.

Step 6: Termination – Step 2 to step 5 are repeated until the predefined number of generations have been reached. After termination, the optimal codebook is generated from N chromosomes in the best member of the current population.

As shown in Fig. 5.2, the GA-GLA2 algorithm is similar to the GA-GLA1 algorithm except that the stochastic relaxation scheme is applied to the mutation step in the codebook generation. A random value is added to selected genes in the mutation step. This perturbation gives the GA-GLA2 algorithm more opportunity to jump off the local optimum. The added value of the perturbation can be a normal distribution, uniform distribution or any other possible distributions. The proposed GA-GLA2 algorithm is stated as the following steps:

Step 1: Initialization – Calculate the central chromosome (centroid) G_0 from the training vectors X_i (i=1,2,...,T). Select N chromosomes G_j (j=1,2,...,N) for every member of the population using random number generator. Here N is the codebook size, so that each codebook

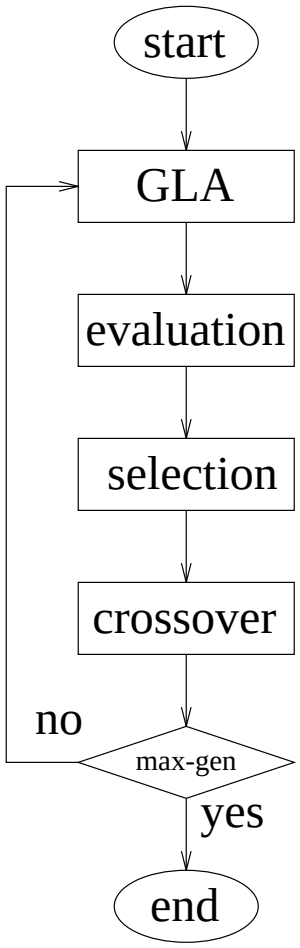

Figure 5.1: Flowchart of GA-GLA1 algorithm

consists of N single-vector chromosomes. P sets of N chromosomes are generated in this step, P is the population size.

Step 2: Update – GLA is used to update N chromosomes for every member of the population.

Step 3: Evaluation – Fitness (or MSE) of every member of the population is evaluated in this step.

Step 4: Selection – The survivors of the current population are decided from the survival rate P_s. A random number generator is used to generate random numbers whose values are between 0 and 1. If the random number is smaller than P_s, this codebook survives; otherwise, it does not survive. The best fitness of the population always survives. Pairs of

parents are selected from these survivors and undergo a subsequent crossover operation to produce the child chromosomes that form a new population in the next generation.

Step 5: Crossover – The chromosomes of each survivor are sorted in decreasing order according to the squared error between the chromosome G_j of the current population and the central chromosome G_0. The 1-point or 2-point crossover technique is used to produce the next generation from the selected parents.

Step 6: Mutation – The genes (or features) in the chromosomes of the population are mutated according to the mutation rate P_m. Here, the total number of mutations is set to population size P * number of chromosomes N * mutation rate P_m. When one chromosome is selected to be mutated from random generation number, the new genes are generated from the old genes by adding the random value θ_n. Here $1 \leq n \leq k$ and k is the number of genes in one chromosome; $-0.5\sigma_n\eta^t \leq \theta_n \leq 0.5\sigma_n\eta^t$, σ_n is the standard deviation of the nth dimension of the vector, t is the number of generations processed at present and $\eta < 1$.

Step 7: Termination – Step 2 to step 5 are repeated until the predefined number of generations have been reached. After termination, the optimal codebook is generated from N chromosomes in the best member of the current population.

Genetic algorithms have previously been applied to VQ codebook generation (Delport & Koschorreck, 1995). Although they use a genetic algorithm, it differs from the GA-GLA1 and GA-GLA2 algorithms considerably (Pan *et al.*, 1996e). The main differences are as follows.

Firstly, Delport and Koschorreck use the codebook indices of the training data as the coding string, the length of the coding string is thus the number of the training data points in the training set. In the GA-GLA1 and GA-GLA2 algorithms, the codebook vectors are used as the coding string, the length of the coding string is thus equivalent to the number of codewords. This means that the length of the coding string is much shorter in the GA-GLA1 and GA-GLA2 algorithms.

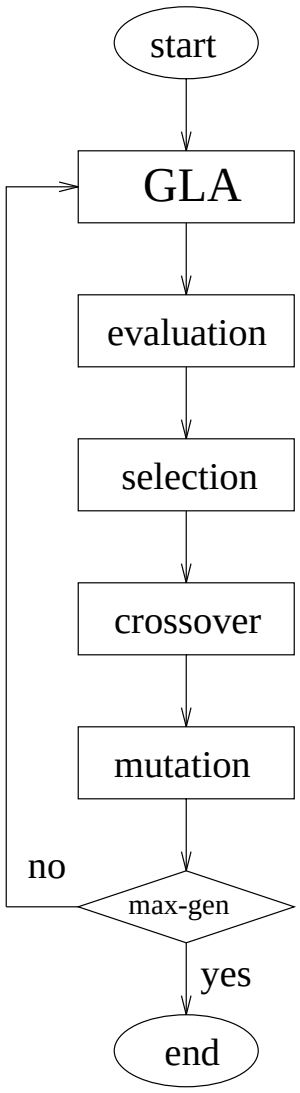

Figure 5.2: Flowchart of GA-GLA2 algorithm

Theoretically and practically, it is difficult to converge to a more optimal value if the length of the coding string is too long.

In addition, in the GA-GLA1 and GA-GLA2 algorithms, the coding strings of the initial population can be assigned randomly from the training data, because it can be converged to an optimal value easily under any initial conditions. Delport and Koschorreck use the binary splitting method to derive the best initial population to improve their algorithm.

Finally, a sorting technique is used based on the central value of the training data to facilitate

convergence to improved optima. This is unique to the GA-GLA1 and GA-GLA2 algorithms.

To sum up, as shown in Fig. 5.3 and Fig. 5.4, Delport and Koschorreck apply a genetic algorithm to adapt the codebook index of points in the training data, whereas the genetic algorithms are applied to GA-GLA1 and GA-GLA2 algorithms to adapt the value of the codebook vector.

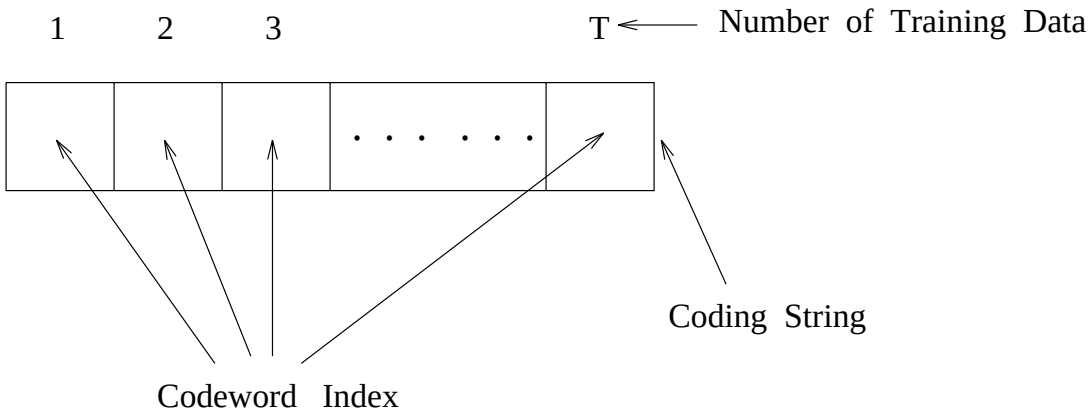

Figure 5.3: Coding String of Delport's Algorithm

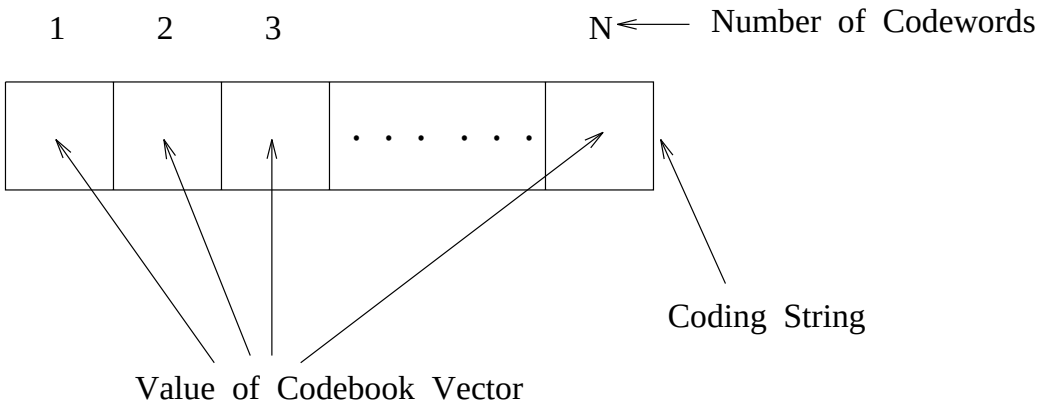

Figure 5.4: Coding String of GA-GLA1 and GA-GLA2 Algorithms

5.3 Experiments and Results

Cepstrum coefficients are used as the test features in the codebook generation experiments. The test materials for these experiments consist of 9 words recorded from one male speaker. The speech is sampled at a rate of 16 kHz and 13-dimensional cepstrum coefficients (including

energy) are computed over 20 ms-wide frames with 5 ms frame shift. A total of 817 analyzed frames are used in the codebook generation experiments.

Firstly, experiments are carried out to test the performance of GA-GLA1, GA-GLA2 and GLA algorithms for 32 and 64 codewords. 1-point and 2-point crossover techniques are tested in the experiments. The performance is measured in terms of mean squared error (MSE) and is averaged from 10 runs. The parameter values used for population size P, predefined number of generations, the survival rate P_s, the mutation rate P_m and η are 20, 100, 0.5, 0.1 and 0.9. As shown in Table 5.1, 5.2, 5.3 and 5.4, in any run, the mean squared error of the GA-GLA1 algorithm and the GA-GLA2 algorithm is smaller than the GLA algorithm. Table 5.5 and 5.6 show that the average distortions of 10 runs for codebooks generated by these new algorithms are much better than those by the GLA algorithm. The 2-point crossover technique is better than the 1-point crossover technique and the GA-GLA2 algorithm is better than the GA-GLA1 algorithm in these experiments. The mean squared error decreases by more than 9% using these new algorithms instead of the GLA algorithm.

Fig. 5.5 depicts the mean squared error (MSE) versus the population size for the GA-GLA1 algorithm using single point crossover and the number of codewords is 32. The most suitable population size is 30 which can be determined from this experiment. The mean squared error versus the number of generations for the GA-GLA1 algorithm using single point crossover and 50 individuals of population is shown in Fig. 5.6 for 32 codewords. This figure is generated from the data of only one run. The mean squared error remains constant for several generations. This means that the GLA algorithm is not useful in decreasing the mean squared error for the best population and the genetic algorithm can not perform better result for the other individuals of the population at the current generation. But after some generations, the genetic algorithm will cause the other individuals of the population to generate better codewords to decrease the mean squared error. The more generations for which this algorithm operates, the lower the mean squared error it generates but the running time will increase with the number of generations.

Another experiment tests the performance of the GA-GLA1, GA-GLA2 algorithms and the stochastic relaxation approach for generating 8 codewords. The single point crossover technique is used in this experiment. The parameter values used for population size P, predefined number of generations, the survival rate P_s, the mutation rate P_m and η are 20, 100, 0.5, 0.1 and 0.9. The results of ten runs are shown in Table 5.7. Both the GA-GLA1 and GA-GLA2 algorithms have similar performance to the stochastic relaxation approach in this experiment. The mean squared error of the global optimum is approximately 0.5832.

From the experimental results, the performance of the proposed GA-GLA1 algorithm and GA-GLA2 algorithm are significantly better than for the GLA algorithm. These new algorithms can be extended by using powerful mutation techniques, chromosome encoding techniques and the other powerful selection and crossover techniques.

GLA	0.28522	
Seed	GA-GLA1 1-point crossover	GA-GLA1 2-point crossover
1	0.2683	0.2602
2	0.2549	0.2585
3	0.2622	0.2583
4	0.2593	0.2562
5	0.2588	0.2584
6	0.2592	0.2586
7	0.2580	0.2593
8	0.2566	0.2584
9	0.2579	0.2566
10	0.2595	0.2573

Table 5.1: Mean squared errors for ten runs of GA-GLA1 algorithm and GLA for 32 codewords

GLA	0.28522	
Seed	GA-GLA2 1-point crossover	GA-GLA2 2-point crossover
1	0.2609	0.2605
2	0.2574	0.2558
3	0.2643	0.2560
4	0.2560	0.2578
5	0.2566	0.2575
6	0.2559	0.2564
7	0.2595	0.2569
8	0.2598	0.2566
9	0.2572	0.2558
10	0.2568	0.2544

Table 5.2: Mean squared errors for ten runs of GA-GLA2 algorithm and GLA for 32 codewords

GLA	0.187098	
Seed	GA-GLA1 1-point crossover	GA-GLA1 2-point crossover
1	0.171316	0.168194
2	0.171781	0.171443
3	0.169128	0.167743
4	0.172415	0.168878
5	0.168015	0.167335
6	0.174766	0.169959
7	0.172679	0.172166
8	0.171191	0.170286
9	0.171656	0.170283
10	0.171595	0.171260

Table 5.3: Mean squared errors for ten runs of GA-GLA1 algorithm and GLA for 64 codewords

GLA	0.187098	
Seed	GA-GLA1 1-point crossover	GA-GLA1 2-point crossover
1	0.170653	0.168353
2	0.168776	0.169391
3	0.169161	0.168561
4	0.171857	0.168010
5	0.170882	0.165990
6	0.166766	0.168568
7	0.169233	0.168356
8	0.171477	0.171594
9	0.173121	0.168429
10	0.168244	0.172010

Table 5.4: Mean squared errors for ten runs of GA-GLA2 algorithm and GLA for 64 codewords

Algorithm		MSE
GLA		0.28522
GA-GLA1	1-point crossover without mutation	0.25947
GA-GLA2	1-point crossover with mutation	0.25844
GA-GLA1	2-point crossover without mutation	0.25817
GA-GLA2	2-point crossover with mutation	0.25677

Table 5.5: Performance comparison of GA-GLA1, GA-GLA2 algorithms and GLA for 32 codewords

Algorithm		MSE
GLA		0.187098
GA-GLA1	1-point crossover without mutation	0.171554
GA-GLA2	1-point crossover with mutation	0.170017
GA-GLA1	2-point crossover without mutation	0.169755
GA-GLA2	2-point crossover with mutation	0.168926

Table 5.6: Performance comparison of GA-GLA1, GA-GLA2 algorithms and GLA for 64 codewords

Seed	stochastic relaxation	GA-GLA1	GA-GLA2
1	0.5834	0.5838	0.5832
2	0.5876	0.5838	0.5832
3	0.5832	0.5832	0.5832
4	0.5832	0.5832	0.5832
5	0.5832	0.5832	0.5832
6	0.5832	0.5838	0.5835
7	0.5875	0.5832	0.5832
8	0.5832	0.5839	0.5832
9	0.5832	0.5832	0.5832
10	0.5832	0.5832	0.5832

Table 5.7: Mean squared errors for ten runs of GA-GLA1, GA-GLA2 algorithms and stochastic relaxation approach for 8 codewords

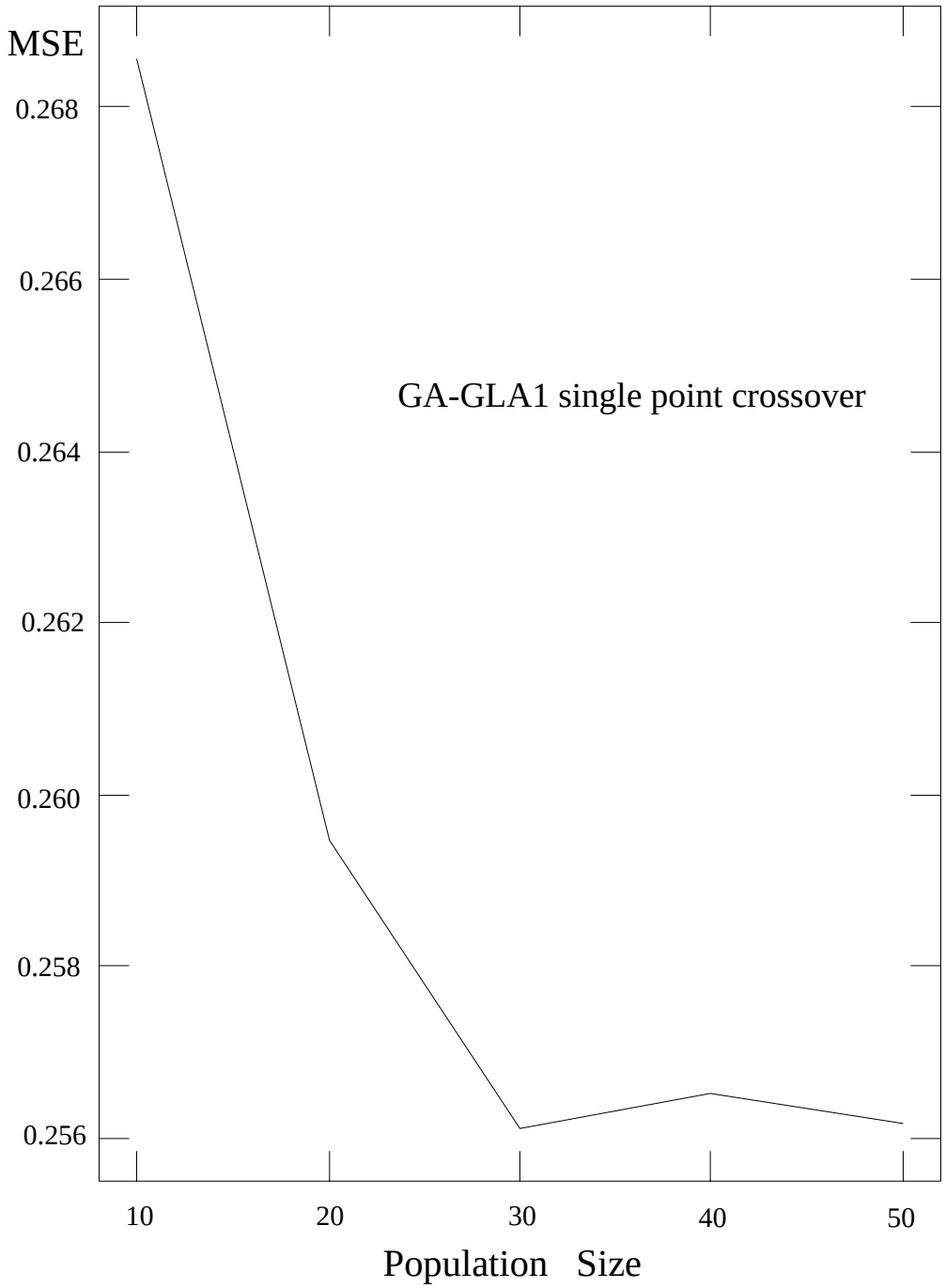

Figure 5.5: Mean squared error of GA-GLA1 algorithm for different population size

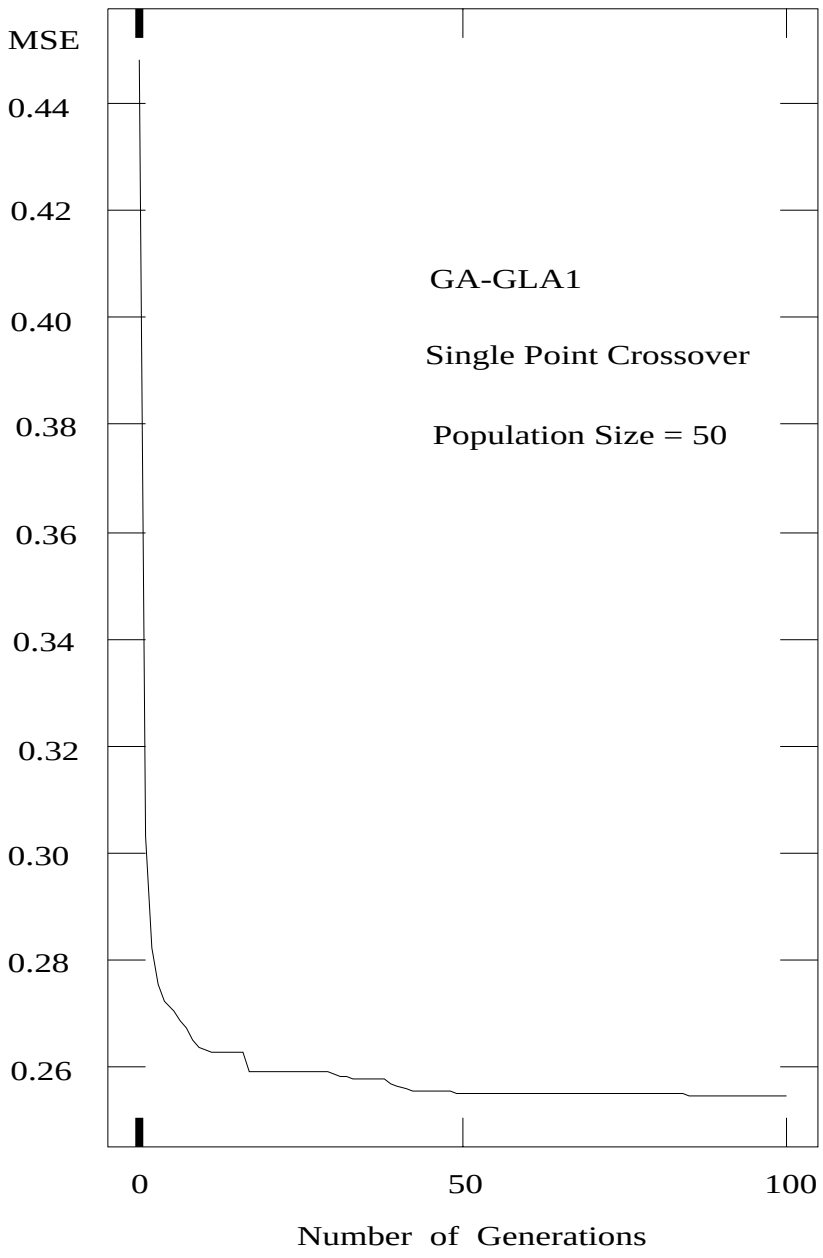

Figure 5.6: Mean squared error of GA-GLA1 algorithm for different number of generations

Chapter 6

Improved Algorithms for Codebook Index Assignment

6.1 Introduction

Vector quantization (VQ) (Gray, 1984) is very efficient for data compression of speech and images where the binary indices of the optimally chosen codevectors are sent. As shown in Fig. 6.1, a vector $X = \{x^1, x^2, ..., x^k\}$ consisting of k samples of information source in the k-dimensional Euclidean space R^k is sent to the vector quantizer. The k-dimensional vector quantizer with the number of codevectors N is defined as follows by using the reproduction alphabet consisting of N codevectors, $C = \{C_1, C_2, ..., C_N\}$, the partitioned set consisting of subspaces of the k-dimensional Euclidean space R^k, $S = \{S_1, S_2, ..., S_N\}$, and the mapping function $Q(\cdot)$:

$$Q(X) = C_i, \qquad \text{if} \quad X \epsilon S_i. \tag{6.1}$$

The sets C and the partitioned set S_i satisfy

$$\cup_{i=1}^{N} S_i = R^k. \tag{6.2}$$

and

$$S_i \cap S_j = \phi \qquad \text{if} \quad i \neq j. \tag{6.3}$$

The output of the vector quantizer is the index i of the codevector C_i which satisfies

$$i = \mathrm{argmin}_p \sum_{l=1}^{k} (x^l - c_p^l)^2. \tag{6.4}$$

Only the index i is transmitted over the channel to the receiver. The transmitting rate is defined as

$$m = \log_2 N \quad \text{bits/vector}, \tag{6.5}$$

and

$$r = m/k \quad \text{bits/sample}. \tag{6.6}$$

The performance of vector quantizer can be evaluated by the squared Euclidean distortion per symbol given by

$$D_s = \frac{1}{k} \sum_{i=1}^{N} \int_{S_i} P(X) \sum_{l=1}^{k} (x^l - c_i^l)^2 \, dX, \tag{6.7}$$

where $P(X)$ is the probability density function of X.

The channel noise will induce channel errors in the communication. The effect of channel errors is to cause errors in the received indices. Thus, distortions are introduced in the decoding step. Distortion due to an imperfect channel can be reduced by assigning suitable indices to codevectors. If the number of codevectors is N, the possible combination of indices to codevectors is $N!$. To test $N!$ assignments is an NP-hard problem.

6.1.1 Simulated Annealing for Optimization of Index Assignment

As described previously, the optimization of index assignment for vector quantizer is computationally intractable for large codebook size even if very powerful computer is used because there exist $N!$ possible ways to arrange the indices of codevectors for N codevectors. In order to avoid the full search procedure, a simulated annealing method has been applied (Kirkpatrick *et al.*, 1983; Bohachevsky *et al.*, 1986) to the codevector index assignment of vector quantizers for noisy channels (Farvardin, 1990). The channel model is assumed to be a binary symmetric channel with bit error probability ε, i.e.,

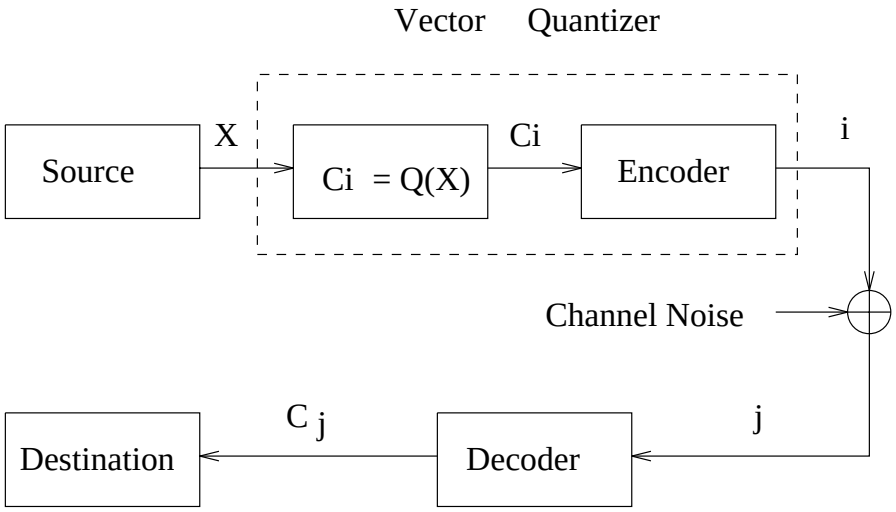

Figure 6.1: Block diagram of VQ communication system for noisy channel

$$P(b(c_j)/b(c_i)) = (1-\varepsilon)^{m-H(b(c_i),b(c_j))}\varepsilon^{H(b(c_i),b(c_j))}, \tag{6.8}$$

where $b(c_i)$, $i = 1, 2, ..., N$, is the index with m bit string of codevector C_i, $P(b(c_j)/b(c_i))$, $i, j = 1, 2, ..., N$, denote the probability that index $b(c_j)$ is received given the index $b(c_i)$ is sent and $H(b(c_i), b(c_j))$ denote the Hamming distance between $b(c_i)$ and $b(c_j)$.

In this previous work (Farvardin, 1990), the channel bit error probability ε is assumed to be sufficiently small ($m\varepsilon \ll 1$), then the error probability due to more than one bit error can be ignored and the bit error probability of the channel model can be expressed as

$$P(b(c_j)/b(c_i)) = \begin{cases} \varepsilon, & H(b(c_i), b(c_j)) = 1 \\ 1 - m\varepsilon, & H(b(c_i), b(c_j)) = 0 \\ 0, & H(b(c_i), b(c_j)) > 1 \end{cases} \tag{6.9}$$

Based on this channel model, the average distortion per source sample caused by the channel noise for a given assignment of indices, $b = (b(c_1), b(c_2), ..., b(c_N))$, can be expressed as

$$D_{cs} = \frac{1}{k} \sum_{i=1}^{N} \sum_{j=1}^{N} P(c_i) P(b(c_j)/b(c_i)) d(c_i, c_j) \tag{6.10}$$

$$= \frac{\varepsilon}{k} \sum_{i=1}^{N} P(c_i) \sum_{j:H(b(c_i),b(c_j))=1} d(c_i, c_j), \tag{6.11}$$

and the ensemble average distortion is derived as

$$\bar{D}_{cs} = \frac{1}{N!} \cdot \frac{\varepsilon}{k} \sum_{i=1}^{N} P(c_i) \sum_{all\, b} \sum_{j:H(b(c_i),b(c_j))=1} d(c_i, c_j) \tag{6.12}$$

$$= \frac{\varepsilon m}{k(N-1)} \sum_{i=1}^{N} P(c_i) \sum_{j=1}^{N} d(c_i, c_j). \tag{6.13}$$

The algorithm using simulated annealing for optimization of index assignment for vector quantizer can be stated as follows (Farvardin, 1990):

Step 1: Choose an initial state b of the indices for codevectors at random and set the initial temperature $T = T_0$.

Step 2: Randomly choose another state b' (perturbation of state b) and calculate $\delta D_{cs} = D_{cs}(b') - D_{cs}(b)$. If $\delta D_{cs} < 0$, replace b by b'; otherwise, replace b by b' with probability $e^{\frac{-\delta D_{cs}}{T}}$ and go to step 3.

Step 3: If the number of average distortion drops exceeds a prescribed number or if too many unsuccessful perturbations occur, go to step 4.

Step 4: Terminate the program if the temperature T is below some prescribed freezing temperature T_f or a stable state is reached; otherwise, lower the temperature T and go to step 2.

Note that the simulated annealing approach in the optimization of codebook index assignment will not affect the quantization accuracy in the error-free case because this method does not change the value of codevectors.

6.1.2 Pseudo-Gray Coding

Pseudo-Gray coding (Zeger & Gersho, 1990; Zeger & Gersho, 1987) provides a redundancy-free error protection scheme for vector quantization of analogue signals when the binary indices of the signal codevectors are sent on a discrete memoryless channel. The main idea of Pseudo-Gray coding is to calculate the expected distortion due to the single bit error in the index of codevectors for every index swapping and swap the index pair that makes the largest improvement in distortion. The approach of Pseudo-Gray coding is stated as follows:

Step 1: Initialization – Assign indices to the codevectors randomly.

Step 2: Sorting – Sort the codevectors in the decreasing order of the expected distortion. Set $i = -1$.

Step 3: Distortion Reduction – Set $i = i + 1$. For $j = i + 1$ to N, calculate the distortion reduction after swapping the index i and j. Let gain = the maximum distortion reduction for swapping the index i and j.

Step 4: Switching – If gain > 0, then switch index i and j and go to step 2.

Step 5: Termination – If $i = N - 1$, then terminate the program; otherwise, go to step 3.

6.1.3 Channel Optimized Vector Quantization

An improvement of vector quantizer performance against channel noise can be achieved by taking the error characteristics of the transmission channel into account in the codebook design as well as in the quantization process. Hence, the expected error of the reconstructed codevector in the decoder can be minimized instead of the irrelevant quantization error in the encoder (Kumazawa *et al.*, 1984; Farvardin, 1990; Balss *et al.*, 1995). If the squared Euclidean distortion is used, the performance can be evaluated using Eq. 6.14 which is the expected squared error per sample in the decoder provided that index $b(c_j)$ is received given that index $b(c_i)$ is sent.

$$D_e = \frac{1}{k} \sum_{i=1}^{N} \sum_{j=1}^{N} P(b(c_j)/b(c_i)) \int_{S_i} P(X) \sum_{l=1}^{k} (x^l - c_j^l)^2 dX, \tag{6.14}$$

where $P(X)$ is the probability density function of X.

In codebook generation for channel optimized vector quantization, the indices as well as the codevectors are modified together. The index of the training data vector X is assigned using

$$i^* = \mathrm{argmin}_i \sum_{j=1}^{N} P(b(c_j)/b(c_i)) \sum_{l=1}^{k} (x^l - c_j^l)^2, \tag{6.15}$$

and the codevector is modified using

$$C_j = \frac{\sum_{i=1}^{N} P(b(c_j)/b(c_i)) \sum_{l:X_l \in S_i} X_l}{\sum_{i=1}^{N} P(b(c_j)/b(c_i))|S_i|}, \tag{6.16}$$

where $|S_i|$ is the number of training data vectors belonging to the partitioned set S_i.

When the codevector $C = \{C_1, C_2, ..., C_N\}$ is given, the partitioned set S_l minimizing Eq. 6.14 is given by

$$S_l = \{X | \sum_{j=1}^{N} P(b(c_j)/b(c_l)) \sum_{m=1}^{k} (x^m - c_j^m)^2 \leq \sum_{j=1}^{N} P(b(c_j)/b(c_i)) \sum_{m=1}^{k} (x^m - c_j^m)^2, \text{ for all } i \neq l\}. \tag{6.17}$$

When the codevectors $C_j, j = 1, 2, ..., N$, partitioned sets $S_j, j = 1, 2, ..., N$ and the training data vectors $X_t, t = 1, 2, ..., T$ are given, the mean squared error per sample in the decoder can be evaluated as follows:

$$D_e = \frac{1}{kT} \sum_{t=1}^{T} \sum_{j=1}^{N} P(b(c_j)/b(c_{i(t)})) \sum_{l=1}^{k} (x_t^l - c_j^l)^2, \tag{6.18}$$

where $b(c_{i(t)})$ is the index of the bit string to which the tth training data vector X_t belongs and T is the total number of training data vectors.

In the training process, Eq. 6.16 and Eq. 6.17 are applied iteratively until a termination criterion is met.

6.2 Average Distortion

N codevectors C_i, $i = 1, 2, ..., N$, are assigned codevector indices with an m bit string $b(c_i)$, where $N = 2^m$. The distortion between codevector C_i and C_j is given by a non-negative distortion measure $d(c_i, c_j)$. Usually, the Euclidean metric is used. Let $P(b(c_j)/b(c_i))$, $i, j = 1, 2, ...N$, denote the probability that the index $b(c_j)$ is received given the index $b(c_i)$ is sent. Assuming random assignment of the codevector indices $b = (b(c_1), b(c_2), ..., b(c_N))$, the average distortion for any possible bit errors caused by the channel noise is given by

$$D_c = \frac{1}{N!} \sum_{i=1}^{N} P(c_i) \sum_{b} \sum_{j=1}^{N} P(b(c_j)/b(c_i)) d(c_i, c_j). \tag{6.19}$$

We assume that the channel is a memoryless binary symmetric channel with bit error probability ε. Thus, the error probability is $\varepsilon^l(1 - \varepsilon)^{m-l}$, where l is the number of bits in which $b(c_i)$ and $b(c_j)$ differ. Let $H(b(c_i), b(c_j))$ denote the Hamming distance between $b(c_i)$ and $b(c_j)$. The average distortion can be written as

$$D_c = \frac{1}{N!} \sum_{i=1}^{N} P(c_i) \sum_{j=1}^{N} d(c_i, c_j) \sum_{l=1}^{m} \sum_{b: H(b(c_i), b(c_j))=l} \varepsilon^l(1 - \varepsilon)^{m-l}. \tag{6.20}$$

There are N choices of $b(c_i)$, $\begin{pmatrix} m \\ l \end{pmatrix}$ choices of $b(c_j)$ and $(N - 2)!$ choices of the rest of b given l. Eq. 6.20 can be expressed as

$$D_c = \frac{1}{N!} \sum_{i=1}^{N} P(c_i) \sum_{j=1}^{N} d(c_i, c_j) \sum_{l=1}^{m} \varepsilon^l(1 - \varepsilon)^{m-l} N \begin{pmatrix} m \\ l \end{pmatrix} (N - 2)! \tag{6.21}$$

$$= \frac{1}{N - 1} \sum_{i=1}^{N} P(c_i) \sum_{j=1}^{N} d(c_i, c_j) \sum_{l=1}^{m} \begin{pmatrix} m \\ l \end{pmatrix} \varepsilon^l(1 - \varepsilon)^{m-l}, \tag{6.22}$$

Since $1 = \{\varepsilon + (1-\varepsilon)\}^m = \sum_{l=0}^{m} \binom{m}{l} \varepsilon^l (1-\varepsilon)^{m-l}$,

$$D_c = \frac{1}{N-1} \sum_{i=1}^{N} P(c_i) \sum_{j=1}^{N} d(c_i, c_j)\left(1 - \binom{m}{0} \varepsilon^0 (1-\varepsilon)^m\right) \tag{6.23}$$

$$= \frac{1-(1-\varepsilon)^m}{N-1} \sum_{i=1}^{N} P(c_i) \sum_{j=1}^{N} d(c_i, c_j) \tag{6.24}$$

After the indices are assigned to the codevectors, the expectation of distortion for the transmission of indices $b(c_i)$, $i = 1, 2, ..., N$, can be written as

$$D = \sum_{i=1}^{N} P(c_i) \sum_{l=1}^{m} \varepsilon^l (1-\varepsilon)^{m-l} \sum_{b(c_j) \in N^l(b(c_i))} d(c_i, c_j) \tag{6.25}$$

where $N^l(b(c_i)) = \{b(c_j) \in I, H(b(c_i), b(c_j)) = l\}$, is the lth neighbour set of $b(c_i)$.

6.3 Multiple Global Optima

Assume $f(c_i) = b_i = (b_{i1}, b_{i2}, ..., b_{im})$ is the function of index assignment. Here $b_{ij} \in \{0, 1\}$, $i = 1, 2, ..., N$, $j = 1, 2, ..., m$. If f is globally optimal, then so is g defined by

$$g(c_i) = (a_{i1}, a_{i2}, ..., a_{im}), \tag{6.26}$$

where

$a_{ij} = b_{ip(j)} \oplus q_{ip(j)}$,

$q_{ij} \in \{0, 1\}$,

p is a permutation of $\{1, 2, ..., m\}$.

There are 2^m possibilities for q_{ij}, $j = 1, 2, ..., m$, and $m!$ possibilities for p. Thus, at least $m!N$ global optima exist for the problem of codebook index assignment. So, an $N!$ search space can

be reduced to an $\frac{(N-1)!}{m!}$ search space. If the number of codevectors is 8 and the globally optimal assignment of the codevector indices $b = (000, 001, 010, 011, 100, 101, 110, 111)$, then there are 8 possible combinations for q_{ij}, $j = 1, 2, 3$, i.e., $H(b(c_i), b(c_l)) = H(b(c_i) \oplus s, b(c_l) \oplus s)$, $i = 1, 2, ..., 8$, $l = 1, 2, ..., 8$ and $s \epsilon \{000, 001, 010, 011, 100, 101, 110, 111\}$ which is depicted in Table 6.1. There are also 6 possibilities of using a permutation in the bit string for each possible combination in Table 6.1. Two examples are shown in Table 6.2 and Table 6.3. This property can be applied to algorithms for codebook index assignment, for example, by setting one index to one codevector at the initial step, and holding this codevector index assignment until the termination of these algorithms, i.e., reduce the search space from $N!$ to $(N-1)!$ without reducing the possibility of producing a better optimum.

globally optimal indices	000	001	010	011	100	101	110	111
$\oplus$ 000	000	001	010	011	100	101	110	111
$\oplus$ 001	001	000	011	010	101	100	111	110
$\oplus$ 010	010	011	000	001	110	111	100	101
$\oplus$ 011	011	010	001	000	111	110	101	100
$\oplus$ 100	100	101	110	111	000	001	010	011
$\oplus$ 101	101	100	111	110	001	000	011	010
$\oplus$ 110	110	111	100	101	010	011	000	001
$\oplus$ 111	111	110	101	100	011	010	001	000

Table 6.1: Example of 2^3 possibilities for q_{ij}, $j = 1, 2, 3$, $i = 1, 2, ..., 8$

bit position	globally optimal indices							
123	000	100	010	110	001	101	011	111
132	000	010	100	110	001	011	101	111
213	000	100	001	101	010	110	011	111
231	000	010	001	011	100	110	101	111
312	000	001	100	101	010	011	110	111
321	000	001	010	011	100	101	110	111

Table 6.2: Example of 3! possibilities for the permutation of bit strings b=(000, 001, 010, 011, 100, 101, 110, 111)

bit position	globally optimal indices							
123	100	000	110	010	101	001	111	011
132	010	000	110	100	011	001	111	101
213	100	000	101	001	110	010	111	011
231	010	000	011	001	110	100	111	101
312	001	000	101	100	011	010	111	110
321	001	000	011	010	101	100	111	110

Table 6.3: Example of 3! possibilities for the permutation of bit strings b=(001, 000, 011, 010, 101, 100, 111, 110)

6.4 Algorithm

Genetic algorithms (Holland, 1975; Goldberg, 1989; Fang, 1994) are adaptive methods which can be used in search and optimization problems. Here, a parallel genetic algorithm (Cohoon *et al.*, 1987; Pettey *et al.*, 1987; Shonkwiler, 1993) is used to optimize the codevector index assignment. The $\mathtt{fitness}$ is the expectation of distortion as in Eq. 6.25. The $\mathtt{chromosome}$ is the index string. The proposed algorithm consists of the following steps (Pan *et al.*, 1996a):

Step 1: Initialization – Randomly assign the indices (i.e. 0 to $N-1$) to every individual of the population. A $\mathtt{chromosome}$ is composed of N indices. Separate the population into G groups. G sets of P members are generated in this step, where P is the population size for each group. Without loss of generality, set $G = 2^n$.

Step 2: Evaluation – The $\mathtt{fitness}$ of every individual of the population in each group is evaluated in this step.

Step 3: Communication – Send the top best B individuals of the $\mathtt{jth}$ group to the $\mathtt{qth}$ groups to substitute B individuals in each receiving group randomly for every R generations, i.e., receive some information from the other groups but keep the same population size. Here, $q = j \oplus 2^i$, $j = 0, 1, ..., G-1$ and $i = 0, 1, ..., n-1$.

Step 4: Selection – Set the number of survivors within each group to $P * P_s$ where P_s is the survival rate. For $r = 1$ to $P * P_s$, randomly choose M individuals from the group and

select the best of these M individuals as a survivor. This selection scheme is also used in the Crossover step and Mutation step to select parents and candidates for crossover and mutation.

Step 5: Crossover – The uniform order-based crossover technique (Davis, 1991) is used to produce the next generation from the selected parents for each group. $P * P_c$ individuals for each group are generated in this step, where P_c is the crossover rate. Several *gene* positions of the *chromosome* are chosen randomly and the order in which these *genes* appear in the first parent is imposed on the second parent to produce offspring. The *genes* in the other positions are the same as the first parent.

Step 6: Mutation – The *genes* (or indices) in the *chromosomes* of the population are mutated according to the mutation rate P_m. Here, the total number of mutations for each group is set to group population size P * mutation rate P_m. The mutation is only operated by exchanging two indices randomly in each group. Here, $P_s + P_c + P_m = 1$.

Step 7: Termination – Step 2 to step 6 are repeated until the predefined *fitness* or the number of generations have been reached. After termination, the optimal codevector indices are generated from the best individual for all groups.

6.5 Experimental Results

The test materials for these experiments consisted of 200 words recorded from one male speaker. The speech is sampled at a rate of 16 kHz and 13-dimensional cepstrum coefficients (including energy) are computed over 20 ms-wide frames with 5 ms frame shift. A total of 20,030 analyzed frames are used to generate 8, 16, 32 and 64 codevectors for the experiments of codevector index assignment.

Experimentthe were carried out to test the performance of the new algorithm and the average distortion of the random assignment for 8, 16, 32 and 64 codevectors. The performance is measured in terms of the average distortion using Eq. 6.25 compared with the average distortion

of the random assignment for any bit error using Eq. 6.24. The inverse of the average distortion is used as the fitness in this new algorithm to test the worst case of the random assignment. The distribution of the codevector probability is set to a uniform distribution. The parameter values used for the group population size P, the number of groups G, the predefined number of generations, the survival rate P_s, the crossover rate P_c, the mutation rate P_m, the number of individuals for selection M, the number of top best for communication B and the number of generations for communication R are 50, 8, 500, 0.5, 0.4, 0.1, 3, 1 and 50 respectively. Table 6.4 shows the average distortion with 0.01 bit error probability for 10 runs. The new algorithm reduces the distortion by more than 59 % compared with the random assignment and better than 75 % compared with the worst case for 64 codevectors and 0.01 bit error probability. For 0.1 bit error probability, the average distortion for 10 runs is shown in Table 6.5. The new algorithm reduces the distortion by more than 51 % compared with the random assignment and better than 66 % compared with the worst case for 64 codevectors and 0.1 bit error probability. The detail results of this algorithm for 0.01 and 0.1 bit error probability are depicted in Table 6.6, 6.7, 6.8, 6.9, 6.10, 6.11, 6.12 and 6.13.

The experimental results of the parallel genetic algorithm in codebook index assignment for different population sizes are shown in Fig. 6.2. The average distortion decreases with increase in the population size. This result is reasonable because for more individuals, the parallel genetic algorithm will provide more possible solutions. All previous algorithms (Marca & Jayant, 1987; Vaisey & Gersho, 1988; Farvardin, 1990; Zeger & Gersho, 1990; Zeger & Gersho, 1987) are simulated on the assumption of single bit error. One of the contributions in this chapter is the derivation of the average distortion for any bit error and the application of the parallel genetic algorithm to codebook index assignment for any bit error. Experimental results for the bit error probability from 0.01 to 0.3 for 32 codewords are depicted in Fig. 6.3.

The spirit of the parallel genetic algorithm is not only to accelerate the speed of running time, but also to produce improved index assignments. In order to reach these objectives, the

communication between groups should be operated for some fixed generations. By sending some top best individuals in the current group to the neighbouring groups, the problem of being trapped in the local optimum due to convergence in an earlier generation can be avoided because some promising individuals are migrated from the other groups to replace some worse individuals in the current group. Experiments have also been carried out to test performance in the separation of the groups. The number of possible solutions that the parallel genetic algorithm provides is $P \times G \times N_g$, where P, G and N_g are the group population size, the number of groups and the number of generations, respectively. The comparisons in the performance of the separating groups are based on the same total number of possible solutions, i.e., $P \times G \times N_g$ is kept constant. The total number of individuals of the population are separated into 8 groups, 4 groups, 2 groups and 1 group (standard genetic algorithm) and the group population sizes are 50, 100, 200 and 400, respectively. The other parameter values used for the predefined number of generations N_g, the bit error probability, the survival rate P_s, the crossover rate P_c, the mutation rate P_m, the number of individuals for selection M, the number of top best for communication B and the number of generations for communication R are 500, 0.01, 0.5, 0.4, 0.1, 3, 1 and 50 respectively. The experimental results for 32 codewords are shown in Fig. 6.4, the more groups are used, the better result is generated.

From the performance noted in these experiments, the proposed algorithm is an effective means for assigning codevector indices for noisy channels. The property of multiple global optima can also be employed to reduce the search space for codevector index assignment of the memoryless binary symmetric channel. Furthermore, the average distortion of random assignment for any bit error is also introduced in this chapter.

Number of codevectors	New algorithm	Random assignment	Worst case
8	0.05019	0.08091	0.11805
16	0.05518	0.10667	0.17194
32	0.05735	0.12900	0.21675
64	0.06370	0.15695	0.26712

Table 6.4: Performance (MSE) comparison of new algorithm, random assignment and worst case (bit error rate: 0.01)

Number of codevectors	New algorithm	Random assignment	Worst case
8	0.49695	0.73824	1.00987
16	0.54219	0.93094	1.35419
32	0.56014	1.07788	1.57570
64	0.60435	1.25663	1.80073

Table 6.5: Performance (MSE) comparison of new algorithm, random assignment and worst case (bit error rate: 0.1)

random	0.08091	
Seed	Parallel Genetic Algorithm	Worst Case
1	0.50194	0.118052
2	0.50194	0.118052
3	0.50194	0.118052
4	0.50194	0.118052
5	0.50194	0.118052
6	0.50194	0.118052
7	0.50194	0.118052
8	0.50194	0.118052
9	0.50194	0.118052
10	0.50194	0.118052

Table 6.6: Mean squared errors for ten runs of the new algorithm and the worst case for 8 codewords (error bit rate: 0.01)

random	0.10667	
Seed	Parallel Genetic Algorithm	Worst Case
1	0.055510	0.171944
2	0.054983	0.171944
3	0.054983	0.171944
4	0.055472	0.171944
5	0.054983	0.171944
6	0.054983	0.171944
7	0.055472	0.171944
8	0.055472	0.171944
9	0.054983	0.171944
10	0.054983	0.171944

Table 6.7: Mean squared errors for ten runs of the new algorithm and the worst case for 16 codewords (error bit rate: 0.01)

random	0.12900	
Seed	Parallel Genetic Algorithm	Worst Case
1	0.057117	0.216169
2	0.057323	0.216480
3	0.057218	0.216682
4	0.057010	0.216950
5	0.057371	0.217128
6	0.057040	0.216832
7	0.057020	0.217046
8	0.058332	0.217064
9	0.057118	0.216629
10	0.057901	0.216559

Table 6.8: Mean squared errors for ten runs of the new algorithm and the worst case for 32 codewords (error bit rate: 0.01)

random	0.15695	
Seed	Parallel Genetic Algorithm	Worst Case
1	0.063189	0.267354
2	0.064277	0.266953
3	0.064599	0.267502
4	0.063936	0.267080
5	0.064027	0.266898
6	0.063622	0.267707
7	0.063165	0.267259
8	0.063798	0.265945
9	0.063265	0.267465
10	0.063097	0.266987

Table 6.9: Mean squared errors for ten runs of the new algorithm and the worst case for 64 codewords (error bit rate: 0.01)

random	0.73824	
Seed	Parallel Genetic Algorithm	Worst Case
1	0.496949	1.009870
2	0.496949	1.009870
3	0.496949	1.009870
4	0.496949	1.009870
5	0.496949	1.009870
6	0.496949	1.009870
7	0.496949	1.009870
8	0.496949	1.009870
9	0.496949	1.009870
10	0.496949	1.009870

Table 6.10: Mean squared errors for ten runs of the new algorithm and the worst case for 8 codewords (error bit rate: 0.1)

random	0.93094	
Seed	Parallel Genetic Algorithm	Worst Case
1	0.540761	1.354326
2	0.544333	1.354326
3	0.540761	1.354326
4	0.540761	1.354326
5	0.540761	1.354326
6	0.544333	1.354326
7	0.544333	1.354326
8	0.544333	1.352996
9	0.540761	1.354326
10	0.540761	1.354326

Table 6.11: Mean squared errors for ten runs of the new algorithm and the worst case for 16 codewords (error bit rate: 0.1)

random	1.07788	
Seed	Parallel Genetic Algorithm	Worst Case
1	0.561347	1.572520
2	0.558456	1.575521
3	0.559565	1.577115
4	0.562493	1.575976
5	0.558365	1.573213
6	0.564351	1.574178
7	0.560843	1.576259
8	0.557643	1.578823
9	0.561906	1.578223
10	0.556460	1.575162

Table 6.12: Mean squared errors for ten runs of the new algorithm and the worst case for 32 codewords (error bit rate: 0.1)

random	1.25663	
Seed	Parallel Genetic Algorithm	Worst Case
1	0.602386	1.799010
2	0.601041	1.798727
3	0.606692	1.797925
4	0.606469	1.804297
5	0.600100	1.801423
6	0.604978	1.802993
7	0.603606	1.802993
8	0.606954	1.801774
9	0.604795	1.798162
10	0.606492	1.799980

Table 6.13: Mean squared errors for ten runs of the new algorithm and the worst case for 64 codewords (error bit rate: 0.1)

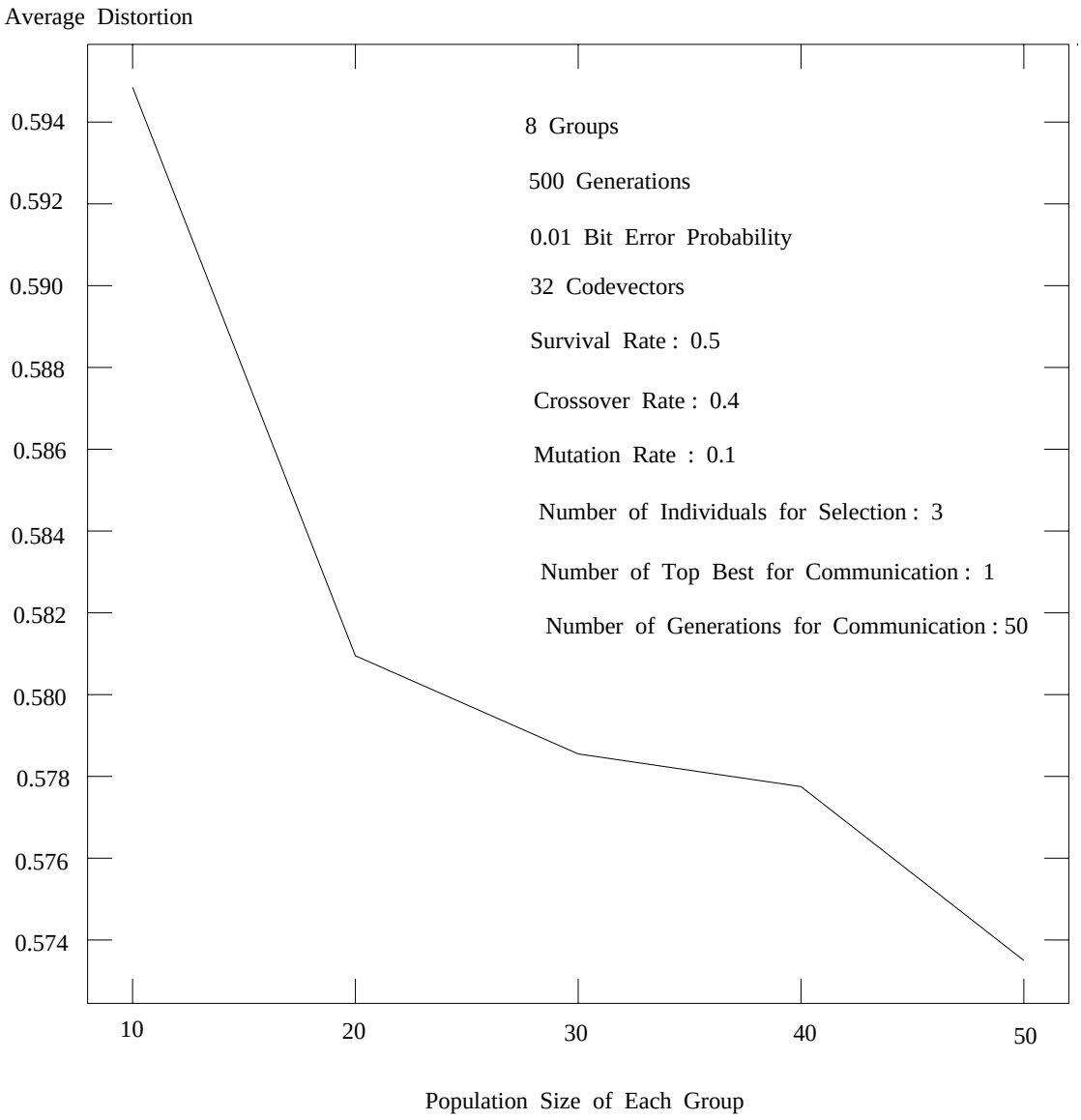

Figure 6.2: Average distortion of parallel genetic algorithm in codebook index assignment for different population size

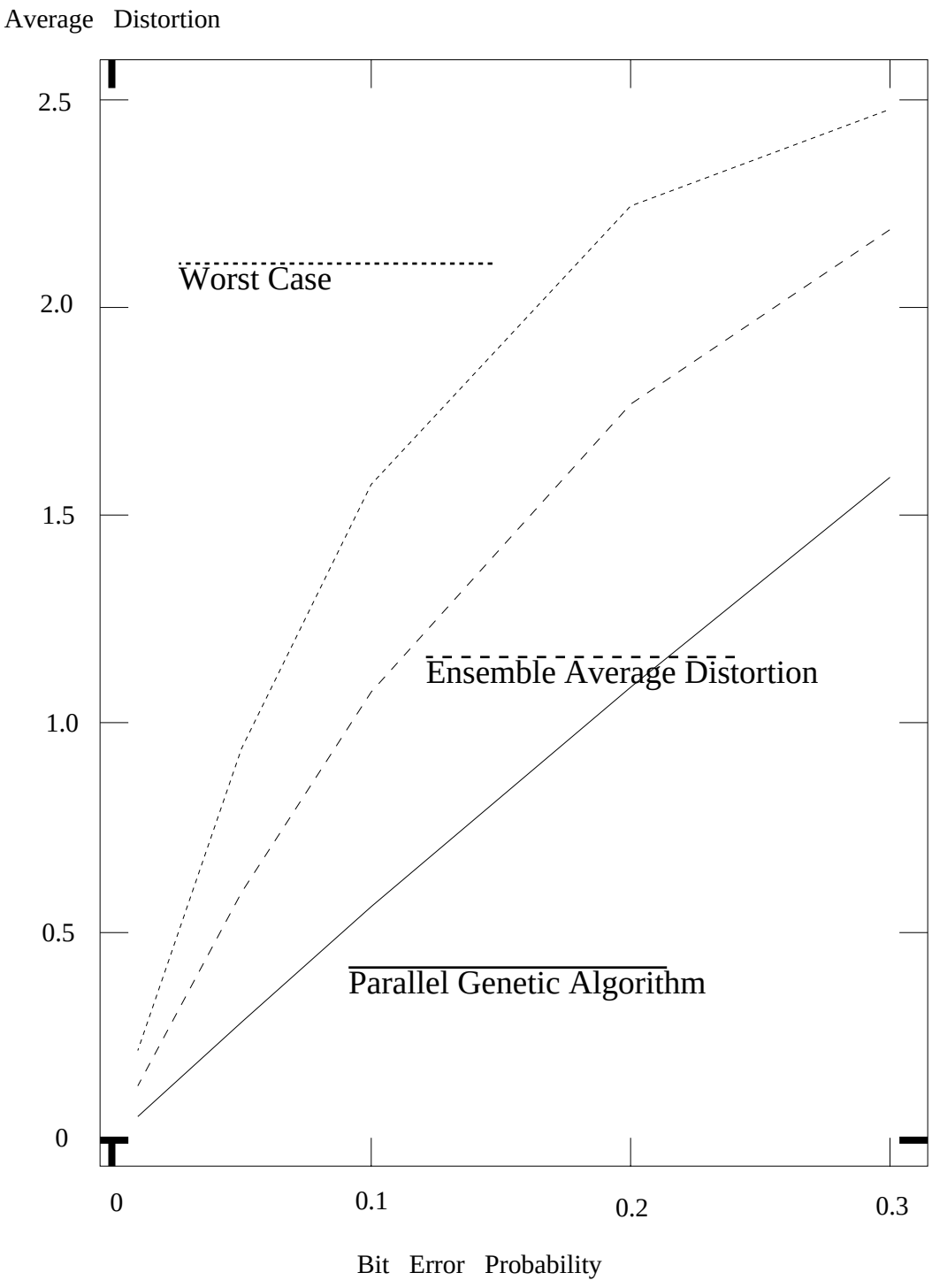

Figure 6.3: Average distortion of parallel genetic algorithm in codebook index assignment for different bit error probability

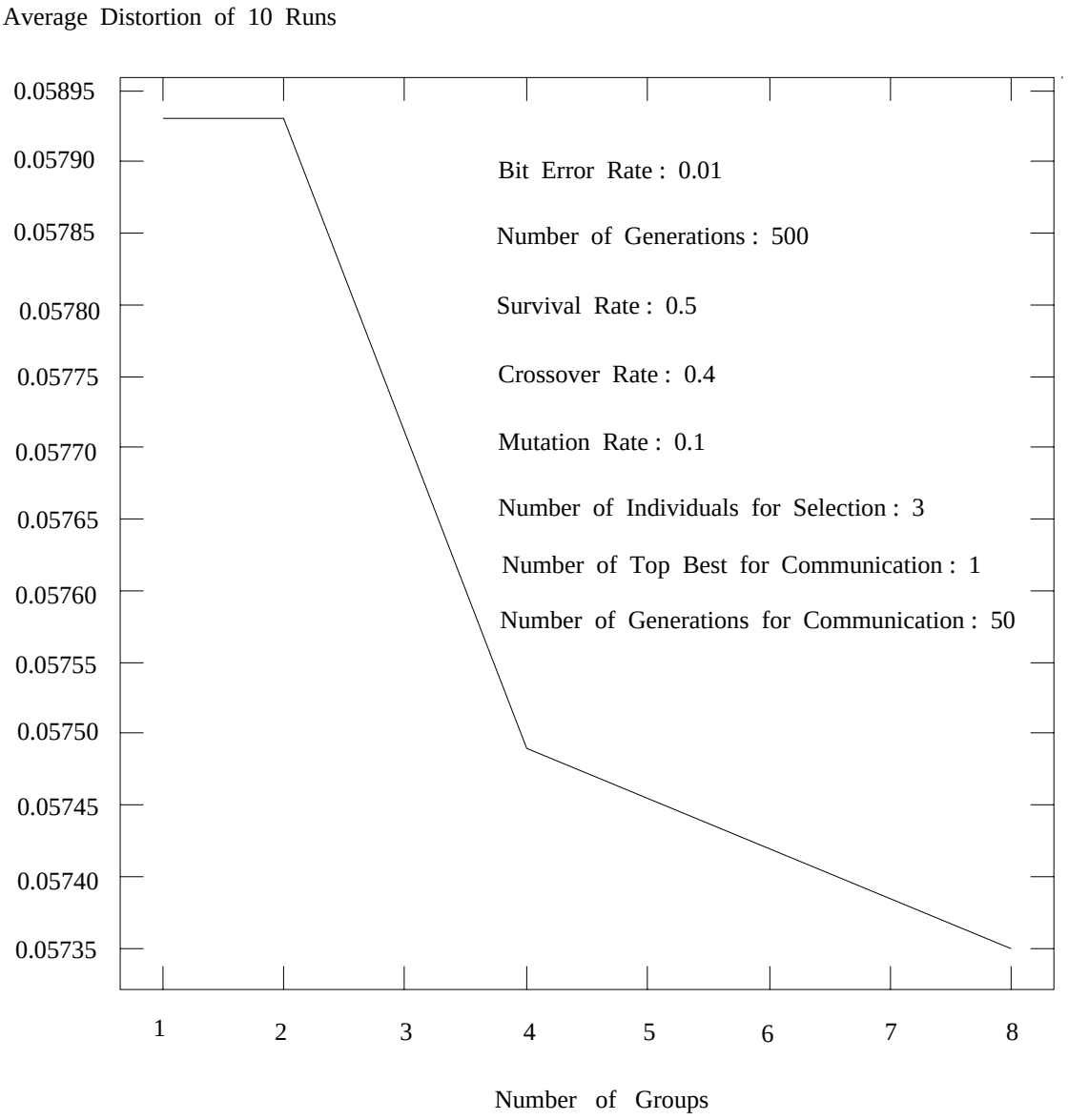

Figure 6.4: Average distortion of parallel genetic algorithm in codebook index assignment for different number of groups

Chapter 7

Summary and Conclusions

7.1 Summary

This thesis can be separated into four topics concerning fast VQ codeword search algorithms, efficient VQ clustering algorithms, improved codebook design algorithms and improved algorithms in VQ codebook index assignment for noisy channels.

In Chapter 3, several fast codeword search algorithms are proposed, such as improved algorithms combining the minimax method and the improved absolute error inequality (IAEI) criterion; improved algorithms for partial distortion search; improved algorithms for extended partial distortion search; fast approximate search algorithm; and an improved algorithm for the mean-distance-ordered search algorithm (MPS) for VQ image coding.

Several fast clustering algorithms for vector quantization are presented in Chapter 4. All these approaches based on the LBG algorithm are compared. From the experiments, the IPC-type clustering algorithm is confirmed to be the most suitable algorithm for the general processors in which the operation of the multiplication is more expensive than the operation of comparison and the TPC-type clustering algorithm is recommended for use with DSP chips in which the operation of comparison is computationally expensive.

In Chapter 5, genetic algorithms are applied to the generation of codevectors. The approach of stochastic relaxation is also combined with the genetic algorithms and the GLA algorithm to

further enhance the search ability of the genetic algorithm in codebook design.

Chapter 6 describes the importance of codebook index assignment for noisy channels and the problem that codebook index assignment is an NP-hard problem. In order to derive improved assignment in the codebook index, a parallel genetic algorithm is demonstrated. Furthermore, the ensemble average distortion with any bit error is derived and the property of multiple global optimal in codebook index assignment is highlighted.

7.2 Conclusions

7.2.1 Efficient Codeword Search Algorithms

Vector quantization has been applied to data compression of speech and images, the coding of speech and images, speech recognition and speech synthesis. The response time of codeword search for vector quantization is a very important factor to be considered for real-time applications. However, the complexity of vector quantization increases exponentially with the bit rate per dimension and the number of dimensions. This limits the application of vector quantization. In order to reduce the computation time, several efficient algorithms for VQ codeword search have been demonstrated.

The bound for Minkowski metric is derived in this thesis. By setting the parameters, this bound can generate the hypercube approach, the partial distortion search (PDS) algorithm, the absolute error inequality criterion (AEI) and the improved absolute error inequality criterion (IAEI) etc. For the Minkowski metric of order n, this bound contributes the elimination criterion from L_1 metric to L_n metric. The bound for Minkowski metric is also extended to the bounds for quadratic metric by using the methods of Karhunen-Loêve transform (KLT) and Triangular Matrix. The bounds for quadratic metric can be applied to the HMM with Gaussian mixture probability density function.

By combining the improved absolute error inequality criterion with the minimax method, several new algorithms are presented. Among these algorithms, the best algorithm will reduce the

number of multiplications by more than 77% and slightly reduce the total number of mathematical operations for 1024 codewords compared with the minimax method. Since the operation of multiplication is far more expensive than the operation of addition or comparison for the general processors, experimental results confirm this new criterion. From many experiments in the literature (Huang *et al.*, 1992; Soleymani & Morgera, 1987b; Soleymani & Morgera, 1989), the absolute error inequality criterion (AEI) is the most efficient criterion in reducing the number of multiplications for a full search algorithm. By comparing the improved absolute error inequality criterion ($IAEI$) with the absolute error inequality criterion (AEI) from theory, the $IAEI$ criterion provides a tighter bound than AEI criterion. From experiments in subsection 3.2.3, the $IAEI$ criterion is shown to reduce the number of multiplications by more than 21% and better than 3% for the total number of mathematical operations compared with the AEI criterion.

The distortion computation of the quadratic metric dominates the computation time in searching the nearest codeword for evaluating the log likelihood of Gaussian mixture distribution in the hidden Markov model with the Gaussian mixture VQ codebook probability density function. The quadratic metric is also popular in clustering algorithms. Unfortunately, the computational complexity is high. That is why the bound for quadratic metric is developed in this thesis. The experiments in the codeword search of the quadratic metric reveal that the new algorithm using the bound for quadratic metric with the partial distortion search is very efficient. The idea of this algorithm is to apply the technique of metric transformation from the quadratic metric to the Euclidean metric. Each input data vector can be transformed from the quadratic metric to the Euclidean metric first, then apply the bound for quadratic metric with the partial distortion search to eliminate impossible codeword matching. As shown in section 3.7, in comparing the new algorithm with the conventional method, the new algorithm will reduce the number of multiplications and the total number of mathematical operations by more than 98 % and 94 %, respectively.

There are two key elements in the design of efficient codeword search algorithms, i.e., an efficient tentative match approach and a powerful elimination criterion. The tentative match approach is used to derive the nearest codeword as soon as possible and the powerful elimination criterion is used to eliminate impossible codeword matching to avoid the full computation of the distortion between the codeword and the data vector. One of the most efficient tentative match approaches in image coding is to use the codeword with the most similar sum of components to the data vector as the most possible candidate. In applying this tentative match approach to the mean-distance-ordered search algorithm (MPS) (Ra & Kim, 1993), a powerful algorithm was reported by Ra and Kim. By extending the IAEI criterion, an even more powerful criterion is obtained. This criterion is the generalised form of the inequality in the mean-distance-ordered search algorithm (MPS). A new and improved algorithm is obtained by modifying the sum of components to a partial sum of components as the tentative match approach and applying the generalised criterion. This algorithm is an improved version of the mean-distance-ordered search algorithm (MPS) and this novel algorithm can be called the improved mean-distance-ordered search algorithm (IMPS). From experiments, without applying partial distortion search algorithm in the IMPS algorithm and the MPS algorithm, the IMPS algorithm will reduce the computation time by more than 43% compared with the MPS algorithm. By applying the partial distortion search algorithm both in the IMPS algorithm and the MPS algorithm, the IMPS algorithm will reduce the number of multiplications by more than 27% and also reduce the total number of mathematical operations about 15% for 1024 codewords.

Normally, the partial distortion search algorithm (PDS) is used at the last stage of the efficient codeword search algorithms because no algorithm can eliminate all impossible codeword matching and the rest of the codewords which cannot be eliminated using some powerful criteria, can be further eliminated using the partial distortion search algorithm. The partial distortion search algorithm is very suitable for general processors in which the operation of multiplication is more expensive than the operation of comparison. In order to enhance the performance of the PDS algorithm to be suitable for any processor, the cost ratio of the comparison computation time

to dimension-distortion computation time is considered, and an improved PDS algorithm and an improved DPPDS algorithm are proposed. If the computation time of the comparison is neglected compared with that of multiplication, the cost ratio will be nearly 0, then the computation time of the improved DPPDS algorithm will be the same as the PDS algorithm. If the cost ratio is 1.0, then the improved DPPDS algorithm will reduce the computation time by more than 27% in comparison with the PDS algorithm.

As described in section 3.4, the extended partial distortion search (EPDS) algorithm is the optimal version of PDS algorithm for considering the number of multiplications needed. The EPDS algorithm is very suitable for computer architectures in which the complexity of comparisons is negligible with respect to that of the multiplications. EPDS algorithm is less suited to some DSP processors in which comparisons are computationally expensive. In order to evaluate and enhance the performance of EPDS algorithm, the cost ratio of the sorting time to dimension-distortion computation time is introduced and the improved EPDS algorithm is proposed. Especially, the optimal inserting point of the sorting and the performance of EPDS and improved EPDS are derived in theory. The improved EPDS algorithm can be applied to dimension-distortion computation for codeword search and the frame-distortion computation for word recognition.

A fast algorithm for approximate codeword search is also presented. Based on the average distortion needed, a rate can be selected. For example, the number of multiplications and the total number of operations will be reduced by more than 80 % and 11 % with only 0.6 % increased distortion for 8 codewords if the selected rate is 1.1 by comparing with the minimax method.

7.2.2 Fast VQ Generation Algorithms

In the efficient algorithms of codebook generation, several fast clustering approaches based on LBG algorithm are proposed and compared. Among these approaches, using the previous partitioned centre as the tentative match with improved AEI and PDS which is called the IPC-type clustering algorithm is the most suitable approach for computer architectures in which the

complexity of comparisons is negligible with respect to that of multiplications. For processor architectures such as those based on the Harvard architecture in which comparisons are computationally expensive, the combination of the previous partitioned centre, triangular inequality elimination (TIE) and PDS which is called a TPC-type clustering algorithm outperforms the other algorithms.

7.2.3 Improved VQ Codebook Design Algorithms

The performance of vector quantization depends on the quality of the codevectors and the existence of a globally optimal algorithm to generate the codevectors. Up to now, no efficient method has been discovered to generate globally optimal codevectors. Although several algorithms were proposed (Ball & Hall, 1967; Linde *et al.*, 1980; Equitz, 1989; Cetin & Weerackody, 1988; Zeger *et al.*, 1992; Chung *et al.*, 1993; Chen *et al.*, 1995) for the design of the codebook, none of these has proven to be globally optimal. In this thesis, genetic algorithms are combined with the GLA algorithm to produce a more optimal algorithm when compared with the GLA algorithm. The approach of stochastic relaxation is also inserted to the mutation of genetic algorithms to further improve this novel algorithm. The main idea of these algorithms is to apply the powerful search ability of genetic algorithms to adapt the value of codevectors. For 32 or 64 codewords, the novel algorithms reduce the mean square error by more than 9% comparing with the GLA algorithm.

7.2.4 New Discoveries of Codebook Index Assignment

Vector quantization is very efficient for data compression of speech and images where the binary indices of the optimally chosen codevectors are sent. Vector quantization as the central data reduction scheme is however highly sensitive to channel errors. The effect of channel errors is to cause errors in the received indices. A parallel genetic algorithm is applied to assign the codevector indices for noisy channels so as to minimize the distortion due to bit errors. A parallel genetic algorithm is a genetic algorithm running on many small subpopulations simultaneously with an occasional identification and exchange of useful information among subpopulations.

The purpose of applying the parallel genetic algorithm in VQ codebook index assignment is not only to use the powerful technique of parallel processors to accelerate the search speed but also a distributed formulation is developed to generate better solutions with less work. Experimental results show that applying a parallel genetic algorithm to the optimization of VQ codebook index assignment will reduce the distortions by more than 59% compared with the random assignment and better than 75% compared with the worst case for 64 codevectors and 0.01 bit error rate. The novel property of multiple global optima has been reported. Using the property of multiple global optima, the complexity of computation can be reduced. All the algorithms (Marca & Jayant, 1987; Vaisey & Gersho, 1988; Farvardin, 1990; Zeger & Gersho, 1990; Zeger & Gersho, 1987) are simulated based on the assumption of single bit error, a condition which is not always true for real applications. The average distortion of the memoryless binary symmetric channel for any bit error in the assignment of codebook indices is also introduced in this thesis.

7.3 Future Work

7.3.1 Quadratic Metric

The bound for quadratic metric not only can be used in HMM-based recognition, but it can also be applied to any codeword search in which the distortion measure is quadratic. In speech recognition systems based on the semi-continuous hidden Markov model, the output probabilities are evaluated as

$$b_i(x) = \sum_{j=1}^{N} c_{ij} \Phi(x, \mu_{ij}, \Sigma_{ij}),$$

where $\Phi(x, \mu_{ij}, \Sigma_{ij})$ are often Gaussians and c_{ij} are the mixture coefficients.

In the most practical implementations, the above summation is extended only to the L most likely Gaussians in the mixture. Thus, the bounds for the quadratic metric can be modified to the search of the L best likely Gaussians in the mixture for a given input data vector X.

The bounds for the quadratic metric are derived from the bound for the Minkowski metric using

methods of metric transformation. This work can thus be extended to investigate other methods of metric transformation and to extend the bound for Minkowski metric to other distortion measures in addition to the quadratic metric.

7.3.2 Vector Quantization of Images

The generalised form of the inequality in the mean-distance-ordered search algorithm (MPS) has been presented. In the proposed new algorithm, each codevector is separated into two sub-vectors only. By using this generalised form, each codevector can be separated into more than two sub-vectors and each sub-vector can be the composition of any components in the codevector. It may be also possible to find the optimal separation of these vector components in theory so that it is the most efficient in the codeword search. In addition, if the sums of the components for the subvectors are calculated first and these values are sorted in increasing order including the indices of codevectors, then the proposed new algorithm can be modified as follows:

Step 1: $FCode\_sum_i = \sum_{j=1}^{k/2} c_i^j$, $SCode\_sum_i = \sum_{j=\frac{k}{2}+1}^{k} c_i^j$ and $TCode\_sum_i = FCode\_sum_i + SCode\_sum_i$ are calculated for each codeword, $i = 1, 2, ..., N$, N is the number of codewords. A sorting list is computed according to the increasing order of the $TCode\_sum_i$.

Step 2: $FData\_sum = \sum_{j=1}^{k/2} x^j$, $SData\_sum = \sum_{j=\frac{k}{2}+1}^{k} x^j$ and $TData\_sum = FData\_sum + SData\_sum$ are calculated.

Step 3: Calculate the tentative matching codeword i using $argMin_i |TData\_sum - TCode\_sum_i|$.

Step 4: Calculate the squared Euclidean distortion D_{min} for the tentative matching codeword. Set l to be the nearest uncalculated codeword to the tentative matching codeword in the sorting list.

Step 5: Check the termination of this program. Test Eq. 3.71 for the neighbour codewords in a back-and-forth manner as in paper (Ra & Kim, 1993), if it is satisfied, delete impossible

codeword matching, set l to be the nearest uncalculated codeword to the tentative matching codeword in the sorting list and goto step 5; Otherwise, goto next step.

Step 6: Test $|FData\_sum - FCode\_sum_l| \geq \sqrt{\frac{k}{2}D_{min}}$ or $|SData\_sum - SCode\_sum_l| \geq \sqrt{\frac{k}{2}D_{min}}$ for the neighbour codewords in a back-and-forth manner as in paper (Ra & Kim, 1993), if it is satisfied, then eliminate impossible codeword matching; otherwise use the IAEI with PDS to the codeword search and update the D_{min}. Set l to be the nearest uncalculated codeword to the tentative matching codeword in the sorting list and goto step 5.

7.3.3 Inequality for Codeword Search

Given codewords $C_i = \{c_i^j; 1 \leq j \leq k\}$, $1 \leq i \leq N$, training data vectors $Y_p = \{y_p^j; 1 \leq j \leq k\}$, $1 \leq p \leq T$ and test data vector $X = \{x^j; 1 \leq j \leq k\}$, from the training data vectors and codewords, compute

$$A(i) = max_p \frac{max_j|c_i^j - y_p^j|}{\sum_{j=1}^{k}(c_i^j - y_p^j)^2} + \delta_1 \tag{7.1}$$

and

$$B(i) = min_p \frac{max_j|c_i^j - y_p^j|}{\sum_{j=1}^{k}(c_i^j - y_p^j)^2} - \delta_2. \tag{7.2}$$

where δ_1 and δ_2 are small scalar values and $1 \leq i \leq N$. For a test vector X, use the minimax method as tentative match and compute

$$n = argmin_i max_j|c_i^j - x^j|. \tag{7.3}$$

If

$$max_j|c_l^j - x^j| \geq \frac{A(l)}{B(n)} max_j|c_n^j - x^j|, \tag{7.4}$$

then

$$\sum_{j=1}^{k}(c_l^j - x^j)^2 \geq \sum_{j=1}^{k}(c_n^j - x^j)^2. \tag{7.5}$$

Eq. 7.4 and 7.5 can be proved as follows:

Assuming that the training data set is sufficiently representative of the test dta so that the ratio $max_j|c_l^j - x^j| / \sum_{j=1}^{k}(c_l^j - x^j)^2$ falls within the range of values $max_j|c_i^j - y_p^j| / \sum_{j=1}^{k}(c_i^j - y_p^j)^2$ observed for the training vectors for each i. From Eq. 7.1 and Eq. 7.2, with the above assumption, the following two equations are obtained.

$$A(l) \geq \frac{max_j|c_l^j - x^j|}{\sum_{j=1}^{k}(c_l^j - x^j)^2} \tag{7.6}$$

and

$$B(n) \leq \frac{max_j|c_n^j - x^j|}{\sum_{j=1}^{k}(c_n^j - x^j)^2}. \tag{7.7}$$

Given

$$max_j|c_l^j - x^j| \geq \frac{A(l)}{B(n)} max_j|c_n^j - x^j| = \frac{A(l)}{B(n)} max_j|c_n^j - x^j|, \tag{7.8}$$

by substituting Eq. 7.7 into Eq. 7.8, Eq. 7.9 is obtained.

$$max_j|c_l^j - x^j| \geq A(l) \sum_{j=1}^{k}(c_n^j - x^j)^2. \tag{7.9}$$

Eq. 7.6 can be rewritten as

$$A(l) \sum_{j=1}^{k}(c_l^j - x^j)^2 \geq max_j|c_l^j - x^j|. \tag{7.10}$$

According to Eq. 7.9 and Eq. 7.10, Eq. 7.5 is obtained and the proof is completed. Eq. 7.4 and 7.5 might be useful in the codeword search. This inequality could combine with the other fast codeword search algorithms. Note that Eq. 7.1 and Eq. 7.2 can be changed to Eq. 7.11 and Eq. 7.12 or some other mathematical forms without having influence on the existence of this inequality, in other words, this inequality can be extended to other distortion measures.

$$A(i) = max_p \frac{max_j|c_i^j - y_p^j|}{\sqrt{\sum_{j=1}^{k}(c_i^j - y_p^j)^2}} + \delta_1. \tag{7.11}$$

$$B(i) = \min_p \frac{\max_j |c_i^j - y_p^j|}{\sqrt{\sum_{j=1}^{k} (c_i^j - y_p^j)^2}} - \delta_2. \tag{7.12}$$

7.3.4 Codebook Design

In the GA-GLA1 algorithm and the GA-GLA2 algorithm, the initial individuals of the population are obtained from a random number generator. If the K-means algorithm is used to generate the initial population, the result might be superior. In the codebook design, the average distortion will be high if the centres of two clusters are very near or too many training data vectors in the same cluster, i.e., the average distortion within one cluster is over some threshold. If two clusters' centres are very near, it is better to merge these two clusters together. If there are too many training data vectors in the same cluster, it is better to split this cluster into two clusters. These properties could be combined with the GA-GLA1 algorithm and GA-GLA2 algorithm, the stochastic relaxation approach and the simulated annealing method to create further improved codebook design methods.

References

Abut, H., Gray, R. M., & Rebolledo, G. (1982). Vector Quantization of Speech and Speech-Like Waveforms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **ASSP-30**, 423–435.

Ahmed, M. E., & Al-Suwaiyel, M. I. (1993). Fast Methods for Code Search in CELP. *IEEE Transactions on Speech and Audio Processing*, **1**(3), 315–325.

Anderberg, M. R. (1973). Cluster Analysis for Applications. *Academic Press*.

Atal, B. S., & Schroeder, M. (1985). Code Excited Linear Prediction (CELP): High quality Speech at very Low Rates. *IEEE International Conference on Acoustics Speech and Signal Processing*, April, 937–940.

Ball, G. H., & Hall, D. J. (1967). A Clustering Technique for Summarizing Multivariate Data. *Behavior Science*, **12**, 153–155.

Balss, U., H. Reininger, H. Schalk, & Wolf, D. (1995). Robust Vector Quantization for Low Bit Rate Speech Coding. *4th European Conference on Speech Communication and Technology*, September, 1057–1060.

Bei, C., & Gray, R. M. (1985). An Improvement of the Minimum Distortion Encoding Algorithm for Vector Quantization. *IEEE Transactions on Communications*, **COM-33**(10), 1132–1133.

Bentley, J. L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. *Communication ACM*, **18**(9), 509–517.

Bezdek, J. C. (1973). Fuzzy Mathematics in Pattern Classification. *PhD Thesis, Department of Apply Mathematics, Cornell University*.

Bohachevsky, Ihor O., Johnson, Mark E., & Stein, Myron L. (1986). Generalized Simulated Annealing for Function Optimization. *Technometrics*, **28**(3), 209–217.

Bottemiller, Robert L. (1992). Comments on 'A New Vector Quantization Clustering Algorithm'. *IEEE Transactions on Signal Processing*, **40**(2), 455–456.

Brindle, A. (1981). Genetic Algorithms for Function Optimization. *PhD Thesis, University of Alberta, Edmonton, Canada*.

Budge, S. E., & Baker, R. L. (1985). Compression of Color Digital Images Using Vector Quantization in Product Codes. *IEEE International Conference on Acoustics Speech and Signal Processing*, April, 129–132.

Buzo, A., Gray, A. H., Gray, R. M., & Markel, J. D. (1980). Speech Coding Based Upon Vector Quantization. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **ASSP-28**(5), 563–574.

Cetin, A. Enis, & Weerackody, Vijitha. (1988). Design Vector Quantizers Using Simulated Annealing. *IEEE Transactions on Circuits and Systems*, **35**(12), 1550–1550.

Chen, C. Q., Koh, S. N., & Sivaprakasapillai, P. (1995). Codebook Generation for Vector Quantization. *Electronics Letters*, **31**(7), 522–523.

Chen, J. H., & Gersho, A. (1987). Gain-Adaptive Vector Quantization with Application to Speech Coding. *IEEE Transactions on Communications*, **COM-35**, 918–930.

Chen, S. H., & Pan, J. S. (1989). Fast Search Algorithm for VQ-Based Recognition of Isolated Word. *IEE Proceedings-I*, **136**(6), 391–396.

Cheng, D., & Gersho, A. (1986). A Fast Codebook Search Algorithm for Nearest-Neighbor Pattern Matching. *IEEE International Conference on Acoustics Speech and Signal Processing*, 265–268.

Cheng, D., Gersho, A., Ramamurthi, B., & Shoham, Y. (1984). Fast Search Algorithms for Vector Quantization and Pattern Matching. *IEEE International Conference on Acoustics Speech and Signal Processing*, 9.11.1–9.11.4.

Chung, F. L., Lee, T., & Chan, W. (1993). Path-following Approach to Globally Optimal Vector Quantizer Design. *Electronics Letters*, **29**(21), 1831–1832.

Cohoon, J. P., Hegde, S. U., Martine, W. N., & Richards, D. (1987). Punctuated Equilibria: A Parallel Genetic Algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, July, 148–154.

Conway, John H., & Sloane, J. A. (1983). A Fast Encoding Method for Lattice Codes and Quantizers. *IEEE Transactions on Information Theory*, **IT-29**(6), 820–824.

Cuperman, Vladimir, Gersho, Allen, Pettigrew, Robert, Shynk, John J., & Yao, Jey-Hsin. (1991). Backward Adaptive Configurations for Low-Delay Vector Excitation Coding. *Advances in Speech Coding, Kluwer Academic Publishers*, 13–23.

Davidson, G., Yong, M., & Gersho, A. (1987). Real-Time Vector Excitation Coding of Speech at 4800 BPS. *IEEE International Conference on Acoustics Speech and Signal Processing*, April, 2189–2192.

Davis, Lawrence. (1991). Handbook of Genetic Algorithms. *Published by Van Nostrand Reinhold*.

Deller, J. R., Proakis, Jr. J. G., & Hansen, J. H. L. (1993). Discrete-Time Processing of Speech Signals. *Macmillan Publishing Company*.

Delport, V., & Koschorreck, M. (1995). Genetic Algorithm for Codebook Design in Vector Quantization. *Electronics Letters*, **31**(2), 84–85.

Devijver, P. A., & Kittler, J. (1982). Pattern Recognition: A Statistical Approach. *Published by Prentice-Hall Inc.*

Dunn, J. C. (1974). A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, **3**(3), 32–57.

Elliott, Douglas F. (1982). Fast Transforms: Algorithms, Analyses, Applications. *Academic Press*.

Equitz, William. (1987). Fast Algorithms for Vector Quantization Picture Coding. *IEEE International Conference on Acoustics Speech and Signal Processing*, 725–728.

Equitz, William H. (1989). A New Vector Quantization Clustering Algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **37**(10), 1568–1575.

Fang, Hsiao-Lan. (1994). Genetic Algorithms in Timetabling and Scheduling. *Ph.D. Thesis, Department of Artificial Intelligence, University of Edinburgh*.

Farrell, K., Mammone, R., & Assaleh, K. T. (1994). Speaker Recognition Using Neural Networks and Conventional Classifiers. *IEEE Transactions on Speech and Audio Processing*, **2**(1), 194–205.

Farvardin, Nariman. (1990). A Study of Vector Quantization for Noisy Channels. *IEEE Transactions on Information Theory*, **36**(4), 799–809.

Fischer, F. P., & Patrick, E. A. (1970). A Preprocessing Algorithm for Nearest Neighbour Decision Rules. *Proc. Nat. Electronics Conf.*, 481–485.

Fissore, L., Laface, P., Massafra, P., & Revera, F. (1993). Analysis and Improvement of the Partial Distance Search Algorithm. *IEEE International Conference on Acoustics Speech and Signal Processing*, 315–318.

Flanagan, J. K., Morrell, D. R., Frost, R. L., Read, C. J., & Nelson, B. E. (1989). Vector Quantization Codebook Generation Using Simulated Annealing. *IEEE International Conference on Acoustics Speech and Signal Processing*, 1759–1762.

Forsyth, Mark E. (1995). Semi-Continuous Hidden Markov Models for Automatic Speaker Verification. *PhD Thesis, Department of Electrical Engineering, University of Edinburgh*.

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An Algorithm for finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, **3**(3), 209–226.

Fukunaga, K., & Narendra, P. M. (1975). A Branch and Bround Algorithm for Computing k-Nearest Neighbours. *IEEE Transactions on Computers*, **C-24**(July), 750–753.

Gamal, Abbas A. EL, Hemachandra, Lane A., Shperling, Itzhak, & Wei, Victor K. (1987). Using Simulated Annealing to Design Good Codes. *IEEE Transactions on Information Theory*, **IT-33**(1), 116–123.

Gersho, A., & Cuperman, V. (1983). Vector quantization: A Pattern Matching Technique for Speech Coding. *IEEE Communication Magazine*, December, 15–21.

Gersho, A., & Ramamurthi, B. (1982). Image Coding Using Vector Quantization. *IEEE International Conference on Acoustics Speech and Signal Processing*, April, 428–431.

Gersho, Allen, & Gray, Robert M. (1992). Vector Quantization and Signal Compression. *Kluwer Academic Publishers*.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. *Addison Wesley Publishing Company*.

Goldberg, D. E. (1990). A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population Oriented Simulated Annealing. *Complex System*, **4**, 445–460.

Goldberg, M., & Sun, H. F. (1986). Image Sequence Coding Using Vector Quantization. *IEEE Transactions on Communications*, **COM-34**(July), 703–710.

Gray, R. M. (1984). Vector Quantization. *IEEE ASSP Magazine*, April, 4–29.

Guan, L., & Kamel, M. (1992). Equal-Average Hyperplane Partitioning Method for Vector Quantization of Image Data. *Pattern Recognition Letters*, **13**(10), 693–699.

Habibi, A. (1974). Hybrid Coding of Pictorial Data. *IEEE Transactions on Communications*, **COM-22**(May), 614–624.

Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. *University of Michigan Press*.

Huang, C. M., Bi, Q., Stiles, G. S., & Harris, R. W. (1992). Fast Full Search Equivalent Encoding Algorithms for Image Compression Using Vector Quantization. *IEEE Transactions on Image Processing*, **1**(3), 413–416.

Huang, S. H., & Chen, S. H. (1990). Fast Encoding Algorithm for VQ-Based Image Coding. *Electronics Letters*, **26**(19), 1618–1619.

Huang, X. D. (1992). Phoneme Classification Using Semicontinuous Hidden Markov Models. *IEEE Transactions on Signal Processing*, **40**(5), 1062–1067.

Huang, X. D., Ariki, Y., & Jack, M. A. (1990). Hidden Markov Models for Speech Recognition. *Edinburgh University Press*.

Itakura, F., & Saito, S. (1970). A statistical method for estimation of speech spectral density and formant frequencies. *Electronics and Communications in Japan*, **53A**, 36–43.

Jeong, Dae G., & Gibson, Jerry D. (1989). Lattice Vector Quantization for Image Coding. *IEEE International Conference on Acoustics Speech and Signal Processing*, 1743–1746.

Jiang, Q., & Zhang, W. (1993). An Improved Method for Finding Nearest Neighbours. *Pattern Recognition Letters*, **14**(7), 531–535.

Juang, B. H., & Gray, A. H. (1982). Multiple Stage Vector Quantization for Speech Coding. *IEEE International Conference on Acoustics Speech and Signal Processing*, April, 597–600.

Kamgar-Parsi, B., & Kanal, L. N. (1985). An Improved Branch and Bound Algorithm for Computing k-Nearest Neighbours. *Pattern Recognition Letters*, **3**, 7–12.

Kang, G. S., & Coulter, D. C. (1976). 600 Bps Voice Digitizer. *IEEE International Conference on Acoustics Speech and Signal Processing*, April, 91–94.

Kirkpatrick, S., C. D. Gelatt, Jr., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, **220**(4598), 671–680.

Kitawaki, Nobuhiko. (1991). Quality Assessment of Coded Speech. *Advances in Speech Signal Processing, Marcel Dekker Inc.*, 357–385.

Koh, J. S., & Kim, J. K. (1988). Fast Sliding Search Algorithm for Vector Quantization in Image Coding. *Electronics Letters*, **24**(17), 1082–1083.

Kubrick, A., & Ellis, T. (1990). Classified Vector Quantization of Images: Codebook Design Algorithm. *IEE Proceedings-I*, **137**(6), 379–386.

Kumazawa, H., Kasahara, M., & Namekawa, T. (1984). A Construction of Vector Quantizers for Noisy Channels. *Electronics and Engineering in Japan*, **67-B**(4), 39–47.

Lee, C. H., & Chen, L. H. (1994). Fast Closest Codeword Search Algorithm for Vector Quantization. *IEE proceedings on Vision Image and Signal Processing*, **141**(3), 143–148.

Leibson, S. H. (1993). EDN-Microprocessor Directory. *EDN*, November, 148–148.

Linde, Y., Buzo, A., & Gray, R. M. (1980). An Algorithm for Vector Quantizer Design. *IEEE Transactions on Communications*, **COM-28**(1), 84–95.

Lippmann, Richard P. (1987). An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine*, April, 4–22.

Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, **IT-28**(2), 129–137.

Lo, K. T., & Cham, W. K. (1993). Subcodebook Searching Algorithm for Efficient VQ Encoding of Images. *IEE Proceedings-I*, **140**(5), 327–330.

Lowry, A., Hossain, S., & Millar, W. (1987). Binary Search Trees for Vector Quantization. *IEEE International Conference on Acoustics Speech and Signal Processing*, 2205–2208.

Lu, N. A., & Morrell, D. R. (1991). VQ Codebook Design Using Improved Simulated Annealing Algorithms. *IEEE International Conference on Acoustics Speech and Signal Processing*, 673–676.

MacQueen, J. (1967). Some Methods for Classification and Analysis of Multivariate Observations. *Proceeding of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, **1**, 281–296.

Marca, J. R. B. De, & Jayant, N. S. (1987). An Algorithm for Assigning Binary Indices to the Codevectors of a Multi-Dimensional Quantizer. *IEEE International Conference on Communications*, June, 1128–1132.

Mathews, V. John. (1992). Multiplication Free Vector Quantization Using L1 Distortion Measure and Its Variants. *IEEE Transactions on Image Processing*, **1**(1), 11–17.

McInnes, F., & Jack, M. (1988). Automatic Speech Recognition Using Word Reference Patterns. *Aspects of Speech Technology, University of Edinburgh Press*, 1–68.

Moayeri, N., Neuhoff, D. L., & Stark, W. E. (1991). Fine-Coarse Vector Quantization. *IEEE Transactions on Signal Processing*, **39**, 1503–1515.

Mohammadi, H. R. Sadeh, & Holmes, W. H. (1994). Fine-Coarse Split Vector Quantization: An Efficient Method for Spectral Coding. *Australian International Conference on Speech Science and Technology*, 118–123.

Nasrabadi, N. M. (1985). Use of Vector Quantizers in Image Coding. *IEEE International Conference on Acoustics Speech and Signal Processing*, March, 125–128.

Nasrabadi, N. M., & King, R. A. (1983). Image Coding Using Vector Quantization in the Transform Domain. *Pattern Recognition Letters*, 323–329.

Nasrabadi, N. M., & King, R. A. (1984). A New Image Coding Technique Using Transform Vector Quantization. *IEEE International Conference on Acoustics Speech and Signal Processing*, March.

Nasrabadi, Nasser M., & King, Robert A. (1988). Image Coding Using Vector Quantization: A Review. *IEEE Transactions on Communications*, **36**(8), 957–971.

Ngwa-Ndifor, J., & Ellis, T. (1991). Predictive Partial Search Algorithm for Vector Quantization. *Electronics Letters*, **27**(19), 1722–1723.

Niemann, H., & Goppert, R. (1988). An Efficient Branch-and-Bound Nearest Neighbour Classifier. *Pattern Recognition Letters*, **7**(2), 67–72.

Nitta, Tsuneo. (1994). Speech Recognition Applications in Japan. *International Conference on Spoken Language Processing*, 671–674.

Nyeck, A., Mokhtari, H., & Tosser-Roussey, A. (1992). An Improved Fast Adaptive Search Algorithm for Vector Quantization by Progressive Codebook Arrangement. *Pattern Recognition*, **25**(8), 799–801.

O'Shaughnessy, Douglas. (1987). Speech Communication: Human and Machine. *Addison-Wesley Publishing Company*.

Paliwal, K. K., & Ramasubramanian, V. (1989). Effect of Ordering the Codebook on the Efficiency of Partial Distance Search Algorithm for Vector Quantization. *IEEE Transactions on Communications*, **37**(5), 538–540.

Pan, J. S. (1988). Fast Speaker Independent Isolated Word Recognition System. *M.S. Thesis, Department of Communication Engineering, Chiao Tung University, Taiwan*.

Pan, J. S., McInnes, F. R., & Jack, M. A. (1994a). Comparison of Fast VQ Training Algorithms. *Proceedings of The Fifth Australian International Conference on Speech Science and Technology*, **1**(December), 106–111.

Pan, J. S., McInnes, F. R., & Jack, M. A. (1994b). Improvements in Extended Partial Distortion Search and Partial Distortion Search Algorithms VQ Search. *Proceedings of The Fifth Australian International Conference on Speech Science and Technology*, **1**(December), 100–105.

Pan, J. S., McInnes, F. R., & Jack, M. A. (1995a). Bound for Minkowski Metric Based on Lp Distortion Measure. *Proceedings of The 4th European Conference on Speech Communication and Technology*, **1**(September), 753–756.

Pan, J. S., McInnes, F. R., & Jack, M. A. (1995b). Improved Absolute Error Inequality Criterion for Vector Quantization. *Proceedings of The First Symposium of the Chinese Institute of Engineers in UK*, 46–46.

Pan, J. S., McInnes, F. R., & Jack, M. A. (1995c). VQ Codebook Design Using Genetic Algorithms. *Electronics Letters*, **31**(17), 1418–1419.

Pan, J. S., McInnes, F. R., & Jack, M. A. (1996a). Application of Parallel Genetic Algorithm and Property of Multiple Global Optima to VQ Codevector Index Assignment. *Electronics Letters*, **32**(4), 296–297.

Pan, J. S., McInnes, F. R., & Jack, M. A. (1996b). Bound for Minkowski Metric or Quadratic Metric Applied to Codeword Search. *IEE proceedings on Vision Image and Signal Processing*.

Pan, J. S., McInnes, F. R., & Jack, M. A. (1996c). Fast Clustering Algorithms for Vector Quantization. *Pattern Recognition*, **29**(3), 511–518.

Pan, J. S., McInnes, F. R., & Jack, M. A. (1996d). A Genetic Algorithm for VQ Codebook Generation. *Proceedings of The Second International Conference on Adaptive Computing in Engineering Design and Control*, March, 319–321.

Pan, J. S., McInnes, F. R., & Jack, M. A. (1996e). Reply: VQ Codebook Design Using Genetic Algorithms. *Electronics Letters*, **32**(3), 194–194.

Pettey, C. B., Leuze, M. R., & Grefenstette, J. J. (1987). A Parallel Genetic Algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, July, 155–161.

Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T. (1986). Numerical Recipes: the Art of Scientific Computing. *Cambridge University Press*.

Ra, S. W., & Kim, J. K. (1991). Fast Weight-Ordered Search Algorithm for Image Vector Quantization. *Electronics Letters*, **27**(22), 2081–2083.

Ra, S. W., & Kim, J. K. (1993). A Fast Mean-distance-Ordered Partial Codebook Search Algorithm for Image Vector Quantization. *IEEE Transactions on Circuits and Systems-II*, **40**(9), 576–579.

Rabiner, L. R., & Juang, B. H. (1986). An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, January, 4–16.

Rabiner, L. R., Rosenberg, A. E., & Levinson, S. E. (1978). Considerations in Dynamic Time Warping Algorithms for Discrete Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **ASSP-26**, 575–582.

Rabiner, Lawrence, & Juang, Biing-Hwang. (1993). Fundamentals of Speech Recognition. *Published by PTR Prentice-Hall Inc.*

Ramamurthi, Bhaskar, & Gersho, Allen. (1986). Classified Vector Quantization of Images. *IEEE Transactions on Communications*, **COM-34**(11), 1105–1115.

Ramasubramanian, B., & Paliwal, K. K. (1990). An Efficient Approximation-Elimination Algorithm for Fast Nearest-Neighbour Search Based on a Spherical Distance Coordinate Formulation. *European Signal Processing Conference*, September, 875–878.

Richter, Stephen L., & DeCarlo, Raymond A. (1983). Continuation Method: Theory and Applications. *IEEE Transactions on Circuits and Systems*, **CAS-30**(6), 347–352.

Sabin, M. J., & Gray, R. M. (1984). Product Code Vector Quantizers for Waveform and Voice Coding. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **ASSP-32**, 474–488.

Saito, T., Takeo, H., Aizawa, K., Harashima, H., & Miyakawa, H. (1986). Adaptive Discrete Cosine Transform Image Coding Using Gain/Shape Vector Quantization. *IEEE International Conference on Acoustics Speech and Signal Processing*, April, 129–132.

Salari, E., & Li, W. (1994). Adaptive Fast Encoding Algorithm for Vector Quantization. *Electronics Letters*, **30**(21), 1733–1734.

Shonkwiler, R. (1993). Parallel Genetic Algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 199–205.

Soleymani, M. R., & Morgera, S. D. (1987a). An Efficient Nearest Neighbor Search Method. *IEEE Transactions on Communications*, **COM-35**(6), 677–679.

Soleymani, M. R., & Morgera, S. D. (1987b). A High-Speed Algorithm for Vector Quantization. *IEEE International Conference on Acoustics Speech and Signal Processing*, 1946–1948.

Soleymani, M. R., & Morgera, S. D. (1989). A Fast MMSE Encoding Technique for Vector Quantization. *IEEE Transactions on Communications*, **37**(6), 656–659.

Stonick, Virginia L., & Alexander, S. T. (1992). Globally Optimal Rational Approximation Using Homotopy Continuation Methods. *IEEE Transactions on Signal Processing*, **40**(9), 2358–2361.

Sun, H. F., & Goldberg, M. (1985). Adaptive Vector Quantization for Image Sequence Coding. *IEEE International Conference on Acoustics Speech and Signal Processing*, March, 339–342.

Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, 2–9.

Tohkura, Yohichi. (1987). A weighted Cepstral Distance Measure for Speech Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **ASSP-35**(10), 1414–1422.

Vaisey, Jacques, & Gersho, Allen. (1988). Simulated Annealing and Codebook Design. *IEEE International Conference on Acoustics Speech and Signal Processing*, 1176–1179.

Vecchi, Mario P., & Kirkpatrick, Scott. (1983). Global Wiring by Simulated Annealing. *IEEE Transactions on Computer-Aided Design*, **CAD-2**(4), 215–222.

Vidal, E. (1986). An Algorithm for Finding Nearest Neighbours in (Approximately) Constant Average Time. *Pattern Recognition Letters*, **4**(3), 145–157.

Ward, J. (1963). Hierarchical Grouping to Optimize an Objective Function. *Journal of American Statistics Association*, **58**(March), 236–244.

Wilpon, Jay G., & Roe, David. (1994). Applications of Speech Recognition Technology in Telecommunications. *International Conference on Spoken Language Processing*, 667–670.

Wong, D. Y., Juang, B. H., & Gray, A. H. (1982). An 800 B/S Vector Quantization LPC Vocoder. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **ASSP-30**(5), 770–780.

Wu, Jianxiong, & Chan, Chorkin. (1993). Isolated Word Recognition by Neural Network Models with Cross-Correlation Coefficients for Speech Dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **15**(11), 1174–1184.

Zeger, K. A., & Gersho, A. (1987). Zero Redundancy Channel Coding in Vector Quantization. *Electronics Letters*, **23**(12), 654–656.

Zeger, K. A., & Gersho, A. (1989). Stochastic Relaxation Algorithm for Improved Vector Quantizer Design. *Electronics Letters*, **25**(14), 896–898.

Zeger, Kenneth, & Gersho, Allen. (1990). Pseudo-Gray Coding. *IEEE Transactions on Communications*, **38**(12), 2147–2158.

Zeger, Kenneth, Vaisey, Jacques, & Gersho, Allen. (1992). Globally Optimal Vector Quantizer Design by Stochastic Relaxation. *IEEE Transactions on Signal Processing*, **40**(2), 310–322.

Appendix A

Publications by the author

Journal Papers

1. † J. S. Pan, F. R. McInnes and M. A. Jack, "VQ Codebook Design Using Genetic Algorithms", IEE Electronics Letters, Vol. 31, No. 17, 1418–1419, 1995

2. J. S. Pan, F. R. McInnes and M. A. Jack, "Fast VQ Codeword Search and Genetic Algorithms for VQ Codebook Design", Postgraduate Journal of The Department of Electrical Engineering, University of Edinburgh, Issue 2, 33–37, January, 1996

3. † J. S. Pan, F. R. McInnes and M. A. Jack, Reply: "VQ Codebook Design Using Genetic Algorithms", IEE Electronics Letters, Vol. 32, No. 3, 194–194, 1996

4. † J. S. Pan, F. R. McInnes and M. A. Jack, "Application of Parallel Genetic Algorithm and Property of Multiple Global Optima to VQ Codevector Index Assignment", IEE Electronics Letters, Vol. 32, No. 4, 296–297, 1996

5. † J. S. Pan, F. R. McInnes and M. A. Jack, "Fast Clustering Algorithms for Vector Quantization", Pattern Recognition, Vol. 29, No. 3, ISSN 0031-3203, 511–518, March, 1996

6. † J. S. Pan, F. R. McInnes and M. A. Jack, "Bound for Minkowski metric or quadratic metric applied to codeword search", IEE-Proceedings on Vision Image and Signal Processing, Vol. 143, No. 1, 67–71, February, 1996

Conference Papers

1. J. S. Pan, F. R. McInnes and M. A. Jack, "Improvements in extended partial distortion search and partial distortion search algorithms VQ search", Proceedings of The Fifth Australian International Conference on Speech Science and Technology (SST-94), 100–105, 1994

2. J. S. Pan, F. R. McInnes and M. A. Jack, "Comparison of fast VQ training algorithms", Proceedings of The Fifth Australian International Conference on Speech Science and Technology (SST-94), 106–111, 1994

3. J. S. Pan, F. R. McInnes and M. A. Jack, “Improved absolute error inequality criterion for vector quantization”, Proceedings of The First Symposium of The Chinese Institute of Engineers in UK, 46–46, 1995

4. J. S. Pan, F. R. McInnes and M. A. Jack, “Bound for Minkowski metric based on Lp distortion measure”, Proceedings of The 4th European Conference on Speech Communication and Technology, 753–756, 1995

5. J. S. Pan, F. R. McInnes and M. A. Jack, “A genetic algorithm for VQ codebook generation”, Proceedings of The Second International Conference on Adaptive Computing in Engineering Design and Control’96, 319–321, March, 1996

6. J. S. Pan, F. R. McInnes and M. A. Jack, “VQ codevector index assignment using genetic algorithms for noisy channels”, Proceedings of The Fourth International Conference on Spoken Language Processing, 1996

† Reprinted in this appendix