Assiut University

Faculty of Engineering

Image Compression Using Vector Quantization

By

Mohammed Farouk Abdellatif

B.Sc., Electrical Engineering (Electronics and Communication),

Assiut University, 2001

A Thesis submitted in partial fulfillment of the requirement for the degree

MASTER OF SCIENCE

Electrical Engineering Department

Assiut University

Assiut, EGYPT

Supervision Committee:

Prof. Dr. Hany Selim
(Assiut University)

Prof. Dr. Magdy M. Doss
(Assiut University)

Dr. Tarik K. Abdel-Hamid
(Assiut University)

Discussion Committee:

Prof. Dr Abdel-Fattah Ibrahim A.
(Mansoura University)

Prof. Dr. Mohamed AboZahhad A.
(Assiut University)

Prof. Dr. Hany Selim

Prof. Dr. Magdy M. Doss

بسم الله الرحمن الرحيم

"سبحانك لاعلم لنا إلاماعلمتنا إنك أنت العليم الحكيم"

صدق الله العظيم

To my family

ACKNOWLEDGEMENTS

I am grateful to **Prof. Dr. Hany Selim**, Professor, Electrical Engineering Department, Assiut University, for giving me the opportunity to be one of his graduate students and his supplementary advices and supervision.

I wish to take this opportunity to express my deepest thanks to **Dr. Tarik K. Abdel-Hamid,** Electrical Engineering Department, Assiut University, who gives me a wealth of inspiration, for his kind supervision, generous advice and his support all the time during my work.

I also would like to express my great thanks to **Prof. Dr. Magdy M. Doss**, Professor, Electrical Engineering Department, Assiut University, for his support, clarifying suggestions and his clear vision provided during my work.

I wish to offer words of appreciation to all my friends in the electrical engineering department, Assiut University, especially to **Eng. Peter E. William** for his valuable cooperation and great technical help.

I also want to express my gratitude to my great family for their encouragement, continuous support, their patience and perseverance they showed during the preparation of this thesis.

Mohammed Farouk

ABSTRACT

In practical Vector Quantization (VQ) of images, the used block dimensions are kept small to reduce the computations cost. This in turn results in highly correlated blocks and the corresponding VQ indices will inherent this high correlation. This high correlation could be used to increase the compression ratio of the basic VQ while keeping the same reconstructed image quality by inserting an index compression stage after VQ. Index compression algorithms work in the index domain with simpler algorithms than that used with memory VQ, which is also utilizing the high correlation between adjacent blocks.

This thesis presents three contributions for VQ index compression to increase the compression ratio of the basic VQ used for image compression.

The first contribution is a modification of the Hu and Chang method [5]. In the original method, the relation between the current index and both the upper and left indices is classified into four different cases. In the modified method, a fifth case is included that applied when the difference between the current index and the index to the left is less than the predefined threshold. Variable length codes generated using Huffman coding are used as prefixes to distinguish the five cases.

The second contribution is a new index compression algorithm that utilizes the local statistics of each image and the repeated pattern of its adjacent indices. A "Next" (or "Right") table is generated for each image. It is a 1xN index array , where N is the codebook size, which keeps for each index a pointer for the most probable index that comes next (or right) to it. These tables are used to categorize the indices into five cases if an index repeated pattern is found.

The third contribution is a remapping process applied to the VQ indices and inserted before any index compression algorithm to increase the compression ratio. The total number of indices used in different parts of image is limited. A

remapping process is applied to different image parts to renumber the used indices and neglect the unused indices. Different index compression algorithms are applied to the remapped indices and the effect of the remapping process is studied. It is found that the remapping process increases the index compression ratio of other algorithms.

TABLE OF CONTENTS

PAGE

List of Tables

TABLE PAGE

Chapter III

Chapter VI

LIST OF FIGURES

FIGURE PAGE

Chapter II

Chapter III

Chapter IV

Chapter V

Chapter VI

LIST OF ABBREVIATIONS

BR	Bit Rate
bpp	Bit Per Pixel
BTC	Block Truncation Coding
CVQ	Classified VQ
DCT	Discrete Cosine Transform
DPCM	Differential Pulse Code Modulation
EZT	Embedded Zero Trees
FSVQ	Finite State VQ
FSA	Full Search Algorithm
GLA	Generalized Lloyds Algorithm
GA	Genetic Algorithms
HDTV	High Definition TV
HVS	Human Visual System
IGA	Index Grouping Algorithm
JPEG	Joint Photographic Experts Group
LPC	Linear Predictive Coding
LBG	Linde-Buzo-Gray
MSE	Mean Square Error
MAE	Mean Absolute Error
MOS	Mean Opinion Score
PDE	Partial Distance Elimination
PVQ	Predictive VQ
PSNR	Peak Signal to Noise Ratio
PNN	Pairwise Nearest Neighbor
SA	Simulated Annealing
SGVQ	Shape/Gain VQ
SOM	Self Organizing Maps
SOC	Search Order Coding
SBC	Sub Band Coding
STC	Switching Tree Coding
TIE	Triangle Inequality Elimination
TSVQ	Tree Structured VQ
RLE	Run Length Encoding
ROI	Region Of Interest
RMVQ	Removed Mean VQ
Rms	Root Mean Square
VLSI	Very Large Scale Integration
VQ	Vector Quantization

CHAPTER I

Introduction

Image compression is a growing field for research and industry. Progress in other fields such as digital communication, data networks, and multimedia requires new varieties of image compression algorithms. The computing power is improving with the development in Very Large Scale Integration (VLSI) technology and its cost is reducing. Image compression is rapidly becoming a dominant technology in the information age. Storing and transmitting the digital image over communication systems are major problems. The amount of data required to represent images at an acceptable level of quality is extremely large. Compression is utilized to reduce the data while keeping acceptable image quality. Algorithms for image compression utilize the high redundancy found in natural images and the limitations found in the Human Visual System (HVS) like low sensitivity to high frequency details. The natural images contain edges and textures which could be represented with different amount of bits. Vector Quantization (VQ) [1]-[3] was found to be a compression method with low complexity and high quality. VQ is a pattern matching process ideally suited for coding digital images in a compact form. Lossless index compression algorithms are introduced to increase the compression ratio of ordinary VQ while keeping the same image quality [4]-[11]. This thesis presents three new VQ index compression algorithms. A description of the new proposed algorithms for VQ index compression and the corresponding experimental results are given. A number of present VQ index compression algorithms are implemented and used for

evaluating the proposed algorithms. Increased compression ratio is achieved using the new proposed algorithms over the present algorithms.

This thesis is organized as follows; in chapter II an overview of image compression is presented. The overview starts with the importance of image compression in modern life. The definitions of digital image and image types are stated and the types of redundancy found in images are mentioned. Lossy and Lossless image compression methods are compared. The different quantitative measures for compressed image quality are discussed. Finally, image compression methods are described briefly.

Chapter III defines VQ. Its application in classification, enhancement, and other image processes are discussed. Codebook design algorithms are given. These algorithms include the famous Linde, Buzo, and Gray (LBG) method [12], the pairwise nearest neighbor (PNN) method [13], other methods will be stated in a later chapter. Fast encoding algorithms are presented. Constrained structure VQ such as removed mean VQ [1], shape-gain [14], Classified VQ [15], and Tree Structure VQ [1] are described briefly.

Chapter IV introduces some known VQ index compression algorithms. Index compression algorithms rely on the high correlation found between adjacent VQ indices. Address VQ [4] is presented, it is a lossless VQ algorithm applied to the ordinary VQ indices. The low complexity algorithm proposed by Hu and Chang [5] is introduced, where the correlation between any index and its upper and left adjacent indices is utilized for index compression. The Search order coding (SOC) algorithm [6] is presented where the neighboring indices are searched for a match to the current index in a predefined search path. Switching Tree Coding (STC) [7] encodes the indices with different prefixes depending on the adjacent indices.

In Chapter V the proposed algorithms are introduced. Three contributions are given. The first contribution is a modification of the Hu and Chang method to full explore the correlation between the adjacent indices. Five cases are defined instead of four cases as in the original Hu and Chang method. This results in increased reduction in bit rate than the original Hu and Chang method. The second contribution is the repeated adjacencies method in which the repeating pattern of adjacent indices is explored and utilized for index compression. The third contribution is the index remapping process which results in a reduced number of indices to be used to encode the image parts and higher compression ratio is achieved.

Chapter VI shows the simulation results with detailed comparison of the different index compression methods and the effect of remapping on the VQ index compression algorithms.

Conclusions and future work are given in chapter VII.

CHAPTR II

Image Compression

2.1. Why Image Compression?

Image compression is one of the most active fields in digital signal processing. With the fast growing development of personal computers, parallel and multiprocessing, specialized hardware and software for imaging, the market for image compression technology has been growing rapidly.

Its importance increased with the progress in the telecommunication, the popularity of internet, medical imaging, multimedia applications, and High Definition TV (HDTV). This gives special importance to invent compression algorithms that meet the challenges of different systems' demands of bit rate, image quality, and transmission time delay for real time performance.

The image in its raw format needs huge amount of data to be stored. A typical grayscale image of 512×512 pixels, each represented by 8 bits, contains 256 kilobytes of data. With the color information, the number of bytes is tripled, therefore any storage or transmission system will suffer from shortage of resources when using image in its raw format. With data compression, one can store more information in a given storage space or transmit information faster over communication channels [16]. Recently, there have been different varieties of compression standards that can achieve a much wide range of compression ratios from HDTV at 20 Mbps to very low resolution telephone transmission at the rate of 9.6 kbps.

The field of image compression is based on a solid foundation of classical methods of transform coding, vector quantization and recent advances of wavelet theory. In the past, image compression algorithms have relied on

classical information theory to achieve respectable compression. These techniques such as Differential Pulse Code Modulation (DPCM) [1] and Huffman coding [17] suffer from low compression ratios. Recently, attempts have been made to develop new image compression techniques that achieve better compression ratios. The new image coding techniques attempt to identify redundant features within the image and remove these features to achieve compression. Using these new methods, such as JPEG [18], it is possible to achieve high compression ratios while maintaining acceptable image quality.

The primary objective of image compression algorithms is to reduce redundancy in data representation in order to decrease data storage requirement. Data compression is the mapping of a data set into a bit stream with a reduced number of bits.

A generic compression system has three components; an encoder, a channel, and a decoder. The function of the encoder is to produce a representation of the data in a form that takes much less storage than the original data and the function of the decoder is to recover the original data after transmission through the channel. Thus the decoder performs a reverse function of the encoder function.

Fig (2.1): A generic compression system.

2.2. Digital Image Definitions

A digital image in a 2D discrete space is derived from an analog image in a 2D continuous space through a sampling process. The effect of sampling is shown in Fig. (2.2), the 2D continuous image is divided into N rows and M columns. The intersection of a row and a column is termed a pixel. The value assigned to the integer coordinates (m,n) with {m=0,1,2,…,M–1} and {n=0,1,2,…,N–1} is a(m,n).

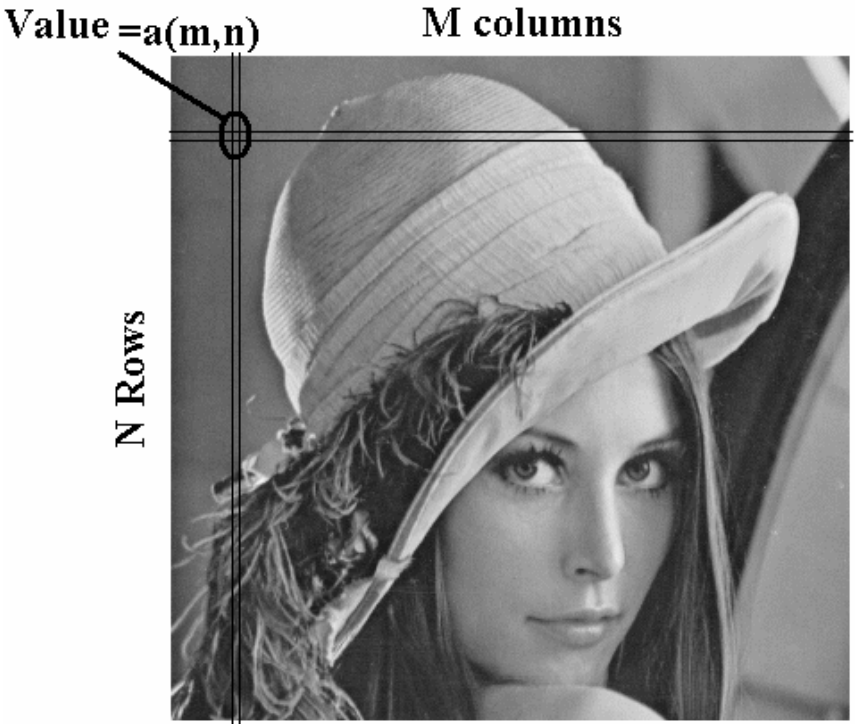


Fig (2.2): Digitization of a continuous image. The pixel at coordinates (m, n) has the integer brightness value a(m,n).

Images can be classified according to the information contents into binary images, grayscale images, and color images. The compression algorithms to grayscale images and color images are similar, since they are related to each other. But for binary images where the pixels can take only two values 0 and 1, the algorithms are different from that of the gray scale and color images algorithms.

A B C

Fig (2.3): Image types a) color image b) grayscale image c) binary image

2.3. Redundancy and Irrelevance

2.3.1. Redundancy

Normally there are some correlations that occur between image components. These correlations increase by increasing the number of pixels per inch (sampling rate) [19]. These correlations indicate the existence of redundant data that can be removed to compress the image without any loss in information. The redundancy found in images can be classified into the following:

1) Spatial: Correlation between neighboring pixel values.

2) Spectral: Correlation between different color planes (in colored images) or spectral bands.

3) Temporal: Correlation between different frames in a video sequence.

An example of the redundancy removing compression methods is the prediction coding methods such as DPCM where the original information can be recovered exactly.

2.3.2. Irrelevance

Irrelevant means visually unrecognized and it is defined as the amount of information that can be removed from the image without any loss of

recognized quality. Irrelevance can be removed from the raw image using quantization where the infinite number of intensity levels is reduced to 256 gray levels only. The quantization is a lossy step where the original infinite levels can not be recovered again. The efficient compression system works to remove the redundant and irrelevant information from the image, see Fig (2.4).

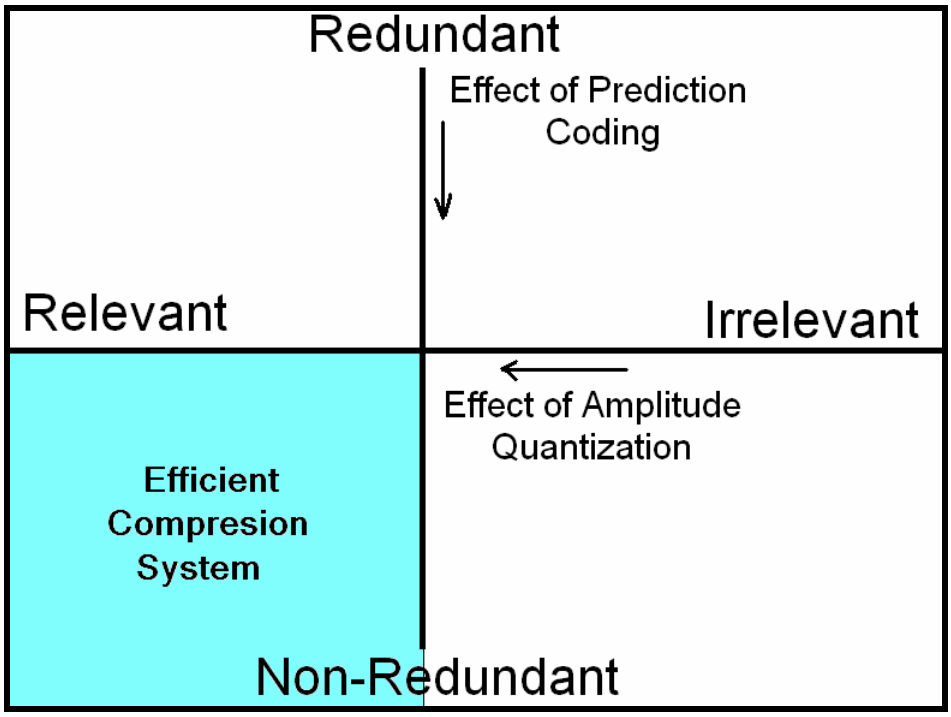


Fig (2.4): Redundancy and irrelevance

2.4. Lossy versus Lossless Compression

Compression methods can be classified according to the distortion caused by compression process into lossless and lossy compression. Lossless compression methods (also called information preserving or reversible) keep the reconstructed image identical with the original; only the redundancy is removed from the image. The lossless compression algorithms are used for all kinds of documents, statistical databases, medical and biological images, binary data such as executables, and remote sensed satellite data, since they

need to be exactly reproduced when decompressed. On the other hand, lossy compression methods (also called irreversible) are associated with loss of information and hence quality. It is useful in image and video processing and constitutes a significant part of data transmission activities where the data rate is more important than quality [16].

Compressing an image is significantly different from compressing raw binary data [20]. This is because images have certain statistical properties which can be exploited by encoders specifically designed for them. Image lossy compression relies on the low pass filter characteristics of the HVS to ignore the high spatial frequency details where it is less visible to the human eye. An approximation of the original image is enough for most purposes, as long as the error between the original and the compressed image is tolerable. Lossy compression is always a trade-off between the bit rate and the image quality. Generally, lossless algorithms cannot produce high compression ratios as the lossy ones.

Image compression algorithms can be implemented with software or hardware. Complexity is essentially different for the software or hardware implementation. The latter depends on the state and availability of technology while the former provides a benchmark for comparison purposes.

2.5. Quantitative Measures

To evaluate the compression algorithm performance, many quantitative measures are used.

1-Compression ratio: It is the ratio between the number of bits used to represent the original image to the number of bits used to represent the compressed image

$$\text{compression ratio} = \frac{N \times K}{C} \qquad (2.1);$$

Where C is the number of bits in the compressed image, N is the number of pixels in the image, and K is the number of bits per pixel in the original image. It affects the bit rate for transmission applications or storage size for storage applications.

Bit rate (BR) gives the average number of bits per pixel (bpp) of the image:

$$BR = \frac{C}{N} \text{ Bits per pixel (bpp)} \qquad (2.2);$$

2-*Reconstructed image quality*: The quantitative distortion of the reconstructed image is commonly measured to judge the compression efficiency. To be useful, the quantitative determined should be easy to compute and meaningful for perception or application [21]. Most common measures are: mean square error (MSE), Weighted or transform/weighted versions are used for perceptual purposes, input-weighted squared error, the mean absolute error (MAE), and peak signal to noise ratio (PSNR). The MSE is the cumulative squared error between the reconstructed and the original image, whereas PSNR is a measure of the peak signal of the error. The mathematical formulas for these measures are

$$\text{MSE} = \left[\frac{1}{MN} \sum_{y=1}^{M} \sum_{x=1}^{N} [I(x,y) - I'(x,y)]^2 \right] \qquad (2.3)$$

$$\text{PSNR} = 20 \times \log_{10} \left(\frac{2^n - 1}{\sqrt{MSE}} \right) \qquad (2.4)$$

Where *I(x,y)* is the original image pixel value, *I'(x,y)* is the decompressed image value and *M,N* are the dimensions of the image, and n is number of bits used to represent the pixel values. A lower value for MSE means less error, and as seen from the inverse relation between the MSE and PSNR, this

translates to a high value of PSNR. Logically, a higher value of PSNR is good because it means that the ratio of signal to noise is higher. Here, the 'signal' is the peak signal of the original image, and the 'noise' is the error in reconstruction. Unfortunately, these measures are not enough to judge the image quality; we need to include the visual limitations. In the objective measures the quality is evaluated by humans. Mean Opinion Score (MOS) is used as objective measure, the image is evaluated by many people and given a score from 1 to 5, where 1 for the poor quality and 5 for the best quality.

3- *Time of the compression and decompression process*: This is important in real time applications where data should be processed on-line, any delay should not be allowed.

4- *Complexity*: Usually, it is measured with the number of additions and multiplications in the algorithm. When the algorithm is to be implemented as hardware, complexity can be measured by the number of components.

5- *Robustness against transmission errors*: It is an important factor for transmission systems where the transmission errors should minimally affect the visual image quality and does not introduce any subsequent errors.

2.6. Image Compression Methods

Lossless compression methods include run length encoding, Huffman coding, Lempel-Ziv, and bit plane coding. Lossy compression methods include sub band coding, transform coding, and Vector Quantization.

2.6.1. Lossless Compression Methods

2.6.1.1. Run Length Encoding (RLE)

RLE is one of the simplest data compression techniques that take the advantage of repetitive data [22]. Run length coding looks for any row of pixel brightness values that repeats exactly, and replaces it with the value and the

number of pixels. For natural grey scale images, such rows do not occur very often and little compression is achieved. However, Run length encoding works very well for images with solid backgrounds like cartoons, computer graphics, drawings, animation images, and binary images. Fax machines use RLE to send images of via telephone lines.

2.6.1.2. Lempel-Ziv

The first compression algorithm described by Ziv and Lempel is commonly referred to as LZ77 [23]. It is relatively simple dictionary based method. The dictionary consists of all the strings in a window into the previously read input stream. A file compression program, for example, could use a 4K-byte window as a dictionary. While new groups of symbols are being read in, the algorithm looks for matches with strings found in the previous 4K bytes of data already read in. Any matches are encoded as pointers sent to the output stream.

The LZ78 algorithm, introduced n 1978 [24], takes a different approach to building and maintaining the dictionary. Instead of having a limited size window into the preceding text, LZ78 builds its dictionary out of all of the previously seen symbols in the input text. But instead of having carte blanche access to all the symbol strings in the preceding text, a dictionary of strings is built a single character at a time. Unlike LZ77 methods, strings in LZ78 can be extremely long, which allows for high compression ratios.

2.6.1.3. Arithmetic Coding

Arithmetic coding takes in the complete data stream and outputs one specific codeword [25] [26]. This codeword is a floating point number between 0 and 1.The bigger the input data set, the more digits in the output number. This unique number is encoded such that when decoded, it will output the exact

input data stream. Arithmetic coding is a two pass algorithm. The first pass computes the characters' frequency and generates a probability table. The second pass does the actual compression.

The probability table assigns a range between 0 and 1 to each input character. The size of each range is directly proportional to a characters' frequency. The order of assigning these ranges is not as important as the fact that it must be used by both the encoder and decoder. The range consists of a low value and a high value. These parameters are very important to the encode/decode process. The more frequently occurring characters are assigned wider ranges in the interval requiring fewer bits to represent them. The less likely characters are assigned more narrow ranges, requiring more bits.

2.6.1.4. Huffman Coding

With Huffman coding [17] [26], variable length codes are assigned to the characters, the length of the encoded character is inversely proportional to that character's frequency. Variable length codes can achieve a higher data density than fixed length codes if the characters differ in frequency of occurrence.

Huffman codes are created by analyzing the data set and assigning short bit streams to the symbols occurring most frequently. The algorithm attempts to create codes that minimize the average number of bits per character.

During the codes creation process, a binary tree representing these codes is created. It is easy to get codes from the tree. Start at the root and trace the branches down to the letter of interest. Every branch that goes to the right represents a 1. Every branch to the left is a 0.

Using a binary tree to represent Huffman codes insures that our codes have the prefix property. This means that one code cannot be the prefix of another code.

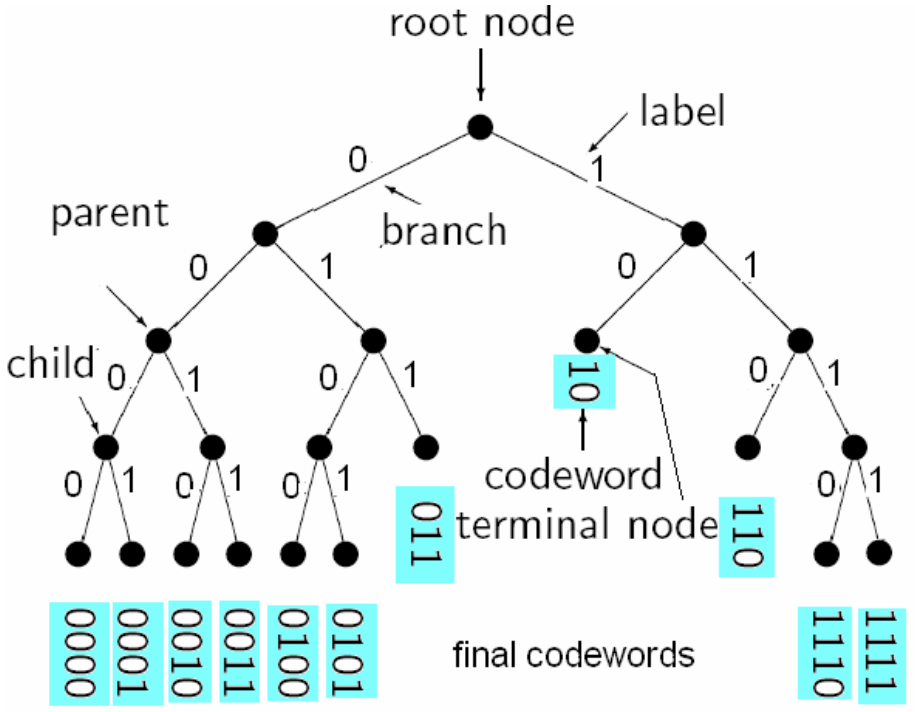


Fig (2.5): Huffman tree

The first step in creating Huffman codes is to create an array of character frequencies. This is as simple as parsing your data and incrementing each corresponding array element for each character encountered. The binary tree can easily be constructed by recursively grouping the lowest frequency characters and nodes.

One drawback to Huffman coding is that encoding requires two passes over the data. The first pass accumulates the character frequency data, which is then compressed on the second pass. One way to remove a pass is to always use one fixed table. Of course, the table will not be optimized for every data set that will be compressed.

The decoder must use the same binary tree as the encoder. Providing the tree to the decoder in advance requires using a standard tree that may not be optimum for the code being compressed. Another option is to store/send the binary tree with the data. Rather than storing/sending the tree, the character

frequency could be stored/sent and the decoder could regenerate the tree. This would increase decoding time. Adding the character frequency to the compressed code decreases the compression ratio.

2.6.1.5. Bit Plane Coding

The idea of bit plane coding is to apply binary compression methods for the bit planes of the gray-scale image. The image is first divided into *k* separate bit planes, each representing a binary image. These bit planes are then coded by any compression method designed for the binary data. The bit planes of the most significant bits are the most compressible, while the bit planes of the least significant bits are nearly random and thus mostly uncompressible. See Fig. (2.6).

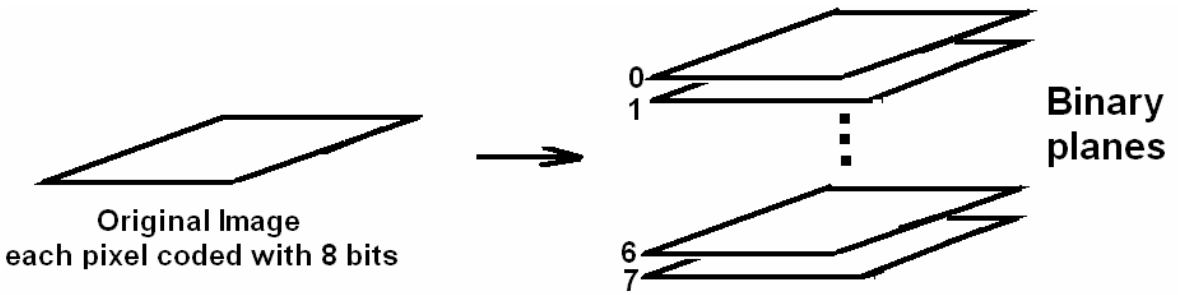


Fig (2.6): Bit plane coding

2.6.1.6. Predictive Coders

The principle of the predictive coders is very simple, each pixel value is predicted from the adjacent already predicted pixel values and the difference between the prediction and the original value is sent. The simplest predictive coders are DPCM and Delta Modulation [22]. In DPCM the prediction process is simply a subtraction from the previous pixel value.

2.6.2. Lossy Compression Methods

2.6.2.1. Subband Based Coding (SBC)

The fundamental concept behind SBC [19] [27] is to split up the frequency band of the image and then coding each sub and using a coder and bit rate that corresponds to the importance of the band in the reconstructed image quality. It consists of analysis stage at the transmitter and synthesis stages at the receiver. Wavelets are used widely in sub band compression.

2.6.2.2. Transform Coding

There are many transforms used in image compression such as Karhunen-Loeve transform, Discrete Fourier transform, Discrete Cosine transform (DCT), and Walsh-Hadamard transform [28]. Transforms convert the image blocks into another domain where some certain features are found, for example DCT concentrates the blocks energy in few coefficients.

2.6.2.3. Vector Quantization

Vector quantization [1] [2] [3] [26] divides the image into blocks (or vectors) of $n \times n$ pixels. These blocks are then compared with a set of representative blocks. This collection of representative blocks or codevectors is called a codebook. The codebook entry with the smallest difference is chosen as the representative codevector. The index of that codevector is then stored to a file or transmitted. VQ has many variants. All VQ algorithms are computationally intensive during the encoding stage, but the decoding is relatively simple. The decoding process is merely a lookup process for codevectors in the codebook and reconstruction of the image.

2.6.2.4. Fractals

Fractals [29] [16] can be considered as a set of mathematical equations (or rules) that generate fractal images; images that have similar structures repeating themselves inside the image. The image is the inference of the rules and it has no fixed resolution like raster images. The idea of fractal compression is to find a set of rules that represent the image to be compressed. The decompression is the inference of the rules. In practice, fractal compressions try to decompose the image into smaller regions which are described as linear combinations of the other parts of the image. These linear equations are the set of rules.

2.6.2.5. Block Truncation Coding

The basic idea of block truncation coding (BTC) [30] [16] is to divide the image into 4×4 pixel blocks and quantize the pixels to two different values, *a* and *b*. For each block, the mean value ($\bar{x}$) and the standard deviation (σ) are calculated and encoded. Then two level quantization is performed for the pixels of the block so that a 0-bit is stored for the pixels with values smaller than the mean, and the rest of the pixels are represented by a 1-bit. The image is reconstructed at the decoding phase from the $\bar{x}$ and σ, and from the bit plane by assigning the value *a* to the 0-value pixels and *b* to the 1-value pixels:

$$a = \bar{x} - \sigma \cdot \sqrt{\frac{q}{m-q}} \tag{2.5}$$

$$b = \bar{x} + \sigma \cdot \sqrt{\frac{m-q}{q}} \tag{2.6};$$

Where *m* (=16) is the total number of the pixels in the block and *q* is the number of 1-bits in the bit plane. The quantization level values were chosen so

that the mean and variance of the pixels in the block would be preserved in the decompressed image.

2.6.2.6. Model Based Compression

The idea here is to characterize the source data in terms of some strong underlying model [31]. The popular example here is faces. We might devise a general model of human faces, describing them in terms of anatomical parameters like nose shape, eye separation, skin color, cheekbone angle, and so on. Instead of transmitting the image of a face, we could transmit the parameters that define that face within our general model. Assuming that we have a suitable model for the data at hand, we may be able to describe the entire system using only a few bytes of parameter data. Both sender and receiver share a large body of priori a knowledge contained in the model itself (e.g., the fact that faces have two eyes and one nose). The more information is shared in the model, the less need be transmitted with any given data set.

2.7. JPEG

JPEG is the widely used Still Image Compression Standard. JPEG is an acronym for "Joint Photographic Experts Group." Which is the organization created the standard. Although JPEG is computationally intensive, its excellent compression generally outweighs the processing required. JPEG is not suitable for some applications, since JPEG in general is lossy, which makes it unsuitable for an intermediate storage format when an image file is being repeatedly edited. JPEG is also not good at compressing text, cartoons and drawings.

JPEG is based on the DCT transform. The compression algorithm is shown in Fig. (2.7).

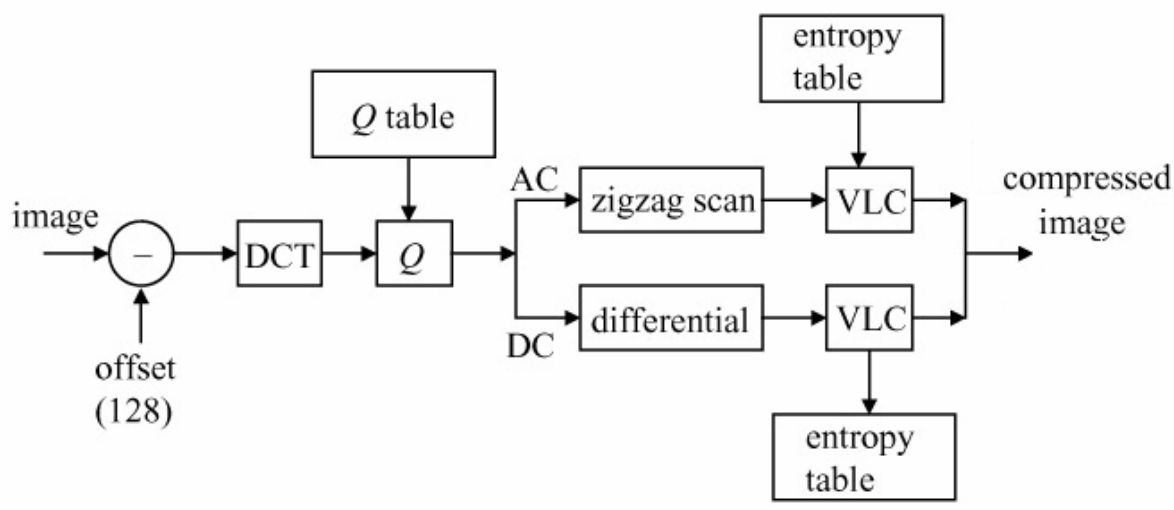


Fig (2.7): JPEG diagram

2.7.1. JPEG Compression Modes

The original JPEG standard defined four compression modes: hierarchical, progressive, sequential, and lossless. In addition, the standard defined multiple encoding processes for the modes. Fig (2.8) shows the relationship of the major JPEG compression modes and encoding processes. While there is some commonality among them, for the most part they must be implemented as completely different techniques.

Sequential				Progressive				Lossless		Hierarchical
Huffman		Arithmetic		Huffman		Arithmetic		Original Lossless	JPEG-LS	
8-Bit	12-Bit	8-Bit	12-Bit	8-Bit	12-Bit	8-Bit	12-Bit			

Fig (2.8): JPEG compression modes

Sequential mode: It is the plain JPEG mode. As the name implies, sequential mode images are encoded from top to bottom. Sequential mode supports sample data with 8 and 12 bits of precision. In sequential JPEG, each color component is completely encoded in a single scan. In most formats, the entire compressed pixel data is stored in one contiguous region in the file. In JPEG,

each pass through the image is stored in a distinct data block called a scan. Within sequential mode, two alternative entropy encoding processes are defined by the JPEG standard: one uses Huffman encoding; the other uses arithmetic coding.

Progressive mode: In progressive JPEG images, components are encoded in multiple scans. The compressed data for each component is placed in a minimum of 2 and as many as 896 scans, although the actual number is almost always at the low end of that range. The initial scans create a rough version of the image while subsequent scans refine it. Progressive images are intended for viewing as they are decoded. They are useful when an image is being downloaded over a network or used in a Web browser because they allow the user to get an idea of what the image contains after as little data as possible has been transmitted.

The main drawbacks of progressive mode are that it is more difficult to implement than sequential and that, if the image is viewed as it is downloaded, it requires much more processing. Progressive JPEG is most suitable when the relative processing power exceeds the relative transmission speed of the image. In general, progressive mode image files tend to be about the same size as their sequential counterparts.

Hierarchical mode: Hierarchical JPEG is a super progressive mode in which the image is broken down into a number of sub images called frames. A frame is a collection of one or more scans. In hierarchical mode, the first frame creates a low-resolution version of the image. The remaining frames refine the image by increasing the resolution. Hierarchical is better than progressive when low transmission rates are used. If only a low resolution of an image is desired, not all of the frames are required to get the desired result.

The obvious drawback of hierarchical mode is its complexity. Hierarchical JPEG clearly requires much more processing than the other modes, and using multiple frames increases the amount of data that must be transmitted.

Lossless mode: The original JPEG standard defined a lossless compression mode that always preserves the exact, original image. A lossless mode could never hope to compress as well as a lossy one. Moreover, for most applications it did not compress as well as formats that were already available, so there was no compelling reason to use it. A new lossless compression method known as JPEG-LS has been created that, for all practical purposes, has made the original lossless format obsolete.

2.7.2. JPEG2000

JPEG 2000 is a Wavelet Based method, which in terms of compression methodology are very different than baseline JPEG. This was in response to growing demands for multimedia, Internet and a variety of digital imagery applications. The baseline JPEG surely is not optimized to efficiently code such a wide range of images. Moreover, scalability and interoperability requirements of digital imagery in a heterogeneous network of ATM, Internet, mobile etc., make the matter much more complicated. Each application area imposes a requirement that JPEG2000 should fulfill. Some of the most important features that this standard aims to deliver are:

Superior low bit rate performance: this standard should offer performance superior to the current standards at low bit rates, which should be achieved without sacrificing performance on the rest of the rate distortion spectrum.

Continuous tone and bilevel compression: it is desired to have a standard coding system that is capable of compressing both continuous tone and bilevel

images. If feasible, the standard should strive to achieve this with similar system resources.

Lossless and lossy compression: it is desired to provide lossless compression naturally in the course of progressive decoding.

Progressive transmission by pixel accuracy and resolution: progressive transmission that allows images to be reconstructed with increasing pixel accuracy or spatial resolution is essential for many applications. This feature allows the reconstruction of images with different resolutions and pixel accuracy, as needed or desired, for different target devices.

Region of interest coding: often there are parts of an image that are more important than others. This feature allows a user defined region of interest (ROI) in the image to be randomly accessed and/or decompressed with less distortion than the rest of the image.

Robustness to bit errors: it is desirable to consider robustness to bit errors while designing the code stream. One application where this is important is wireless communication channels. Portions of the code stream may be more important than others in determining decoded image quality. Use of error confinement, error concealment, restart capabilities, or source channel coding schemes can help minimize the effects of bit errors.

Open architecture: it is desirable to allow open architecture to optimize the system for different image types and applications. With this feature, the decoder is only required to implement the core tool set. If necessary, unknown tools are requested by the decoder and sent from the source.

Protective image security: protection of a digital image can be achieved by means of methods such as watermarking, labeling, stamping, fingerprinting,

encryption, scrambling….etc. Watermarking and fingerprinting are invisible marks set inside the image content to pass a protection message to the user. Stamping is a mark set on top of a displayed image that can only be removed by a specific process. Encryption and scrambling can be applied on the whole image file or limited to part of it (header, directory, or image data) to avoid unauthorized use of the image.

CHAPTER III

Vector Quantization

3.1. Definition of Vector Quantization

One of the competitive compression methods is VQ. It was first proposed as a quantization method for Linear Predictive Coding (LPC) parameters, and later was applied to waveform coding and image compression [1] [2] [3]. VQ is the general case of scalar quantization, instead of using one sample, a set of samples is used to be quantized each time. A vector quantizer Q of dimension k and size N is a mapping from a vector in k-dimension Euclidean space R^k into a finite set C containing N reproduction vectors called *codevectors*. The set C is called the *codebook*. It has a limited number of possible reproduction codevectors.

VQ is considered to be consisting of two processes, an encoding process and a decoding process. The encoder is a mapping from R^k to the index set, the operation of the encoder requires a choice of the mapping rule. The optimal encoder thus operates in a nearest neighbor or minimum distortion rule. The most commonly used distortion measure for image compression is the MSE. The encoder is optimal for a given codebook if it minimizes the average distortion. With full search VQ, the encoder determines the closest codevector in the codebook by an exhaustive search. A constrained search can speed up the encoding but may not be guaranteed to find the overall nearest neighbor in the codebook.

The decoder maps the index set into the reproduction set C, it is just a table look up. The operation of the decoder is thus completely described once the

codebook is specified. Since the same codebook is found in both the transmitter and receiver, it is sufficient to send only the indices as representative to the codevectors. The transmitted index has much fewer bits than those required for transmitting the codevector; the compression comes from sending the index rather than the whole codevector itself.

The application of VQ to image compression research activity expanded greatly. The earliest study, by Yamada, Fujita, and Tazaki [32], appeared in Japanese in 1980. Subsequently commercial video teleconferencing coders emerged in the U.S. and Japan based on VQ [1]. A natural way to apply basic VQ to images is to divide the image into non overlapping blocks of fixed size which then are converted into 1×k vectors. A typical gray digital image has a resolution of 8 bits per pixel. The goal of VQ compression is to reduce this bit rate without perceptible loss of picture quality.

For the same codebook size, VQ image quality mainly depends on generating representative codebook to the image statistics. Many algorithms are developed for codebook generation. LBG algorithm is the most cited one [12], PNN [13], genetic algorithms (GA) [33]-[36], simulated annealing (SA) [37], and Kohonen self organizing maps (SOM) [38] are commonly used in codebook generation. The challenge is to find the set of vectors such that the overall distortion between the actual image and the reconstructed image is as minimum as possible under a fixed rate.

The VQ compression ratio depends on the codebook size. If the image is composed of pixels with 8 bits each, and it is divided into L×L pixels blocks. When using codebook of size N then $\log_2$ (N) bits are needed to represent each index. The image block has L×L×8 bits. The compression ratio can be calculated from:

$$\text{compression ratio} = \frac{\mathrm{L} \times \mathrm{L} \times 8}{\log_2(N)} \tag{3.1}$$

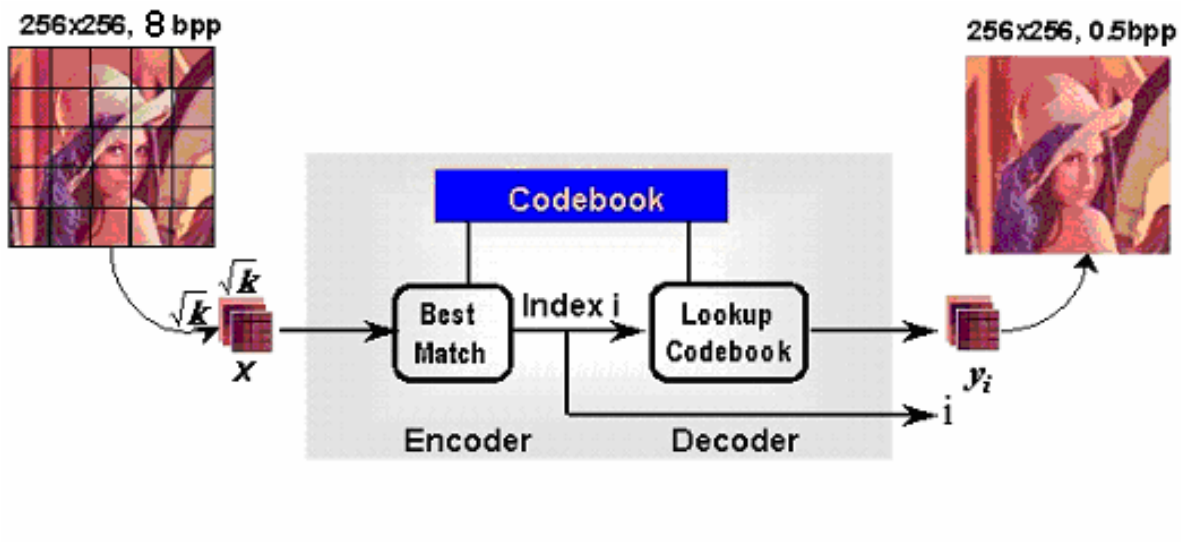


Fig (3.1): VQ schematic diagram

And the bit rate (in bpp)

$$BR = \frac{\log_2(N)}{L \times L} \tag{3.2}$$

While compression efficiency of VQ does not typically match that of Wavelet or DCT based compression schemes, it is used for image compression due to its simple decoder which is used for low complexity end user systems; also VQ variants can give high compression ratios with competitive peak signal-to-noise ratio.

Since VQ is block based algorithm, where the image blocks are replaced by reproduction blocks, the boundaries are not assured to be continuous as the original blocks and most of the distortion is found at the edges, this results in the so called blocking effect and the staircase effect. The blocking effect can be reduced by using weighted MSE as a distortion measure, where more weight to the pixels in the block boundary is given, but this will reduce the overall PSNR of the image. Another method to solve this problem is to use

blocks having shapes which does not have vertical or horizontal edges, like hexagon, where the eye is not sensitive to the non horizontal edges [39].

3.2. VQ Applications

While VQ is used mainly for image compression, it is also used as a signal processing tool; it could be used in classification, recognition, enhancement, edge detection, and many other applications [40]. VQ can be used as the front end to a signal processing systems. In such applications VQ can be viewed as a complexity reducing technique because it permits the signal processing to be replaced by simple table lookups.

VQ simplifies image processing tasks where the computational complexity is reduced by performing the processing tasks on the codevectors instead of the image vectors themselves. The original data are first compressed and stored as a list of codevector indices. The decoder reads the indices and generates the reconstructed image.

The key issues in combining processing with VQ are the size of the operational blocks and whether the blocks are processed in an independent non overlapping or dependent manner. The processor must be able to operate on blocks of the image that are no larger than the size of the codevectors. In addition the codevectors must be processed independently of each other or at least of the blocks which have not yet been decompressed.

At the time of codebook design each codevector is processed as if it were a block of an image. Each codevector is then linked to its processed result. The processing is done off line at the time of codebook design and the decoder codebook contains the codevectors and processed results linked together When the system decompresses an image the decoder can put out the codevectors, the processed result or both and no additional time is required.

This method has some disadvantages; first the decoder must store the processed codevectors which require varying amounts of storage space. Second if the processing step normally depends on the entire decompressed image, the results obtained by processing the codevectors off line would be different from the results usually obtained.

3.2.1. Thresholding

It operates on a grayscale image in a pixel-wise manner. The VQ is designed as usual and each pixel in each codevector is thresholded to produce the linked codebook. When the codebook is used for decompression the decoder can select the output from either the original or the thresholded codevectors. Thus a thresholded compressed image can be obtained for no post-processing costs.

3.2.2. Enhancement and Contrast Enhancement

VQ has the inherent ability to remove speckle noise because of the smoothing or averaging performed by the centroid operation during the codebook generation. Of course such smoothing can be considered as enhancement only if the speckle is indeed undesirable noise and not part of the signal.

Histogram equalization is a powerful tool for contrast enhancement that increases the dynamic range of images to bring out features hidden in dark regions or washed out by light areas. Histogram equalization remaps each pixel to intensity proportional to its rank among surrounding pixels. Instead of performing the decoding and equalizing operations sequentially, one can perform them simultaneously by equalizing the decoder codebook off line at the time of codebook generation. The intensity distribution of the training set must be very similar to that of the test set for a VQ codebook to work well and the histograms would likely be quite similar.

3.2.3. Halftoning

Halftoning is the conversion of a grayscale image to a binary image suitable for display on a binary device. A combination of compression and halftoning would be useful for transmitting images by facsimile, transmitting images to printers, and storing images for display on monochrome monitors.

3.2.4. Classification and Analysis

Scene analysis and image understanding range from character recognition and medical image analysis to automatic defect analysis and cartography. Most such problems require examination of blocks considerably larger than a VQ vector but the algorithms usually begin with low level operations on small blocks. VQ can be used for this low level classification or detection and the low level classification itself may be useful.

Combining VQ with classification is natural because both techniques can be designed and implemented using methods from statistical clustering and classification trees. The goal of such a combination is to incorporate classification information into the codevectors by classifying the codevectors themselves during codebook design. By combining VQ and low level classification certain simple features in an image can be classified automatically as part of the compression process

3.2.5. Edge Detection

Variable rate VQ has some inherent edge detecting properties. A variable rate tree structured VQ usually uses more bits for regions of greater activity such as edges and fewer bits for homogeneous or inactive regions. The number of bits allocated to a particular block provides information about the block activity.

3.2.6. Related Work in Speech Processing

VQ has been extensively used for speech compression [41] but it can also be used for other speech processing tasks including speech and isolated word recognition, speaker recognition and verification and noise suppression. VQ has found wide use in speech recognition systems as a front end but it can also be used as a pattern matching technique to perform all or part of the recognition itself

3.3. Codebook Design

The main design problem in VQ is to generate a codebook that covers all the image statistics. The reconstructed image quality at fixed bit rate depends on the used codebook. There are many approaches for codebook design, the generalized Lloyd algorithm (GLA), which is sometimes referred to as LBG algorithm or k-means algorithm, is the most cited one [12]. The GLA is a descent algorithm run on a training set, and it can always be used to improve a codebook in terms of reducing distortion.

Many other clustering algorithms have also been applied to VQ design, including PNN method [13], Kohonen's SOM [38], simulated annealing [37], and deterministic annealing [42].

3.3.1. Random Codebooks

The simplest approach to generate a codebook of N codevectors is to randomly select the codevectors from the image blocks [1]. The obvious variation when designing based on a training sequence is to simply select the first N training vectors as codevectors. If the data is highly correlated, it will likely produce a better codebook if one selects as spatially separate training vectors as possible. Unfortunately, the codebook will not be useful, since it is

not even local optimal. LBG iterations should be applied on that codebook to improve it.

3.3.2. LBG

It iteratively improves an initial codebook by alternately optimizing the encoder for the decoder using the nearest neighbor condition (For a given set of codevectors, each training vector is mapped to its nearest codevector in respect to the minimum distortion.) and the decoder for the encoder using the centroid condition (For a given partition, the optimal codevector is the centroid of the vectors within the partition.). The centroid is the Euclidean mean of the input vectors mapping into a given index. Codebook design is based on a training set of typical data rather than on mathematical models of the data. It can always be used to improve a codebook in terms of reducing distortion. On the basis of these two optimality conditions, the LBG algorithm is described as follows:

1. Start with an initial set of reconstruction values Y= {Yi; i=1: N}.
2. Divide the training set into partitions by mapping each vector X to its nearest codevector Y using the Euclidean distance. We assume that none of the quantization regions are empty.
3. Calculate the centroid of each region and replace the codevectors Y by the centroids of their corresponding partitions.

The algorithm iterates the optimality conditions, thus the resulting codebook after each iteration can never be worse than the previous one. The iterations are thus continued until the decrease in the overall distortion is less than a predefined threshold or for certain number of iterations. The algorithm doesn't necessarily reach the global optimum, but converges to a local minimum.

The main disadvantage of LBG is that the iterative algorithm finds the nearest locally optimal codebook in respect to the initial one, Choosing the initial values of the codebook is critical to the quality of the quantizer. The initial codebook for LBG can be constructed by any existing codebook design method, or by the random heuristic.

3.3.3. Pruning

In the pruning algorithm [1], the codebook generation starts with the training set as candidates to be codevectors, and selective eliminating (pruning) training vectors as candidate code vectors until final set of training vectors remains as the codebook. In such method a sequence of training vectors is used to populate a codebook recursively. First, put the first training vector in the codebook, and then compute the distortion between the next training vector and the first codevector. If it less than some threshold, continue. If it is greater than the threshold, add the new vector to the codebook as a codevector. With each new training vector, find the nearest neighbor in the codebook. If the resulting distortion is not within some threshold, add the training vector to the codebook. Continue in this fashion until the codebook has fewer codevectors than the desired number of codevectors. If this happens, the threshold value must be reduced and the process repeated.

3.3.4. The Pairwise Nearest Neighbor

The PNN algorithm [13] starts by including all the L training vectors into the codebook. In each step, two codevectors are merged into one codevector so that the increase in the overall distortion is minimized. The algorithm is then iterated until the number of codevectors is reduced to the desire codebook size. The first step is to finding the candidate code vector to be merged by calculating the distortion between all pairs of training vectors. The two nearest

neighboring vectors are combined into a single cluster and represented by their centroid. The input space consists now of L-1 clusters, one containing two vectors and the rest containing a single vector. At every iteration of the algorithm, the increase in average distortion is computed for every pair of clusters, resulting if the two clusters and their centroids are replaced by the merged two clusters and the corresponding centroid. The two clusters with minimum increase in the distortion are then merged, and the codevector of that cluster is its centroid. This is also a form of pruning. It begins with the entire training sequence of L vectors, and ends with a collection of N vectors.

3.3.5. Product Codes

Product codebook [1] may provide a good initial guess. When designing a codebook for a k-dimensional VQ with codebook size N. General product structures can be used, first a one-dimensional (scalar) quantizer q with N levels is designed. The scalar quantizer is used then k times as an initial guess to design a good k-dimensional quantizer, one scalar quantizer for each dimension in the VQ.

3.3.6. Splitting

The splitting algorithm [1] has the opposite approach to codebook construction compared to the PNN method. It starts by a codebook with a single codevector which is the centroid of the complete training set. The algorithm then produces increasingly larger codebook by splitting certain codevector Y to two codevectors Y-e and Y+e, where e is a vector of small Euclidean norm. The codevector that causes the largest distortion is selected to be splitted. The new codebook after the splitting can never be worse than the previous one, since it is the same as the previous codebook plus one new codevector. The algorithm is iterated until the size of codebook reaches N.

3.3.7. Stochastic Relaxation

Introducing randomness into each iteration of the LBG algorithm makes it possible to evade local minima, reduce or eliminate the dependence of the solution on the initial codebook, and locate a solution that may actually be a global minimum of the average distortion as a function of the codebook. In the stochastic relaxation algorithm [43] each iteration of a search for the minimum of the average distortion consists of perturbing the codebook in a random fashion. The magnitude of the perturbations generally decreases with time, so that convergence is achieved. A key feature of SR algorithm is that increases in the value of the average distortion are possible in each iteration. Introducing randomness into each iteration could be done using two methods. In the first method noise is added to the training vectors prior to the LBG iterations and the variance of the noise is gradually decreased to zero. In the second method, after each centroid calculation, zero-mean noise is added to each component of each codevector. The variance of the noise is reduced with successive iterations according to a predetermined schedule.

3.3.8. Simulated Annealing

Simulated annealing is a stochastic relaxation technique in which a randomly generated perturbation to the codebook at each iteration is accepted or rejected with probability, where the probability depends on the change in value of the average distortion resulting from such a perturbation. Specifically, let H(C) denote the average distortion for a nearest neighbor quantizer as a function of the codebook C. In each iteration a candidate perturbation of the partition is randomly generated and then either accepted or rejected with a probability that depends on the average distortion of the new codebook. If the distortion decreases, then the change is always accepted. If the distortion increases it is accepted with probability P. If a sufficient number of iterations are made for a

given variance, then the code book that is globally optimal or very nearly so have very high probability. Several reports of successful use of simulated annealing for VQ codebook design have been reported.

3.3.9. Fuzzy Clustering (Deterministic Annealing)

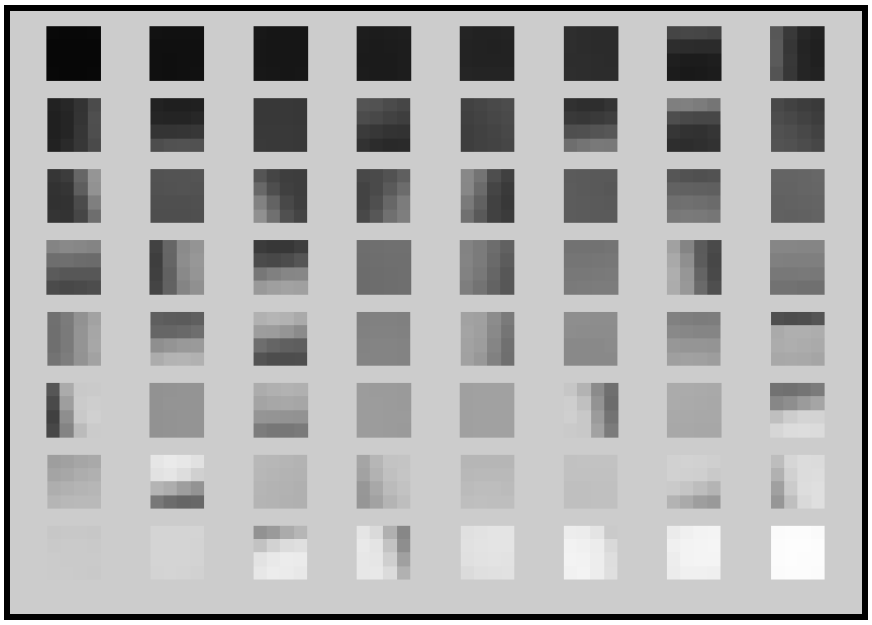
Deterministic annealing appears to capture the benefits of SA for VQ codebook design without any randomness in the design process. It is conceptually similar to the technique of fuzzy clustering [44]. In designing a VQ codebook from training set, when calculating the centroid of a cluster (partition region) and for the average distortion, respectively, both use a membership functions, has value one if the training vector, is a member of the cluster and value zero otherwise. A cluster is said to be a fuzzy set if we may assign to each element of the training set a degree of membership or partial membership value between zero and one which indicates to what extent the particular vector is to be regarded as belonging to that cluster. In this way, we define a fuzzy distortion to contain a parameter that controls the fuzziness of the distortion. It is then simple to calculate the average distortion where the contribution of each training vector is weighted by its degree of membership in the cluster. In the deterministic annealing, the statistical contribution of the randomness (that would arise in a corresponding simulated annealing scheme) is incorporated into the average distortion calculations.

3.4. Codebook Issues

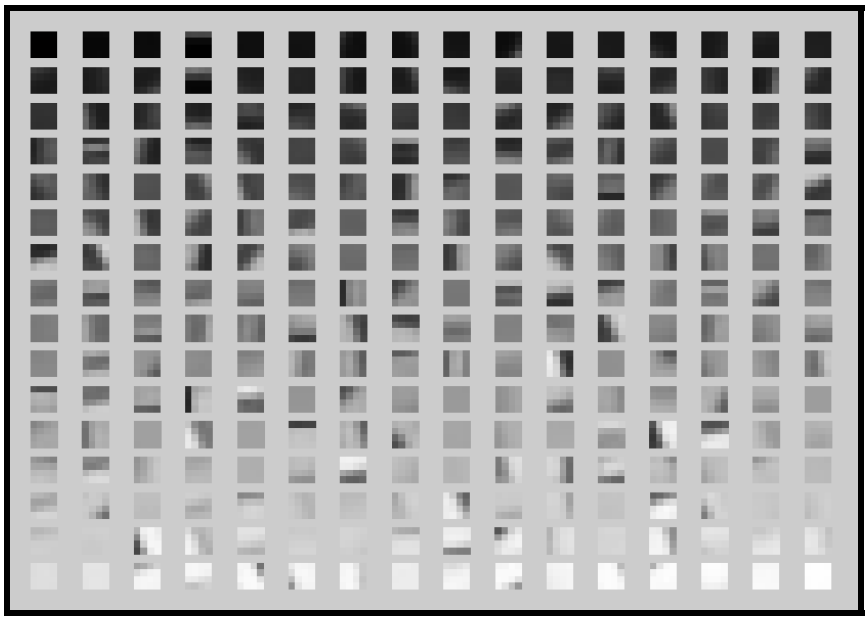
3.4.1. Size of Codebook

Codebook size (N) is an effective factor in both the quality and the bit rate of the VQ. When increasing the codebook size, we cover a more wide range of image block statistics, and hence improve the reconstructed image peak signal to noise ratio. Also this increase in codebook size will increase the bit rate.

This conclusion is clear from Fig (3.2) which shows an example of two codebooks with sizes 64 and 256. Fig (3.3) shows the reconstructed image of the two codebooks.



a)



b)

Fig (3.2): The codevectors of codebook of size
a) N=64, BR=0.375 (bpp) b) N=256, BR=0.5 (bpp)

3.4.2. Selection of Block Size

VQ can perform arbitrarily close to the theoretical optimal performance for a given rate if the vectors have sufficiently large dimension, see Fig (3.4) for "Lena" image that vector quantized with different block sizes at the same bit rate. As seen from the Fig (3.5) when increasing the dimension of the block, it contains more data and adjacent blocks become less correlated. For a certain bit rate when increasing the block dimension the codebook size increases and that allows covering all image statistics. Unfortunately computations complexity grows exponentially with vector dimension.

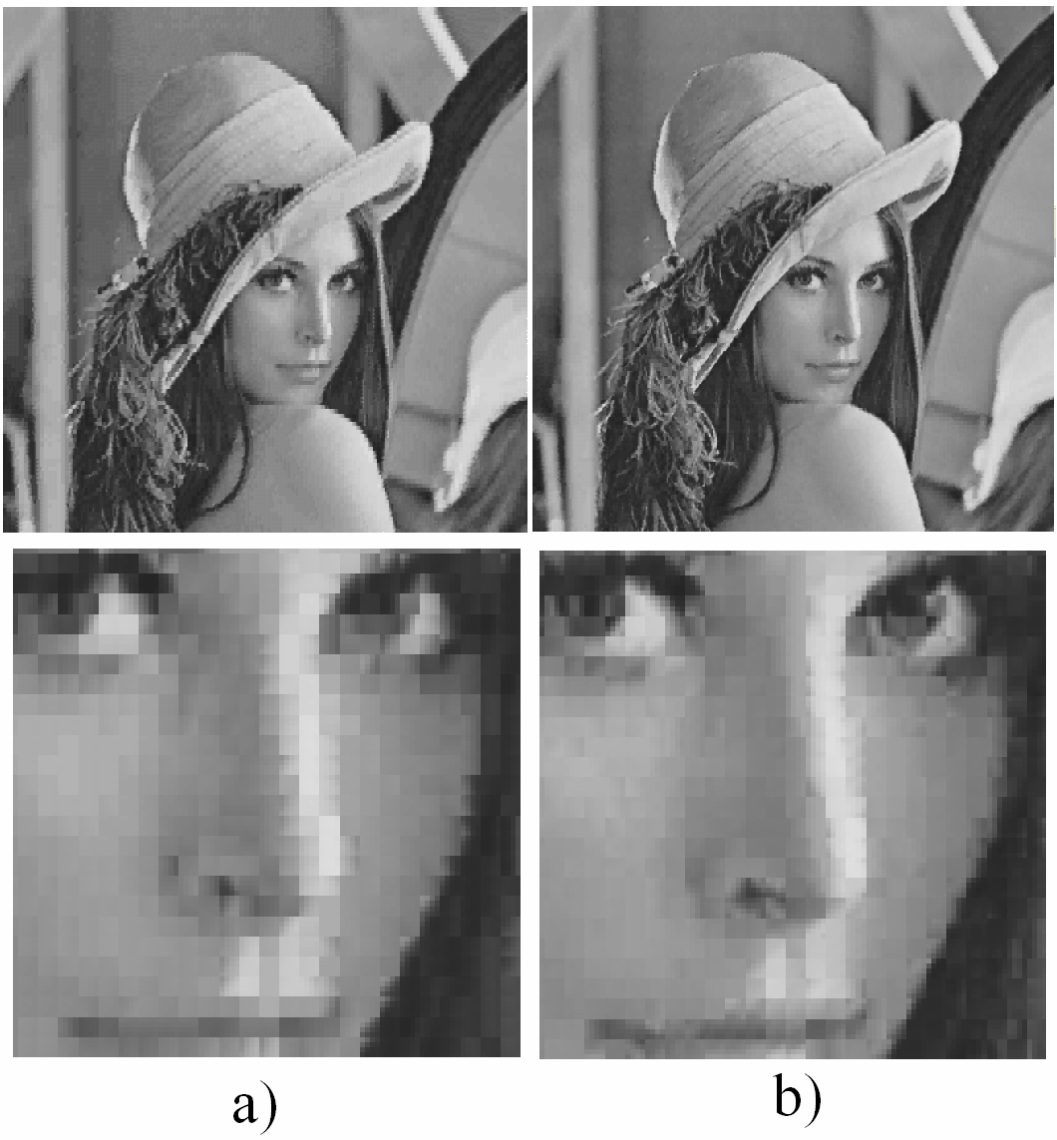


Fig (3.3): VQ reconstructed image using codebook sizes a) N=64 b) N=256

Table (3.1): Relation between codebook size block size and bit rate

Block size (k)	Codebook Size (N)	Index Length (bits)	BR (bpp)
2x3=6	64	6	1
	512	9	1.5
	4096	12	2
3x4=12	64	6	0.5
	512	9	0.75
	4096	12	1
4x4=16	64	6	0.375
	512	9	0.5625
	4096	12	0.75
4x6=24	64	6	0.25
	512	9	0.75
	4096	12	0.5

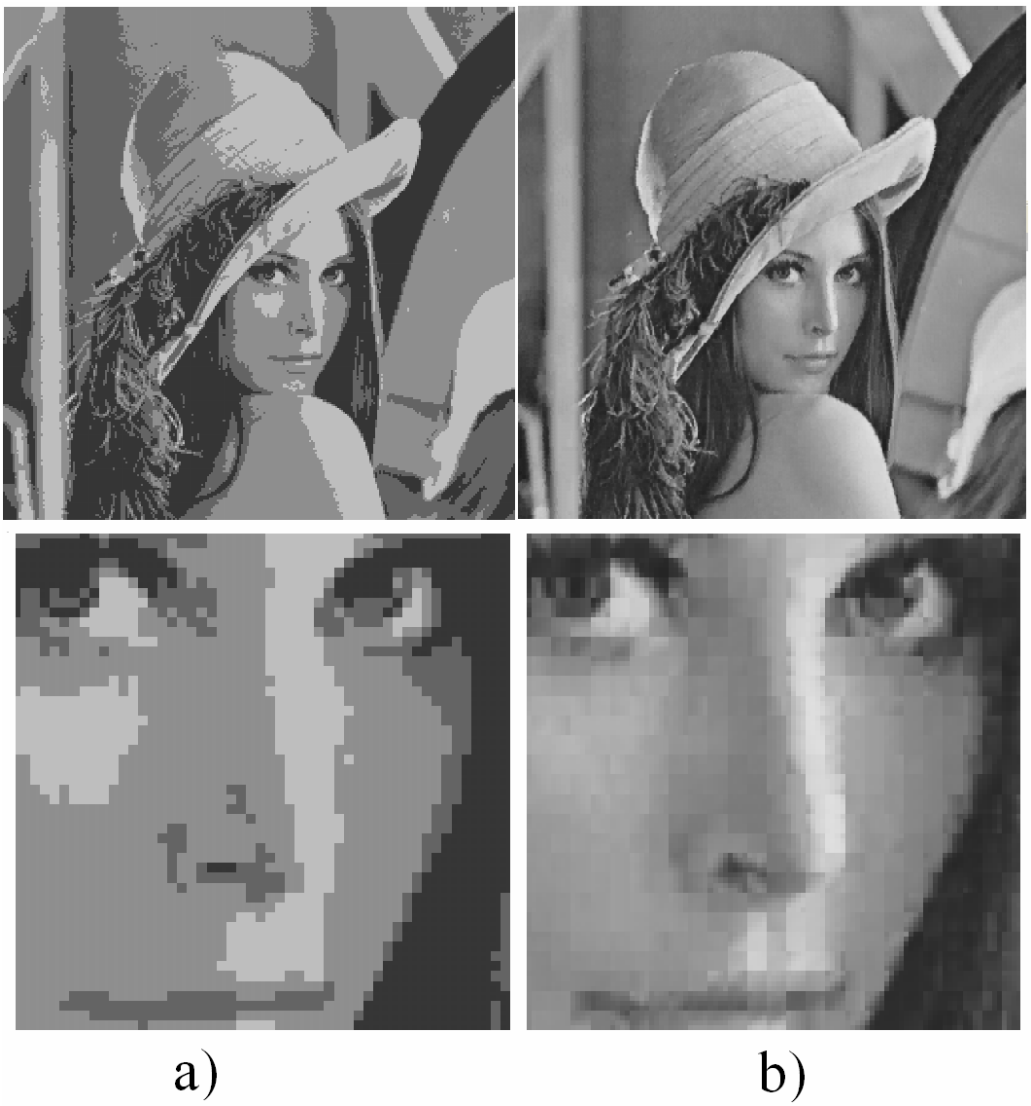


Fig (3.4): reconstructed image at bit rate = 0.5 with block size a) 2×2 b) 4×4

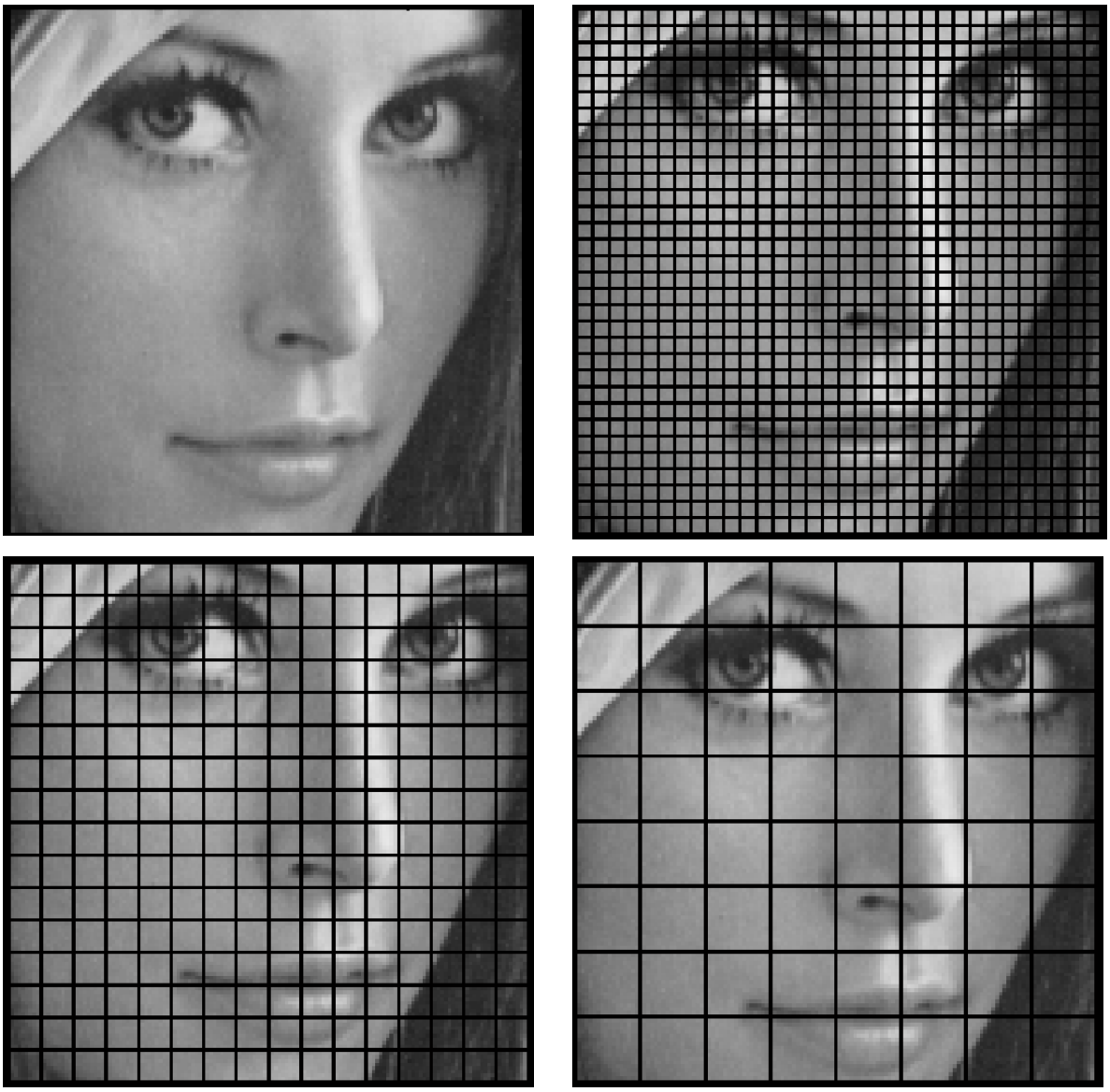


Fig (3.5): Image of Lena, original and divided into different block sizes 4×4, 8×8, and 16×16

3.4.3. Local codebooks versus global codebooks

VQ can be applied by designing a codebook with the image to be compressed itself as the training set. The codebook, however, must then be included in the compressed file. For example, consider a codebook of 256 entries each requiring 16 bytes. The complete codebook thus requires 256×16= 4096 bytes of memory increasing the overall bit rate by 0.125 bits per pixel (in the case of 512×512 image). Better results are obtained when the codebook is generated from the image content itself (local codebooks).

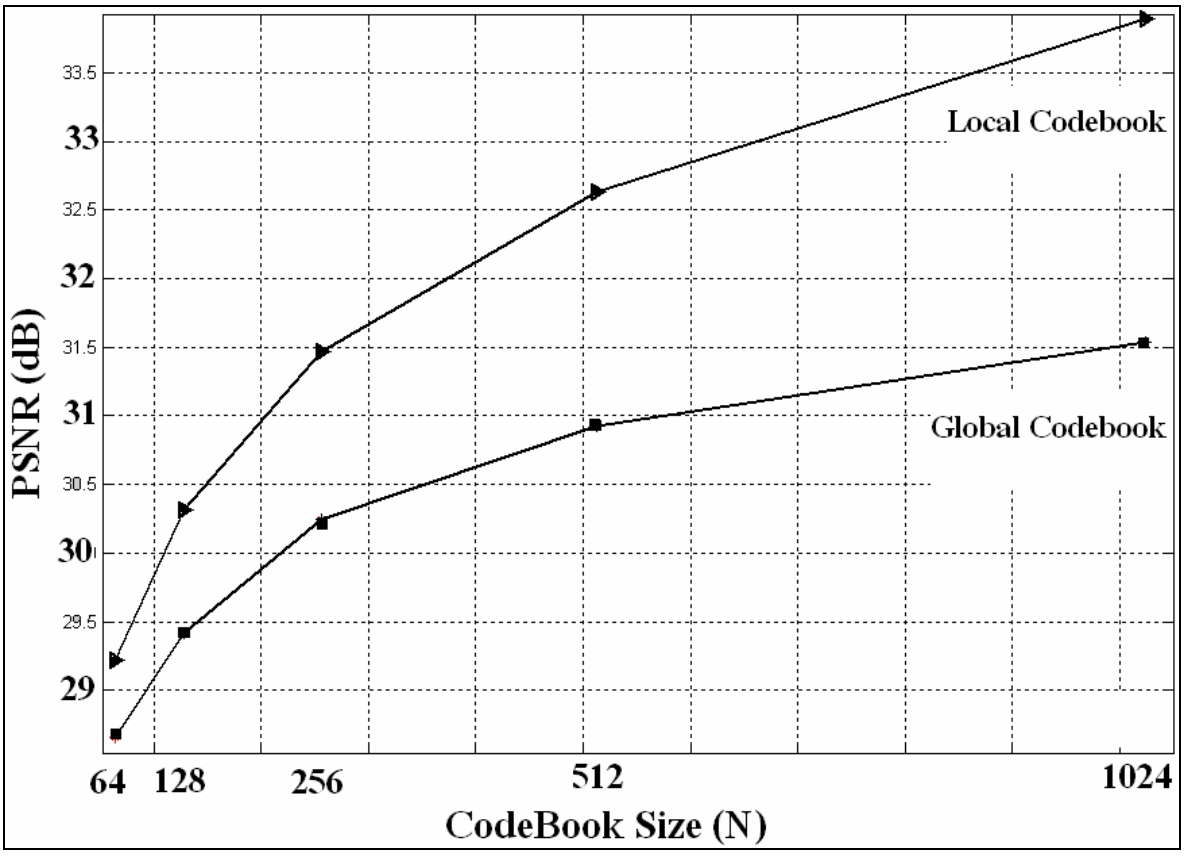


Fig (3.6): PSNR for different codebook sizes using local codebook and global codebook for image Lena

3.5. Fast Encoding Algorithms

To find the best matched codevector in the codebook to the input vector, we need a matching criterion and a codebook search algorithm. The most popular matching criterion is the Euclidean distance. Basic VQ employ the Full Search Algorithm (FSA) which calculates the Euclidean distance between the input vector and every codevector in the codebook. The codebook search is very computationally intensive. The encoding time complexity (average number of operations per input vector) needed for full search algorithm with Euclidean distance measure is $k \times N = k \times 2^{kR}$, where k and R represent the vector dimension and the resulting bit rate, respectively. The time complexity grows exponentially with vector dimension and bit rate, which severely restricts the size of codebooks. While many of constrained VQ methods offer reduced

encoding complexity, several methods have been developed for reduced complexity (i.e. fast) encoding while retaining a given unstructured codebook [45]-[52]. The objective is to find the nearest neighbor codevector to a given input vector without incurring the complexity of full search through the codebook. The algorithms can be roughly categorized as lossy or lossless approaches. The former imposes structural constraints on the codebook to reduce the computation. However, the performance compared to FSA is degraded due to both the suboptimal search procedure and suboptimal codebook. The latter achieves the same quality of encoded images as FSA.

It should be noted that the relative advantage of a particular technique is generally implementation dependent. In particular, a programmable signal processor chip may require an entirely different algorithm than an application specific integrated circuit. Furthermore, unlike exhaustive search which has a fixed search time, many techniques have a random search time. In some applications, such as image compression for storage, the average search time is a more important indicator of performance than the worst case search time. On the other hand, the reverse is true in some other applications such as speech coding for telecommunications.

3.5.1. The Partial Distance Elimination (PDE) Method

One of the simplest methods for fast search, called the partial distance elimination method, can typically reduce the average search time by a factor of 4 and requires only a small modification of the exhaustive search nearest neighbor encoding rule and is based on the squared Euclidean distortion measure. For a given input vector x in k-dimensions, the search proceeds sequentially through the codebook and after the m^{th} codevector has been tested, the minimum distortion so far (D_{min}) and the best index so far (I) are retained. Then to test the $(m+1)^{th}$ codevector, the distortion is computed

cumulatively; the partial distortion is compared with D_{min} after each dimension's distortion. If the partial distortion exceeds D_{min} then reject this codevector and go to the next codevector, else add the distortion of the next dimension to the partial distortion. If the current vector total distortion is less than $D_{min,}$ then update D_{min} by the current calculated distortion and go to the next codevector.

The validity of this algorithm is self evident. Also, it should be noted that the algorithm always results in locating the true nearest neighbor since no approximations are made. The worst case search time is no better than exhaustive full search although it would be virtually impossible for the worst case to occur. The utility of this algorithm is limited to applications where the time incurred in a comparison operation is relatively small compared to a multiplication operation.

3.5.2. Projection Method

One general approach to reduce the encoder complexity is based on identifying the geometric boundaries of the Voronoi cells and storing a suitable data structure to ease the real time search process. Much of the encoding complexity is transferred from the real time encoding operation to the off-line design of the data structure. An example of such a technique is the projection method, where the data structure is obtained by finding for each Voronoi cell the boundaries of the smallest hyper rectangular region in k-dimensional space that contains the nearest neighbor cell. Specifically, the minimum and maximum values of the J^{th} coordinate of points in cell R_i are found by projecting the cell onto the J^{th} coordinate axis. This gives for each cell a set of 2N hyperplanes (each orthogonal to one of the coordinate axes) that specify the faces of the bounding hyper rectangle. This provides a partition of each coordinate axis into 2xN-1 intervals. A table is then

generated for each coordinate indicating which Voronoi cells have a projection that lies in each interval. Various ways of organizing the table are possible. The search procedure then scalar quantizes one component at a time of the input vector and identifies the set of candidate cells. As successive components are quantized then the intersection of the prior set of candidates with those associated with the current components is formed until all components have been examined. Then, an exhaustive search is performed over the final set of candidates, which is much smaller in number than the original codebook size N. In principal this method will always yield the nearest neighbor, however, the task of finding the exact projections is extremely demanding and appears to be computationally infeasible. In practice, the projections can be found by encoding a large training set, storing the clusters so obtained and allowing the projection of the i^{th} cluster to substitute for the projection of the cell itself.

3.5.3. K-d Trees Method

A K-d tree is a binary tree with a hyperplane decision test at each node where each hyperplane is orthogonal to one of the coordinate axes of the k-dimensional space and a set of terminal nodes. The search process is simplified since each node requires the examination of only one component of the input vector to see if it lies above or below a threshold. The K-d tree can be very efficient for codebook search but it requires an effective design procedure based on training data. The projection method is actually a special case of a K-d tree.

3.5.4. The Triangle Inequality Elimination (TIE)

The multiple triangle inequalities confine a search range using the intersection of search areas generated from several control vectors. The idea is to use some

control vectors as reference points from which the distance to each codevector is pre-computed and stored. The encoder then computes the distance between the input vector and each control vector after which some simple comparisons using the pre-computed data eliminate a large number of codevectors as candidates for the nearest neighbor. A variety of refinements have emerged to make this approach an efficient method for codebook searching. In general, there is a trade-off between the search speed and the size of the pre-computed data table required.

3.6. Constrained Structure VQ

VQ has excellent rate-distortion performance; it can reach the rate distortion bound when increasing the vector dimension at fixed bit rate [1]. The performance improvement with increase in vector dimension can be explained by recognizing that we are exploiting longer term statistical dependency among the image pixels and higher block shapes. Using large blocks increases the memory requirements and complexity costs, since they are exponential in block size. Unconstrained VQ is limited to modest vector dimension and codebook sizes for practical systems. But using small dimensions sacrifices the efficient exploiting of all the statistical dependency between adjacent pixels. Many constraints are applied to VQ structure that will allow using higher vector dimensions. The best codebook is that which matches the probability distribution of the signal, but in constrained VQ, the codebook is not generated optimally for the training set but is restricted in distribution for a certain manner. The codevectors cannot have arbitrary locations as points in k-dimensional space but they are distributed in a restricted manner that either allows an easy search for the nearest neighbor or else allows for a search that does not necessarily produce the nearest neighbor. Almost all of these constrained VQ give compromise performance achieved by unconstrained VQ

but give a trade of between performance and complexity. The imposed structural constraints return computation savings, a memory savings, or sometimes both. Constrained VQ can be designed for rate that is not practical for ordinary VQ.

3.6.1. Product Code VQ

When the VQ dimension is large to be practically used, the complexity is reduced by converting the large dimension vector into a set of reduced dimension vectors. The simplest form of product code VQ is the portioned VQ, where the reduced dimension vectors results from dividing the large vector into lower dimension subvectors, separate VQ is needed for each sub vectors. That is the encoder generates set of indices, one for each VQ, at the decoder the set of indices is used to reconstruct the set of subvectors then reconstruct the original vector. The disadvantage of using this method is that when dividing the large dimension vector into subvectors, the statistical dependency between adjacent pixels is lost and the quality is reduced.

The solution to this is to convert the vectors into independent components. There are two cases where the subvectors are independent, the removed mean VQ and the shape gain VQ.

3.6.2. Removed Mean VQ

The blocks of an image do not have zero mean. The mean of the vector varies from vector to another and is independent on the vector shape. Thus the vector can be decomposed into two features the mean and the shape, where the mean is subtracted from the individual pixel values to form the shape vector. The mean is a scalar quantity and the shape is a vector quantity with the same dimension of the original vector. Now we have to quantize two components; the mean value of the pixels which could be coded using DPCM, and

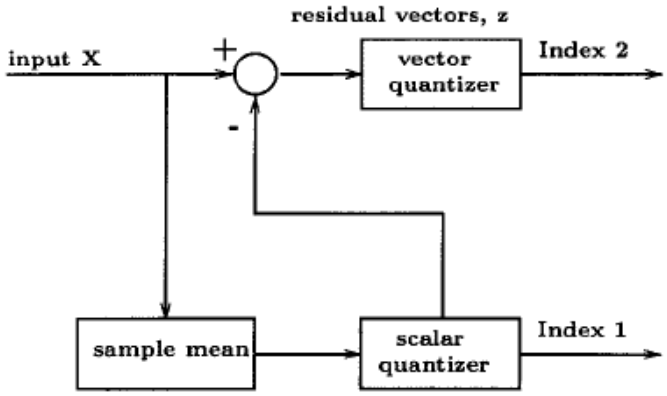


Fig (3.7): Removed mean VQ block diagram

the shape vector which is vector quantized. It is easier to design a codebook for the shape vectors than for the original vectors, since the shape vectors have fewer shapes than the original vectors, so we can use codebook with small size. The product codebook is suboptimal because it imposes a structural constraint on the codebook, but in practice it can improve performance by allowing larger dimensions for the shape codebook. Fig (3.8) shows the Lena image decomposed into mean values and shape vectors.

a) b)

Fig (3.8): Lena image a) shape vectors (512x512 pixels) b) mean values (128 x128 block)

3.6.3. Shape/Gain VQ

Other form of product code VQ is shape/gain VQ. In shape/gain VQ, the input vector is divided to the root mean square (rms) value of the vector components. The rms is called the gain and it is a scalar value. The normalized vector is called the shape of the vector which is the correlations between the neighboring pixels. The shape is a vector with the same dimension as the original vector. The probability distribution of the gain is independent on the shape vectors. The shape is then coded by vector quantization. Other scalar coding methods such DPCM can be more suitable for the gain. Fig (3.9) shows the Lena image decomposed into gain values and shape vectors.

a) b)

Fig (3.9): Lena image a) shape vectors (512x512 pixels) b) gain values (128x128 blocks)

3.6.4. Transform VQ

Using an orthogonal linear transform, we can transform the input vector into another vector which has some certain advantages than the original vector. The VQ is applied to the transformed vectors instead of the original vectors.

At the receiver the reconstructed vector is inverse transformed to obtain a reconstruction of the original input vector. If the VQ dimension is the same as the transform dimension, then the optimal codebook for the transform VQ is the transform of the codebook of the ordinary VQ with the same dimension [1]. The average distortion is the same of the transform VQ and the ordinary VQ if they have the same dimension, but it increases the complexity slightly, which comes from the transform and the inverse operations. The advantage of using transform VQ is to compact the information of a large vector into reduced size vector, transforms such as DCT do this. This allows using large block dimensions than could be used in ordinary VQ where many components are discarded. Also the transformed vector can be divided into sub vectors depending on the features in each part of the vector, and each of them is quantized separately. Bit allocation can be applied to the different coefficients. This cannot be done in ordinary VQ. Fig (3.11) shows "Lena" Image where each block of 8x8 pixels is transformed to the shown DCT coefficients.

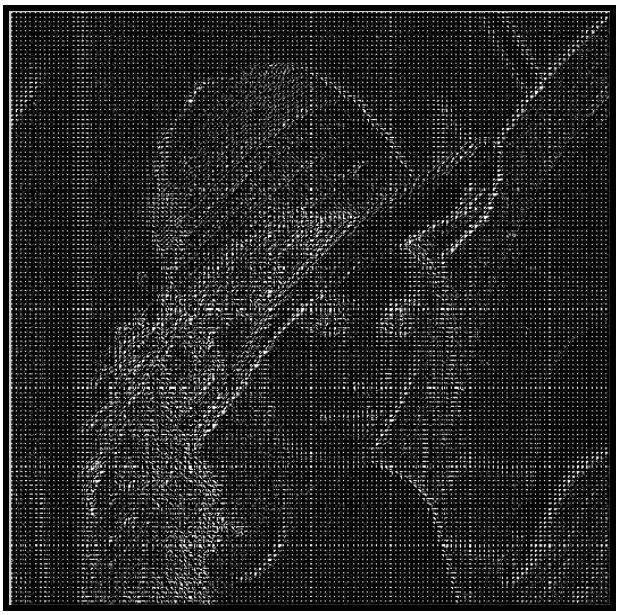
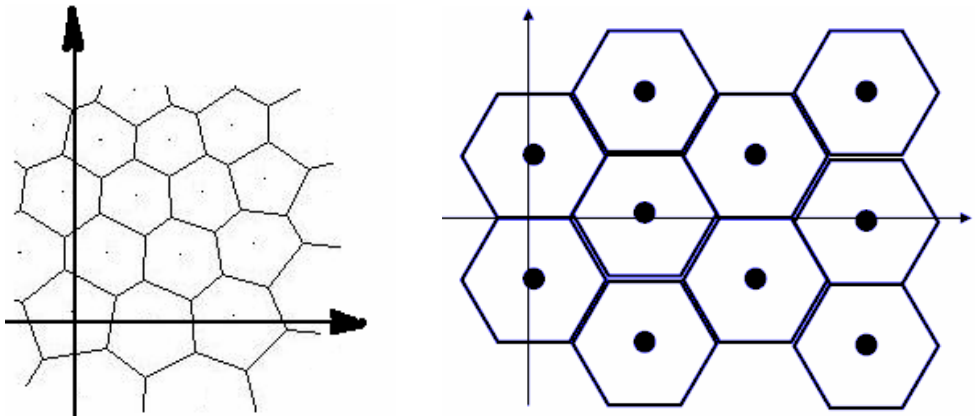


Fig (3.10): DCT coefficients vectors for Lena Image

3.6.5. Subband VQ

In subband coding, the image is decomposed into different subband signals [53] [54]. Each of them is coded separately depending on its importance in signal reconstruction. The biorthogonal wavelet is most used for image decomposition. The image is decomposed into approximation, horizontal, vertical, and diagonal images. Each of them is vector quantized. Note that the approximation can be further decomposed. The approximation is important in good reconstruction of the original image since it contains most of the power.

3.6.6. Lattice Codebook VQ



a) b)

Fig (3.11): a) 2D regular codebook b) 2D lattice codebook

A lattice in R^k Euclidian space is composed of all integral combinations of a set of linearly independent vectors, a lattice quantizer is a uniform quantizer with a (theoretically) infinite number of levels, where the entire real line is partitioned into a countable set of equal size intervals. 2D lattice is a set of points with a regular arrangement in the plane, see Fig (3.12). This concept generalizes to N dimensions. Lattice VQ has fast quantizing and decoding

algorithms [1]. For each cell in the region of support of a lattice VQ, the reproduction vector is taken to be either the midpoint of the cell or else the centroid of the training vectors laying in that cell.

Unfortunately, however, lattice quantizers have two inherent shortcomings. The first is that their structure can prevent them from being optimal in the sense of minimizing average distortion for a given rate except for Sources uniformly distributed over some region of space. The uniform distribution of reproduction vectors is well matched to uniformly distributed source vectors, but it is sub-optimum for non-uniform distributions. The second problem with general lattice codebook VQ is the implicit assumption of very large resolution; the codes remain primarily suited for large bit rate and small distortion applications.

3.6.7. Tree Structured VQ (TSVQ)

In the Tree Structured VQ (TSVQ) the codebook take the form of a branched tree and the search is preformed in levels, in each level a sub set of candidate code vectors are eliminated. In the *m*-ary tree search, the input vector is compared with m pre-designed test vectors at each level or node of the tree. The nearest test vector determines which of m paths through the tree to select in order to reach the next level of testing. The codevectors are located in the terminal nodes, i.e. the final nodes in the in tree. At each level the number of code vectors is reduced to 1/m of the previous set of candidates. In many applications m=2 and we have a binary tree. Although the resulting codevectors is not optimal, it gives performance near to the ordinary VQ while allowing low complexity encoding algorithm than exhaustive search through unconstrained codebook of the same size.

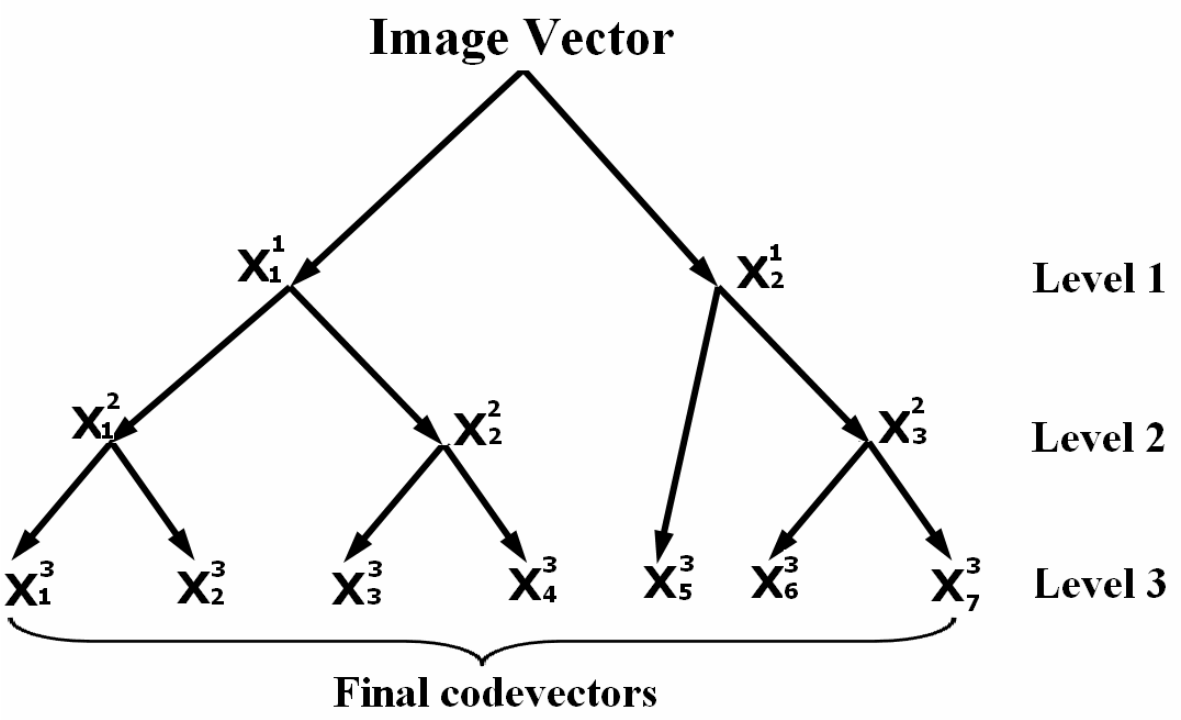


Fig (3.12): Tree structured codebook; level 3 is the terminal node.

Compared with a full-search unstructured VQ, the search complexity of a balanced tree is linear in the bit rate instead of exponential but at the cost of a roughly doubled memory size.

The decoder is the same as the ordinary VQ and does not need the tree structure. A disadvantage of TSVQ is the storage requirements, also the best match is not necessarily found because the search is made heuristically on the basis of the search-tree.

When the tree has the same number of branches at each node, then it is balanced tree. Beside the balanced tree there are non balanced trees such as:

Tapered tree: - where the number of children per node increases as one moves down the tree

Pruned trees: - where we eliminate the code vectors that contribute little to distortion reduction.

Unbalanced tree yielding a variable rate code, the advantages of variable rate TSVQ are that it usually yields lower distortion than fixed rate full search VQ

for a given average rate and block size. It also has a natural successive approximation progressive property and is well matched to variable rate environments such as storage or packet communications. In this case the tree should be provided to the receiver where the bit sequences can be transmitted (encoded) in progressive manner from the most significant bits of the blocks to the least significant bits, so that at the first phase the bits of the first branches in the tree structure are sent. Thus the decoder can display the first estimate of the image as soon as the first bits of each block are received. This kind of progressive coding consists of as many levels as there are bits in the code indices. The progressive coding is an option that requires no extra bits in the coding.

3.6.8. Multistage VQ

Multistage VQ divides the encoding task into several stages. The first stage performs a relatively crude encoding of the input vector using a small codebook. Then, a second-stage quantizer operates on the error vector between the original vector and the quantized first stage output. The quantized error vector provides a refinement to the first approximation. At the decoder, the reproduction vectors produced by the first and second stages will be added together. Additional stages of quantizers can provide further refinements. Unlike full search VQ, whose encoding complexity and memory requirements increase exponentially with the dimension rate product, in multistage VQ, the increase is only linear [1].

Multistage VQ is sometimes referred to as residual VQ, or cascade VQ. Like TSVQ, this scheme can be used for progressive reconstruction of the image.

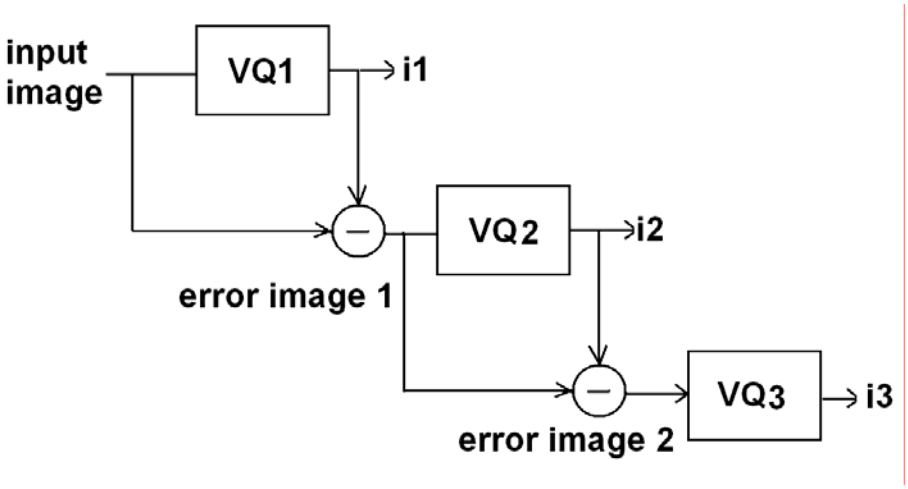


Fig (3.13): Multistage VQ

3.6.9. Classified VQ

In classified vector quantization [15], instead of having one codebook, several smaller codebooks are used. For each block, a classifier selects the codebook where the search is performed. Typically the codebooks are classified according to the shape of the block so that codevectors having horizontal edges might be located in one codebook, blocks having diagonal edges in another, and so on. The encoder has to send two indices to the decoder; the index of the chosen codevector within the codebook, but also the index of the class where the codevector was taken. The classified VQ can be seen as a special case of tree structured VQ where the depth of the tree is one level.

There are two motivations for classified VQ. First, it allows faster search from the codebooks since the codebooks can be smaller than the codebook of a full search VQ. Second, the classified VQ can also be seen as a type of codebook construction method where the codevectors are obtained due to their type (shape). However, the classified VQ codebooks in general are no better than full search codebooks. By choosing the full search codebook as union of all the classes' codebooks of classified VQ, the result cannot be any worse than the classified VQ. Thus, the primary benefit of classified VQ is that it allows a

faster search (at the cost of extra distortion). Varying size class codebook is allowed depending on the image statistics and that result in variable rate VQ. The classifier can be based on texture properties, edge directions, features of diagnostic or scientific importance or irrelevance, perceptual masking, or a variety of other criteria to select the class codebook. For the image, the class codebooks can be classified into edge and gray vectors with the edge vectors having many orientations. The classifier could be designed heuristically or using the LBG with the feature vector as the training set. The class codebooks are designed regularly with each class having its own set of training vectors

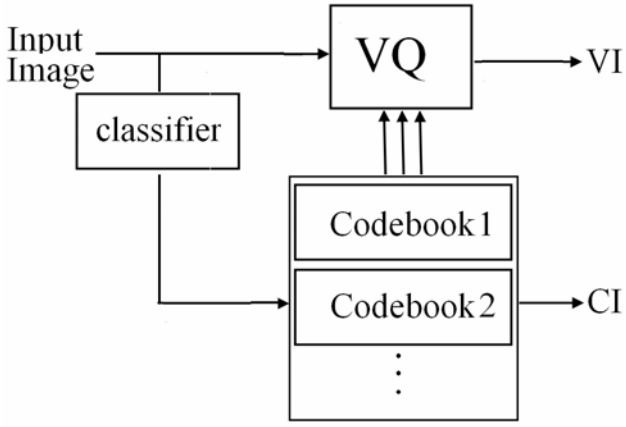


Fig (3.14): Classified vector quantization

3.6.10. Finite State VQ (FSVQ)

Like a classified VQ, FSVQ has multiple codebooks [55], and the next vector to be encoded is referred to the appropriate codebook. Unlike classified VQ, however, no side information is sent to the decoder to inform it of which codebook to use. That information is inferred using the next-state rule. The next-state rule uses the information available to the decoder in order to decide which codebook to use. For instance, reconstructed vectors from nearby locations, the codebook that was used the last time (corresponding to the previous state of the system) and the codevector that was chosen from that last

codebook, are all pieces of information available to both the decoder and the encoder that can be used to decide which codebook to use for the current vector (corresponding to the current state of the system). The decoder and encoder both know the next-state function. Assuming that the decoder and encoder start with the same codebook and that there is no channel noise, the decoder will always be able to track which codebook the encoder is using next since the next-state function employs only information available to both parties.

3.6.11. Adaptive VQ

To improve the VQ image quality, we can use an adaptive codebook which changes its codevectors depending on the local statistics in the different image regions [56] [57]. The overhead data used to update the codebook should not be allowed to be much.

3.6.12. Predictive VQ

Predictive VQ is a straightforward vector extension of traditional scalar predictive quantization (like DPCM). The encoder makes a prediction of the incoming vector based on previously encoded vectors. The difference between the actual input vector and its prediction is called the residual; this residual is vector quantized. Because the encoder only uses the previous outputs in making its prediction, the decoder is able to make the same prediction. After dequantizing the residual vector, the decoder adds the prediction to it to form the reproduction vector. The predictor is often a simple linear predictor that takes a weighted average of nearby previously encoded vectors.

CHAPTER IV

Vector Quantization Index Compression

VQ has been found to be an efficient compression technique for images due to its simple decoder which is just a table lookup. It can be roughly classified into memoryless VQ and memory VQ [1]-[3]. In a memoryless VQ, each image block is quantized independently and its corresponding index is sent to the decoder. The block size used in memoryless VQ is usually 4 x 4 pixels which is relatively small, thus there is usually a high correlation between the neighboring blocks and hence the resulting neighboring indices.

A number of memory VQ algorithms that exploit the inter block correlation has been proposed to achieve a lower bit rate. However, most of memory VQ algorithms are complicated in implementation because they exploit the correlation information using the original high dimensional vectors.

Lossless index compression algorithms are introduced to solve this problem [4]-[11]. Index compression algorithms usually are much simpler than conventional memory VQ algorithms. The inter block correlation is exploited in the index domain and the performance is improved in terms of bit rate reduction while still using the basic codebook of the memoryless VQ.

The VQ system with index compression is shown in Fig (4.1). A lossless index compression stage is inserted after the basic VQ. After an image is vector quantized on a block by block basis, 2D index map is then generated Fig (4.2). Each point in the index map corresponds to a 4 x 4 block of pixels. While VQ is lossy stage, index compression is lossless and doesn't introduce extra losses in image quality.

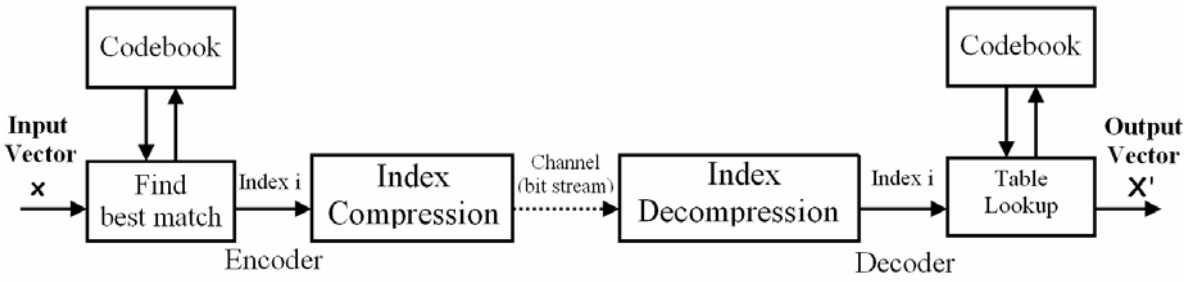


Fig (4.1): VQ index compression block diagram

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	24	52	47	29	24	28	28	30	28	28	28	30	36	32	32	30
2	28	57	37	24	24	24	28	28	28	28	30	36	30	32	30	36
3	30	57	37	24	24	28	24	23	28	30	32	30	30	30	36	36
4	33	57	37	24	24	24	28	28	30	30	28	28	28	30	30	36
5	41	55	36	28	24	24	24	24	28	30	28	30	36	32	30	28
6	41	51	36	29	28	28	24	24	28	28	30	30	30	30	28	39
7	56	51	38	30	24	28	24	28	28	28	30	30	36	28	36	42
8	58	51	42	36	30	28	24	28	28	28	32	30	30	30	42	38
9	58	51	44	38	36	28	28	28	28	28	29	28	36	38	38	38
10	57	51	45	42	38	30	28	24	24	28	28	30	38	38	38	37
11	57	51	45	42	38	36	28	24	28	24	33	38	36	38	32	30
12	54	53	45	44	39	36	30	28	24	30	38	38	36	32	30	38
13	57	53	47	45	42	42	36	24	30	38	36	36	32	30	38	38
14	57	54	47	45	44	44	37	28	32	36	36	32	30	36	38	38
15	57	54	53	47	38	42	37	28	30	30	32	28	36	38	36	32
16	58	54	54	47	36	36	32	28	28	29	28	36	38	36	32	36

Fig (4.2): VQ index map of a 64x64 image

At the receiver, the full indices are recovered again using index decompression before applying them to the VQ decoder where the image blocks are reconstructed again. In this chapter, many index compression algorithms are introduced.

4.1. Huffman Coding of Indices

If variable length codes are used to represent indices, storage requirements can be further reduced by optimizing the code lengths according to the indices probabilities. One of the most widely used variable length coding schemes is Huffman coding which uses the indices occurrence probabilities in the image to assign a code to each index. Shorter codes are used for the higher probability occurring indices, and vice versa. This can be done either for the original indices, or for the adjacent indices differences. Note that each code is uniquely decodable and no sequence of codes can be mistaken with any other codes, which is a characteristic of this type of coding, for example see Fig (4.3). The index distribution probability is estimated for each individual image or estimated for all images. The codes based on the individual image statistics are called local codes, while the codes based on a set of images statistics are called global codes. Global codes are generated once and found at both the transmitter and the receiver to be used with all images. Local codes need to be generated for each image and sent to the decoder; however, this will introduce extra complexity to the algorithm as well as extra overhead to the bit rate.

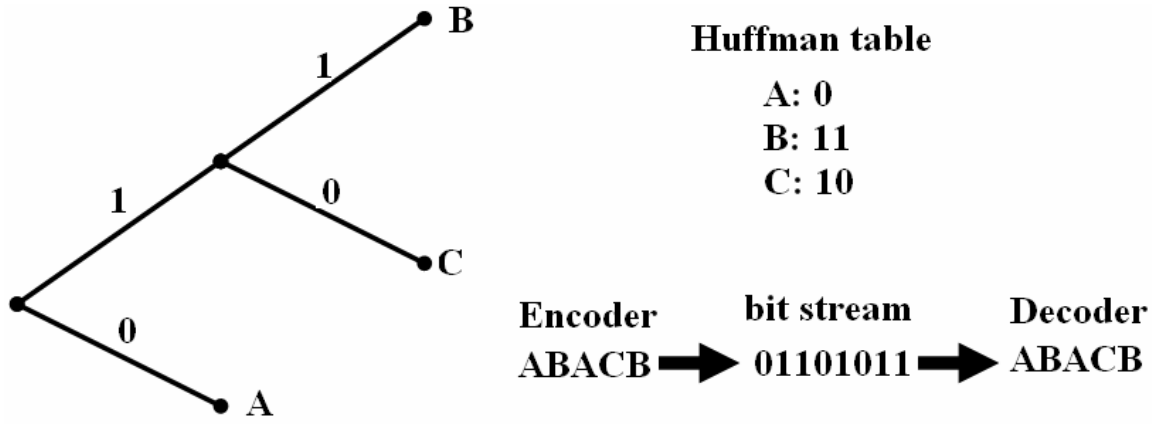


Fig (4.3): Huffman codes

DPCM can be also applied to the indices and then applying Huffman coding to the resulting differences. This will increase the efficiency of the Lossless coding, since the correlation between adjacent indices is removed by difference operation. The bit rate obtained using DPCM and Huffman coding is the lower bound that can be reached by variable length codes. However, Huffman coder does not give the bit rate that can be reached with other VQ oriented methods [4]-[11].

4.2. Address VQ

Address VQ is the earliest work proposed for VQ index compression [4]. It is simply a lossless VQ of the indices that utilize the advantage of the behavior of groups of adjacent blocks. The address VQ use an address codebook constructed with the indices of four adjacent blocks, see Fig (4.4). If the adjacent pixels forming these blocks are highly correlated, VQ indices for these adjacent blocks will tend to occur in patterns. Consider groups consisting of four adjacent indices forming a square, the more correlated the adjacent blocks, the fewer are the regularly appearing patterns of indices. Two codebooks, the original VQ codebook and the codebook of combinations of commonly occurring groups of indices (called address codebook), are combined into a single extended codebook. The address codebook contains the N' most likely possible groups of four indices that result when the original VQ is applied to the image. The indices codebook can in principle be found by encoding the image with the original VQ and computing the histograms of all 4 possible combinations of four VQ indices. The N' most probable combinations are then listed in the address codebook.

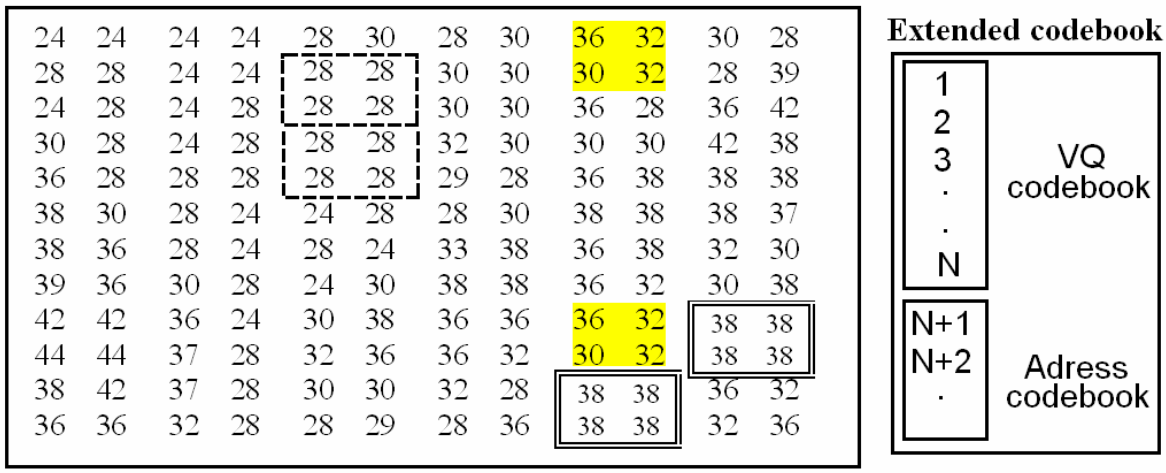


Fig (4.4): Address VQ

The extended codebook is used to encode the input as follows: the input image is divided into 8 x 8 blocks instead of only 4 x 4 blocks as would be used by the original VQ alone. For a given 8 x 8 big block the encoder first encodes the 8 x 8 block by using the original VQ on the four 4 x 4 blocks and produces a group of four small VQ codevectors. If this particular pattern is in the index codebook, then its index in the extended codebook is sent using $\log_2(N')$ bits. If this occurs, then the $\log_2(N')$ bits specify a complete big block to the decoder. If the original VQ encodes the input 8 x 8 block into a group of four VQ indices that is not in the address codebook, then we essentially return to separate coding of each 4 x 4 block with the original VQ. Hopefully the average number of bits per pixel will be reduced from the original VQ. Address VQ improves compression efficiency significantly at the cost of a great amount of computations [6].

4.3. VQ Index Compression with SOC

Fig (4.2) shows a typical 2D index map of size 64x64, since image blocks are highly correlated; many blocks in an image corresponding to the same index

are found in nearby area (e.g. the index 28 in the middle of the index map). In other words, many points in the index map have the same index value. The main idea behind this scheme is that it searches the previous indices in a predefined search path for a matched index to the current index and sends its corresponding search order if any previous index is matched, the repeated index values are neglected to improve the performance. Because the quantization index is a scalar, not a high dimensional vector, the algorithm is simple. Furthermore, not only the correlation of adjacent blocks but also that of blocks located apart are exploited.

First, the blocks are scanned in a raster scan order, i.e., from left to right and top to bottom. An attempt is made to find a matched point of the search center indicated by the black circle in Fig (4.5). Matched point signifies a point whose index value is equal to that of the search center. In general, all points in the index map except the search center are candidate points for matching. However, this will result in a coding delay, since VQ indices are obtained in a raster scan order. Therefore, the semi causal neighbors of the search center are chosen as search points (SP's). To begin a search, one should select a starting search point (SSP) from the SP's. The SSP is determined by the four directions defined in Fig (4.6). After a search center and an SSP have been determined, one searches the four SP's of the first level in a clockwise way, beginning from the selected SSP. If no SP's are matched with the search center, one searches the SP's of the second level in a similar way. If a matched point is found, the search order code of the matched point is sent to the decoder. Otherwise, the original index of the search center is transmitted. The search order is defined as the order m which a SP is compared with the search center. To let the decoder distinguish between the search point from the full index, an extra bit is needed. They may exist a repetition points in the search path, where the repetition point is the point with index equal to the previous index.

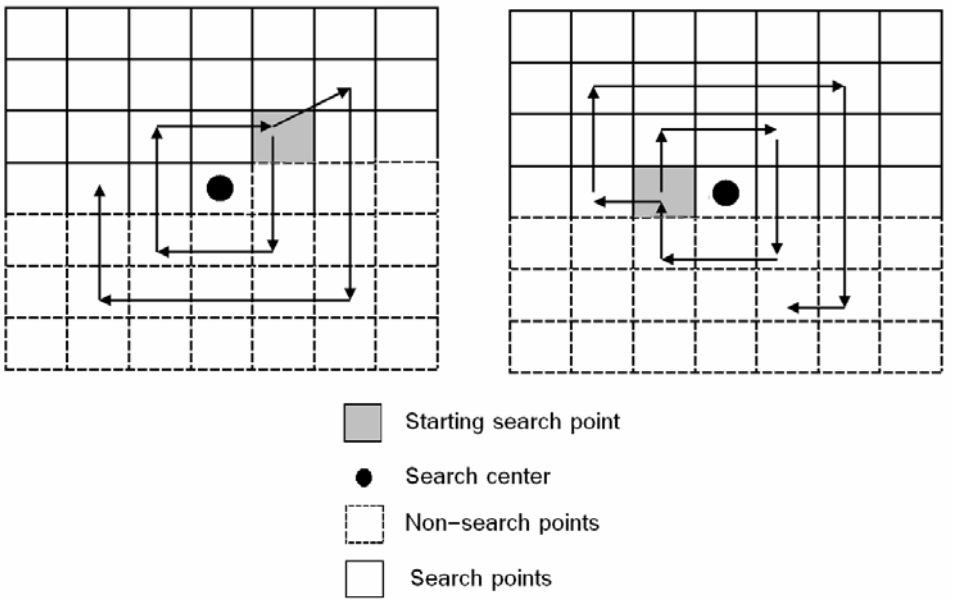


Fig (4.5): Different search paths

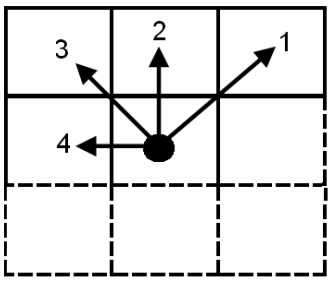


Fig (4.6): Different starting search points (SSP)

Before matching the (SP) is checked to determine if it is a repetition point, if so, the matching operation for this point is neglected. The exclusion of repetition points will increase the possibility for finding matched points because the number of bits *n* assigned to search order code is limited. For example, if a 2 bits search order code is used, we have only four SP's to be compared without excluding the repetition points; whereas, we have more than four SP's when the repetition points are excluded. Consequently, it will increase the coding efficiency. Fig (4.7) illustrates the search order coding that excludes the repetition points. In such a case, the code words are given only to the non repetition SP's. It is apparent that one cannot find the matched point if

the repetition points are not excluded.

The bit rate of this system is given by

$$BR = [Nr \times \log N + (Nb - Nr) \times n + Nb \times 1]/(K \times L) \qquad (4.1)$$

Where

Nr: total number of indices sent to the decoder;

N: codebook size (total number of indices);

Nb: total number of blocks of an image;

n: number of bits assigned to search-order code;

K x *L* = image size.

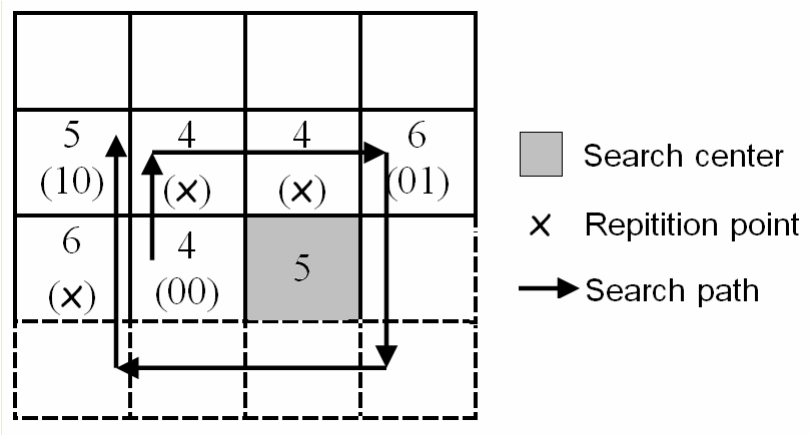


Fig (4.7): Excluding the repetition points the search-order coding

4.4. Low Complexity Index Compression

This method is introduced by Hu and Chang [5]. It depends on the index correlation between the adjacent indices, namely the upper and left indices. It achieves index compression while keeping low computation complexity. The codebook used is sorted by the mean values of the codevectors. By using the mean sorted codebook, the resultant index map tends to be more compact. In other words, the neighboring indices are quite similar to each other and the differences between any two of them are small.

Each index in the 2D index map is processed in the raster scan order to be compared with both the upper and left indices. First, the current index is checked to see whether its adjacent left entry and its adjacent upper entry in the index map have the same value as it, see Fig (4.8).

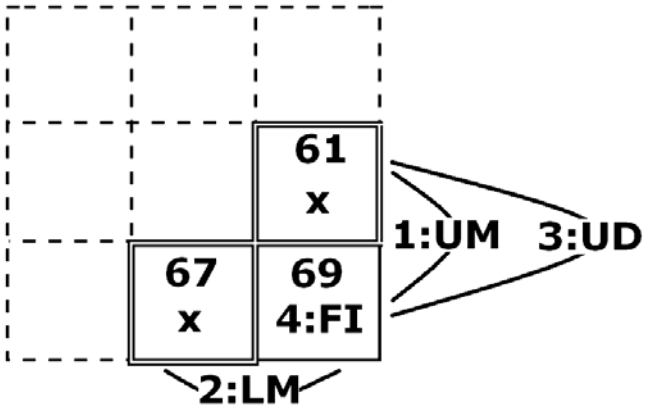


Fig (4.8): The adjacent (upper and left) indices in the index map

If the same index is found in either of the two positions, only one bit will be used to indicate which one of the two entries has the same value as the current index. If r bits are used to represent the index, that is, there are 2^r codevectors in the codebook. In transmission, 2 bits is required for the current index having the same index value or (r+l) bits in the case of not having the same index. Note that an extra indicator bit is needed to distinguish whether the same index value of the current index has been found or not.

For each index that does not have the same index value as its adjacent left and its upper indices, the offset between this index and its previously index is computed. If the offset is smaller than a predefined threshold, the relative offset is transmitted to the decoder; otherwise, the original index for this index is transmitted to the decoder. Note that another extra bit is needed to indicate the two different types. In the decoder, the image blocks are constructed in a raster scan order. The first two bits of the received index are used to judge to which group the current decoded index belongs.

4.5. Index Grouping Algorithm

Index grouping Algorithm (IGA) [8] is a noiseless coding algorithm which groups up the blocks with the same index and then encodes the group path by using an efficient search order coding scheme. The algorithm improves the coding efficiency without introducing any extra coding distortion, since the indices are grouped in an appropriate manner.

The algorithm consists of two processes a search process and an encoding process. The search process is used to locate the points with the same index value from the index map and then to connect these points so as to form a group path. The encoding process is used to encode the resulting group path in an effective and ordered manner.

Every point in an index map is called a search point (SP). The starting point of a group path is called a reference point (RP). The value of a RP is called a reference index (RI). Now the search process is to match up a SP with a given RP, where the matched SP's have the same index as the RP.

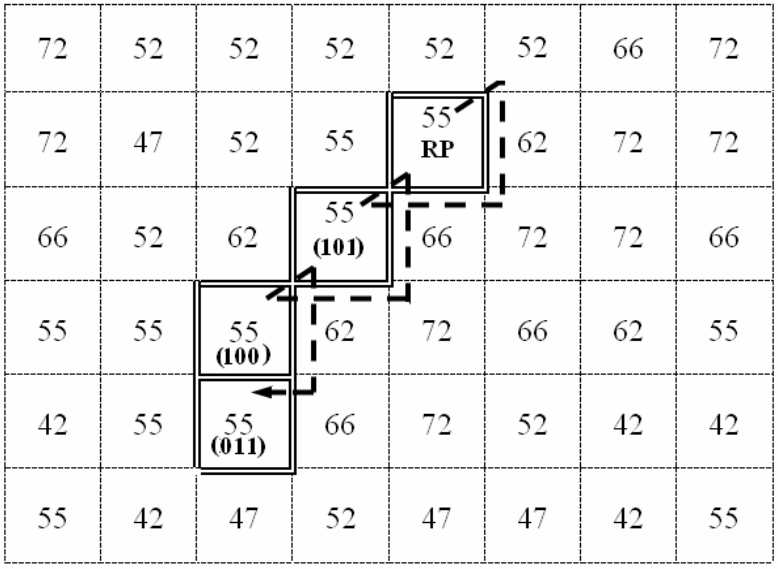


Fig (4.9): The IGA search process

The current RP is selected as a search center, and a starting search point (SSP) from the neighborhood of the search center. The SSP is determined by any index of the neighboring eight indices. From Fig (4.9) we begin with the

selected SSP to search clockwise the SP's in the first level. If no SP's are matched up with the center, we search the SP's in the second level in a similar way. If a matched point is found, we send its corresponding code to the decoder. The matched point is then used as a new search center, and the search procedure repeats in an analogous manner. This search procedure will divide all indices in the index map into several groups, each of which corresponds to a RP. The RP of a group is selected from the SP's of the index map in a raster scan order except the points which have been already grouped. The above search procedure may yield a redundant matching which is defined to be the matching of indices between a search center and a non candidate SP. Therefore, before a matching up is performed we should determine whether a SP is a candidate for a certain search center. If it is not a candidate, the matching operation or the SP and the search center is neglected. For the current search center with index i, a SP is regarded as a non candidate if:

1) It has been compared by other search centers and admitted as a matched point.

2) It has been compared by other search centers with index i and admitted as a mismatched point.

An N bit (N is the total number of indices) status word, records the status of its corresponding SP in the index map. Each bit of a status word designates the status corresponding to each index, respectively. For a SP, if all the N bits of its status word are logic "0", it means that this SP is a candidate for all indices. In other words, it is permitted to be compared by a RP with any index value. Initially, every status word of the status map must be reset to the logic zero because every point in the index map is a candidate SP before the grouping procedure starts. The use of the status map makes the coding of search orders very efficient.

After the search process for a RP has been completed, we obtain a group path which is formed by connecting the matched points. The matched points are found from the candidate search points. To increase the coding efficiency of a group path, the search order coding scheme of SOC method is used here. This scheme encodes the search order of a matched point, which is defined as the order that the candidate SP's are compared. It should be noted that the non candidate SPs are excluded in the generation of search order code (SOC). In a transmission, for each group we send a RP followed by the associated SOCs to the decoder. To let the receiver distinguish RP from SOC, an extra flag bit is added in each RP and SOC before the transmission. In the decoder, we also have an identical status map.

However, IGA is complex for hardware implementation, since the operation of group search is not very regular. Also, it requires extra huge memory space in either encoding or decoding operations.

4.6. Switching Tree Coding

In STC algorithm [7], four relations between adjacent indices are defined in a 2D index map.

1. Upper-connection (U-con): the current input index is the same as the upper index.
2. Left-connection (L-con): the current input index and the left index are identical.
3. Around-connection (A-con): the current index is identical to the around index except the upper and left indices, the search path is defined in Fig (4.10).
4. Disconnection (D-con): the current index is fully different to all indices in the neighbor area scanned by the search process.

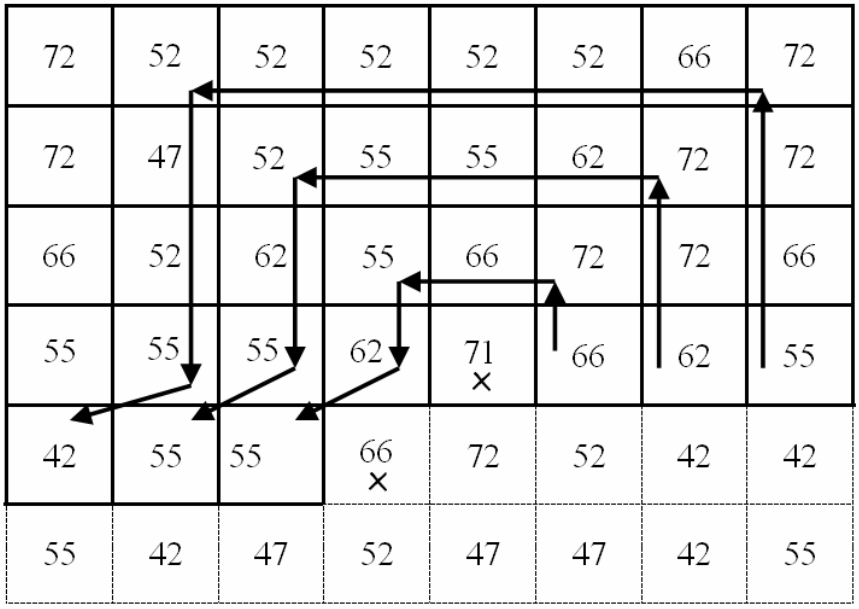


Fig (4.10): Search order for the STC method

Based on the above connections, three connection prefix trees are introduced, see Fig (4.11). These trees have different coding results according to 4 types of connections.

In around connection, the current input index has to compare the previous indices in nearby area. These previous indices can be stored in the memory buffer with M locations according to the arrow order. In order to extend the search area, all content in memory are unlike. Consequently, only one of the same indices can be saved in the shift-register. When the current index finds the match index in the memory buffer, the current index would be encoded in the corresponding address code and plus prefix code in connection tree.

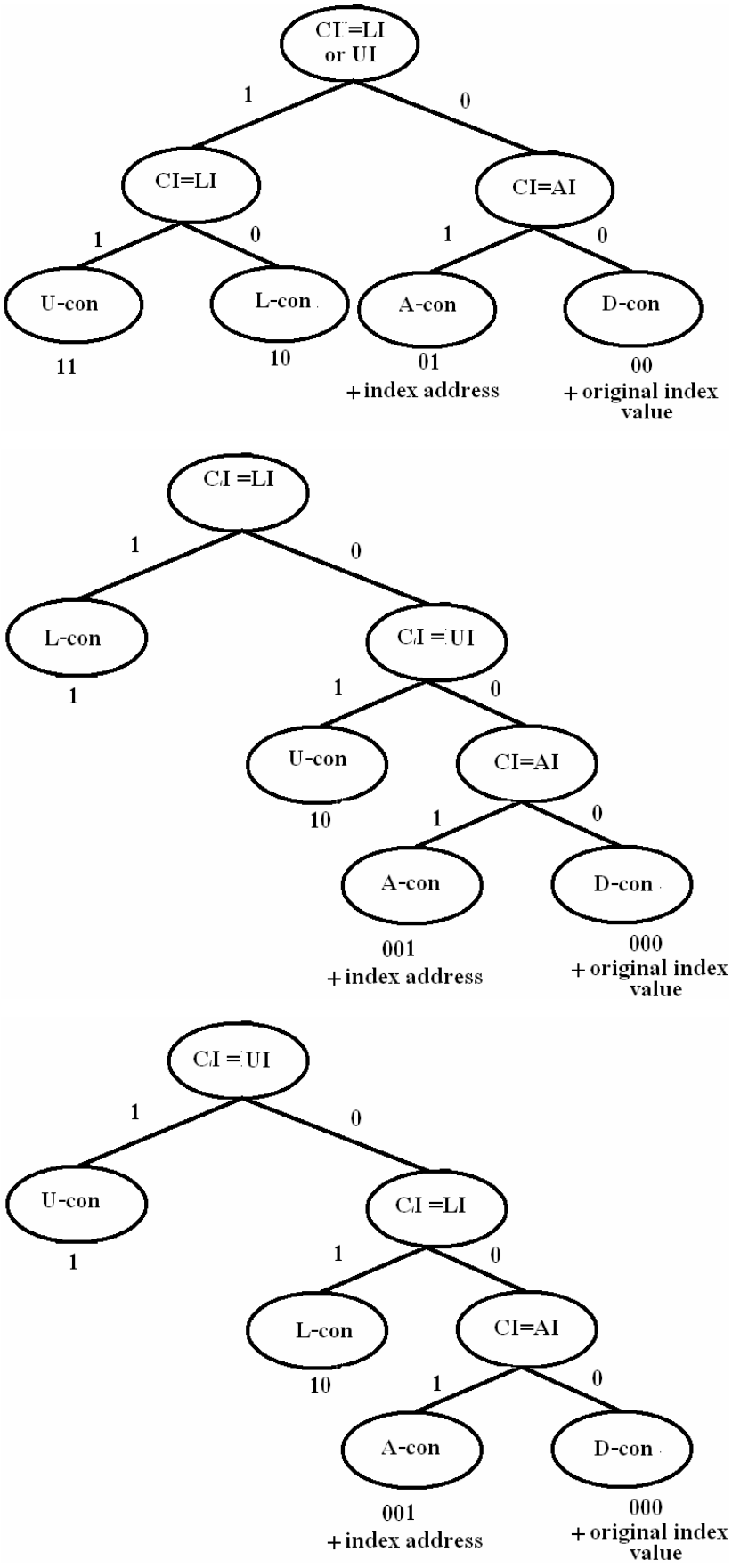


Fig (4.11): different switching trees

CHAPTER V

New Index Compression Algorithms

In memoryless VQ, each block is encoded independently and the index of its corresponding closest codevector in the codebook is transmitted to the decoder. The decoder can then reconstruct the encoded image according to the received indices. In general, the image quality of the memoryless VQ depends on the codebook size used in the encoding and decoding processes. Higher reconstructed image quality can be obtained when a larger codebook size is used in a VQ scheme. However, a higher bit rate is needed then. The adjacent indices in the 2D index map are found to loose correlation when increasing the codebook size. Index oriented compression algorithms are presented in the previous chapter. These methods give a recognizable reduction in the bit rate, but there are still improvements that can be done especially for larger codebook sizes.

In this chapter, three contributions for index compression are introduced. In the first contribution, a modification on the Hu and Chang method is introduced. In the second method a new proposed algorithm called the repeated adjacencies method is presented. Finally, a remapping process is introduced. This process is applied to the VQ indices in different regions of the image exploiting the fact that not all indices are used in each region, so that fewer bits are needed to represent the indices of each region.

5.1. Enhanced Hu and Chang Method

The Hu and Chang method explores the correlation between adjacent indices, namely the upper and left indices, in the index domain. The goal of this

algorithm is to improve the performance of the basic VQ scheme at low bit rate by index compression, while keeping the same image quality with a low computations complexity. In the Hu and Chang method, the index follows one of four cases, the first one when the current index matches the upper index, the second when matches the left index, the third when the difference between the current index and the upper (or left) index is less than a predefined threshold, the forth case when the index does not match any of the previous conditions. Only two bits are used to distinguish between the four cases.

The main disadvantage of this method is that it does not check the differences between the current index and both the upper and left index. This will result in increasing the percentage of indices that belong to the forth case of sending the full index.

The main idea behind the proposed modification is to completely explore the correlation in the index domain between the current index and both the upper and left indices. This is done by including a fifth case applied when the difference between the current index and the left (or upper) index is less than the predefined threshold. Thus, the indices in the 2D index map can belong to fives different cases. Each case is encoded using its own code. Using fixed length codes requires extra bits and thus could reduce the compression ratio dramatically. Variable length codes are the solution to this situation where Huffman coder is used to generate these codes based on the probability of occurrence of the five cases. The average bit rate will be reduced much and improvement in compression ratio is achieved. There are two options; the first is to generate global Huffman codes that will be used with all images, or local Huffman codes which are generated for each image. Using global Huffman codes sustains the simplicity of the original Hu and Chang method; this will be suitable for hardware implementation where the complexity is an important issue. If the complexity is not such important such as bit rate or the algorithm

is implemented using software, local Huffman codes will be more preferable. Local Huffman codes are more optimized to individual images statistics. However, generating Huffman table with five codes is still a simple process.

In this algorithm, each index in the index map is processed in the raster scan order. First, the current processed index is checked to see whether its adjacent left or upper indices in the index map has the same value as the current. If the same index is found in either of the two positions, then only a prefix code will be used to indicate which one of the two entries has the same index value as the current processed index.

If both indices does not match with the current index, the current index is then checked to see if the offset to either the upper or the left indices is smaller than a predefined threshold, the relative offset is transmitted to the decoder, otherwise, the original index is transmitted to the decoder.

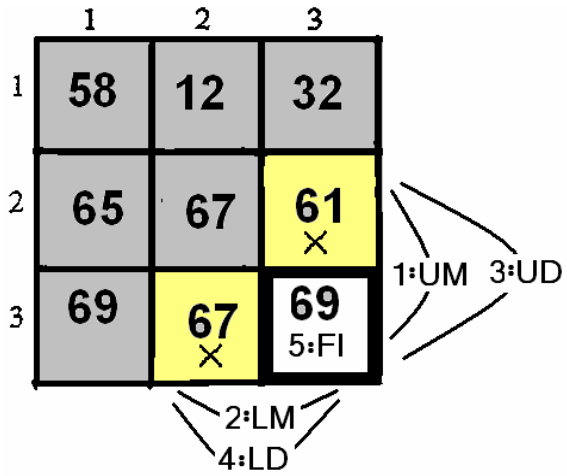


Fig (5.1): The adjacent indices (upper and Left)

The Algorithm:

Step 1: Arrange the codebook vectors according to their mean values.

Step 2: Apply the full search algorithm or any equivalent to find the best match codevector in the codebook for the input vector in terms of minimum MSE between them.

Step 3: The VQ indices are then scanned in order from left to right and from top to bottom, the current index is checked to determine if it has the same value as the upper index, if it does match, the code A is sent, and go to step 7.
Step 4: The current index is checked to determine if it has the same value as the left index, if it does match, the code B is sent, and go to step 7.
Step 5: The differences between the current index and each of two adjacent indices (upper and left) are checked to determine if the smaller difference is a below a predefined threshold, the corresponding code will be [prefix C + index difference] or [prefix D + index difference], respectively. The index difference is sent as a signed number (needs an extra bit), and go to step 7.
Step 6: When the index does not match any of the previous conditions, then the code will be [prefix E + full index].
Step 7: If it is not the last index in the index map, go to step 3 with the next scanned index, else end.

A, B, C, D, and E are binary prefixes used to indicate the different index cases. They are generated locally or globally using Huffman coder depending on the percentage of different cases in the image. This algorithm is very simple and suitable for hardware implementation in either of its two versions of the local and global prefixes.

5.2. Utilizing Repeated Adjacencies for VQ Index Compression

VQ is mainly a mapping process that projects the image blocks into a reduced set of codevectors. For natural images, there is some kind of pattern repeating where similar image components tend to repeat in a certain arrangement. This is clear especially at textures, gray backgrounds, and edges. In VQ the image is divided into small blocks which inherit the natural images pattern repetition property. When mapping the image blocks into the codevectors, the resulting

indices are found to confirm the repeated patterns. This could be employed for a new index compression algorithm where the local statistics of each image and the repeating pattern of its adjacent indices are utilized.

In the proposed algorithm the 2D index map is scanned from up to down and from left to right. A search is done for the repeated patterns of spatially adjacent indices pairs. The repeated patterns between VQ indices can be exploited by generating for each individual image a "Next" (or "Right") table. It is an 1xN array, where N is the codebook size, that keeps for each index a pointer for the most probable index that comes next (and/or right) to it. The "Next" table for every individual image is constructed by finding for every index the spatially lower index with highest repetition rate, see Fig (5.2). For example, in Fig (5.2) the index value 32 is followed by the index values 36,44,32,30, and 26 with the index value 36 has the highest repetitions.

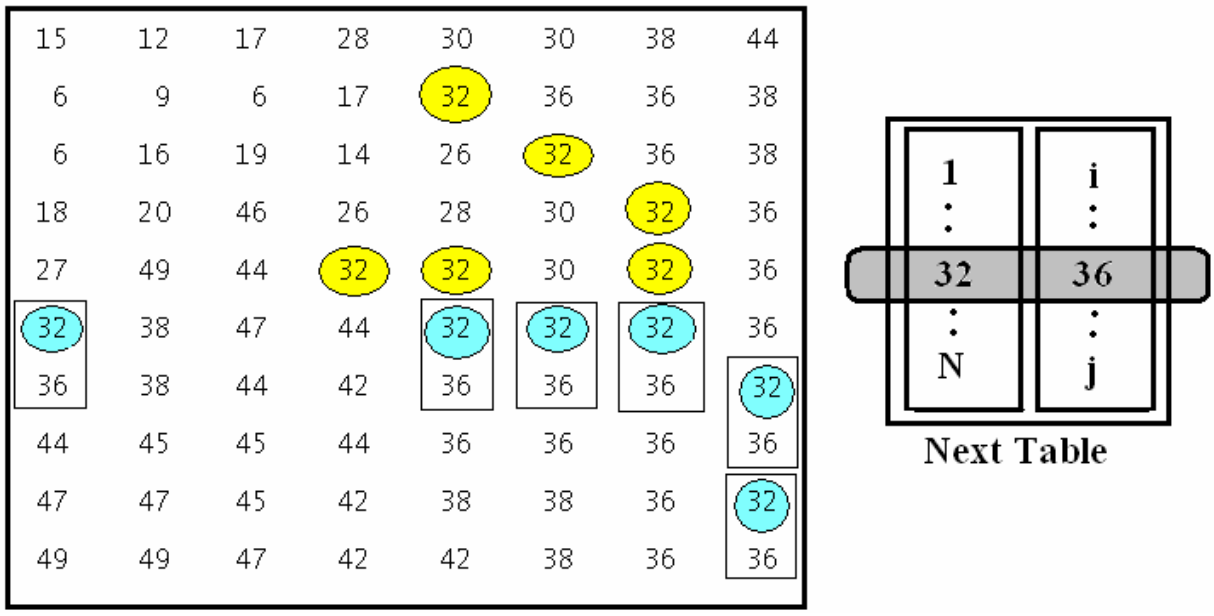


Fig (5.2): "Next" table generation process

Also the "Right" table is constructed the same way, but it keeps for each index the most probable index which comes just right to it. Since this table is constructed on individual image basis, it needs to be sent to the receiver each time we encode an image and it represents an extra overhead.

The current index is to be checked to determine if it has the same value as the index in the "Right" (and/or "Next") table of the adjacent left (and/or upper) index; in this case a short binary code is sent. If the current index does not match the previous condition, then the difference in index value between the current index and the adjacent indices (upper or left) is checked, if it is less than a predefined threshold then this difference is sent, else the full index must be sent.

The used codevectors should be arranged according to their mean values to reduce the average difference in value between adjacent indices.

The Hu and Chang method can be considered as the special case of this method where the "Next" or "Right" tables contain the same index value as the index itself.

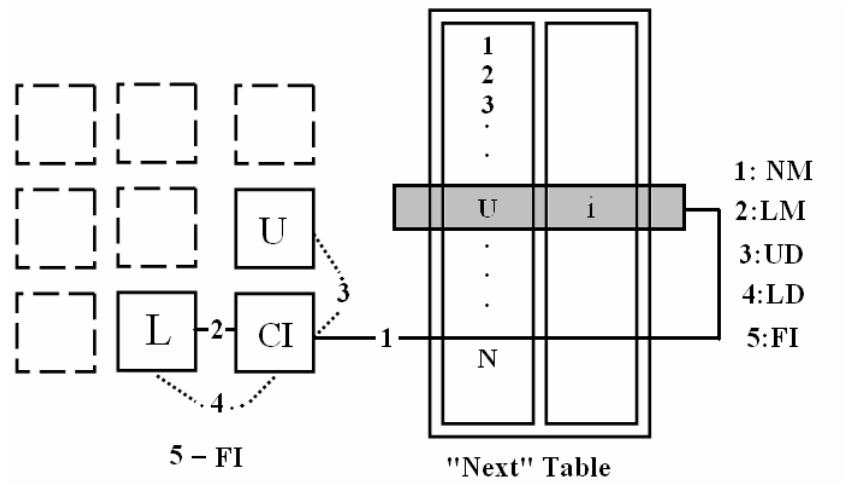
Three methods for the proposed algorithm are studied:

Method 1: Compares the current index value with the value of the "most probable next index" of the upper adjacent index and with the value of the left adjacent index, Fig (5.3 a).

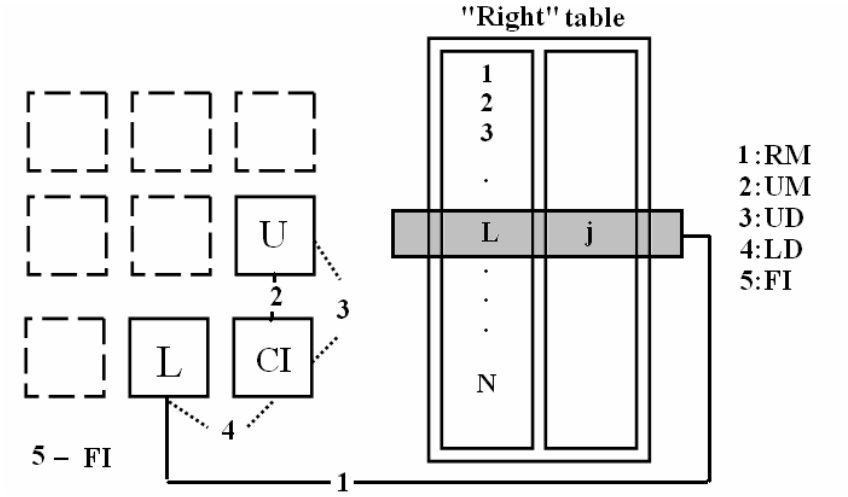
1- The indices in the 2D index map are scanned from top to bottom and from left to right. The "Next" table is constructed as described previously.

2- The scanned current index **CI** is checked to determine if it has the same value **i** as the most probable next index of its upper index **U**, if it does match, the code A is sent, and go to step 6.

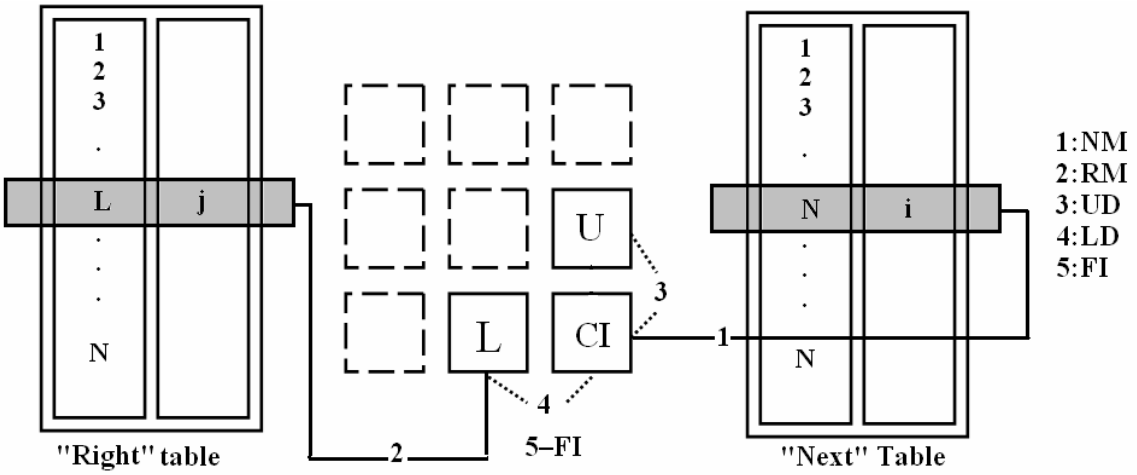
3- **CI** is checked to determine if it has the same value **L** as the adjacent left index, if it does match, the code B is sent, and go to step 6.



a)



b)



c)

Fig (5.3): Repeated Adjacencies method a) Using "Next" table, b) Using "Right" table, c) Using "Next" and "Right" tables

4- The differences between **CI** and each of **L** and **U** are checked to see if they are below a certain threshold or not. If any of them doesn't exceed this threshold, the corresponding code will be [prefix C + index difference (**CI-L**)] or [prefix D + index difference (**CI-U**)], respectively. Whereby the index difference is sent as a signed number (needs an extra bit), and go to step 6.
5- When the index does not match any of the previous conditions, then the code [symbol E + full index] is sent.
6- If CI is not the last index in the index map, go to step 2 with the next scanned index, else end.

Method 2: Compare the current index value with the value of the "most probable right index" of the left index and with the value of the upper adjacent index, Fig (5.3 b).

The method is similar to method 1, but differing only in the indices to be compared with. "Right" table is constructed instead of "Next" table. Steps 2 and 3 are modified as follows:
2- The current index **CI** is checked to determine if it has the same value **j** as the most probable right index of its left index **L**, if it does, code A is sent, and go to step 6.
3- **CI** is checked to determine if it has the same value **U** as the adjacent upper index, if it does match, code B is sent, and go to step 6.

Method 3: Compare the current index value with the value of the "most probable next index" of the upper adjacent index and with the value of the "most probable right index" of the left adjacent index, Fig (5.3 c).

The method is similar to method 1, but differs only in the indices to be compared with. In this method both "Next" and "Right" tables are used. The steps 2 and 3 are modified as follows:

2- The current index **CI** is checked to determine if it has the same value **i** as the most probable next index of the upper index **U**, if it does, code A is sent, and go to step 6.

3- **CI** is checked to determine if it has the same value **j** as the most probable right index of the left index **L**, if it does match, code B is sent, and go to step 6.

Codes A, B, C, D, and E are determined by local or global Huffman coder.

The total bit rate needed to encode the image is calculated as follows:

$$\begin{aligned} BR = [&A\_size \times A\_HuffLen + B\_size \times B\_HuffLen \\ &+ C\_size \times (C\_HuffLen + \log_2(Thr) + 1) \\ &+ D\_size \times (D\_HuffLen + \log_2(Thr) + 1) \\ &+ E\_size \times (E\_HuffLen + \log_2(N)) \\ &+ TableCost + HuffmanTableCost] / (512 \times 512) \end{aligned} \quad \text{(bpp)} \qquad (5.1)$$

Where:

A_size, B_size, C_size, D_size, and E_size: Are the number of indices which are coded using A, B, C, D, or E respectively.

Thr: predefined index difference threshold. This number defines the bits needed to code the index difference.

A_HuffLen, B_HuffLen, C_HuffLen, D_HuffLen, and E_HuffLen: Are the lengths of the codes A, B, C, D, or E, respectively.

Table Cost: the bits needed to store "Next" table, "Right" table, or both depending on the applied method. Each table requires

$$TableCost = \log_2(N) \times N \quad \text{(bits)} \qquad (5.2)$$

For the third method we send two tables so the table cost will be doubled.

Huffman Table Cost: the bit size needed to store the binary codes A, B, C, D, and E for the local Huffman case.

5.3. Index Remapping Using Index Usage Maps

VQ maps the image blocks into codevectors represented by a set of indices. The natural image contents are varying between different textures and edges which cover relatively large areas of the image. This results in indices that are locally present in some areas of the image and not found in other areas, which is true especially for the large codebook sizes. This property could be used to reduce the range of indices, and hence the bit rate. A new VQ index remapping process is introduced to increase the compression ratio of different index compression algorithms. It utilizes the indices usage distribution to reduce the number of required bits. The total number of used indices in a certain area of an image is usually limited. A remapping process is employed to renumber the used indices and neglect the unused indices. Different index compression algorithms are applied to the remapped indices and the effect of the remapping process is studied. It is found that the remapping process increases the index compression ratio of other algorithms and makes it less sensitive to the increase in the codebook size.

In this algorithm, VQ is applied to the image using codebook of size N. The resulting indices are used to construct the 2D *index map*, which is divided into non overlapping square *index blocks* with size LxL. An *index usage map* is constructed for each index block, it is 1xN binary array that indicates which indices in that index block are used (a '0' refers to an unused index, while '1' refers to a used index), see Fig (5.4). Usually only a fraction of the overall number of indices is used in each index block. With the aid of the usage map the used indices in the index block are remapped (renumbered) into a reduced set of indices which are fewer in number than the original indices. This remapping reduces the dynamic range of indices, and hence reduces the bits needed to represent these indices. Since the index map is not the same for

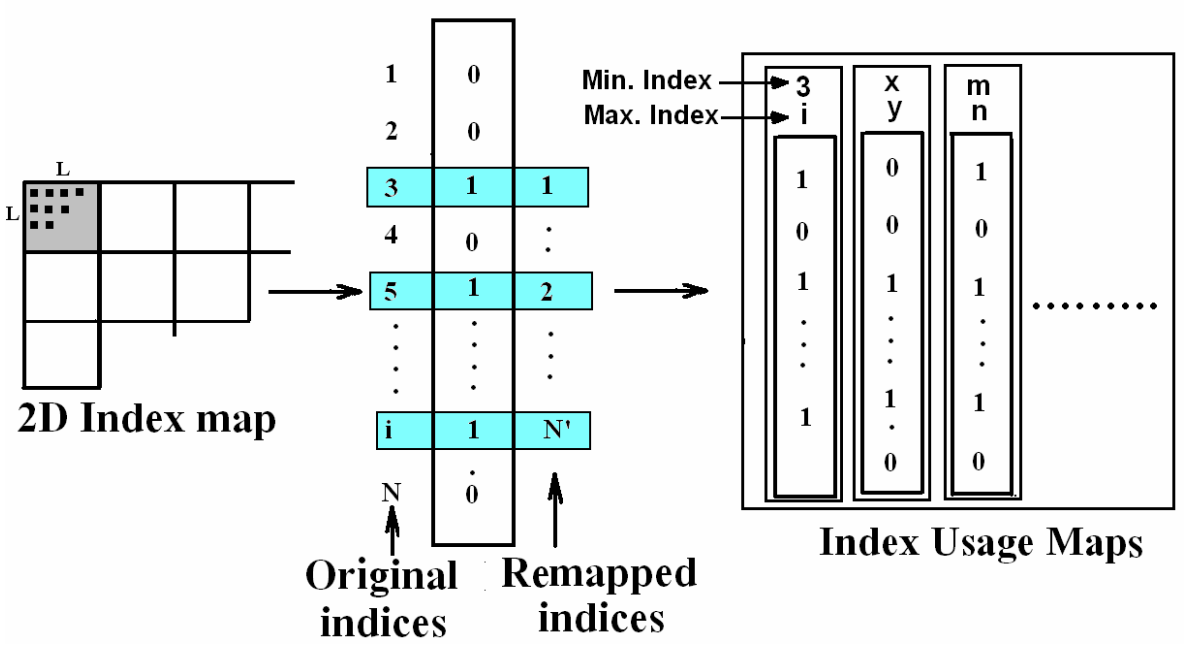


Fig (5.4): The remapping process.

different index blocks, the index usage map should be transmitted to the receiver as an overhead.

The index usage maps can be stored as binary array of zeros and ones, but this will increase the bit rate overhead. Alternatively, for each index block, the value of the minimum and maximum indices and only the usage map between them are sent, this will reduce the bit rate especially when the codevectors are arranged according to their mean values, see Fig (5.4).

Fig (5.5) shows "Lena" image that are vector quantized with codebook with size N=256 and the result is 2D index map that are divided into 32x32 index blocks, then the remapping process is applied to them. Thereby 4x4 pixels are represented by their remapped index, where remapped index of value '0' is appears in the figure as a black dot and with the increase of the index value we move along the gray scale to reach white dot by remapped index value of 255 (only possible if all indices are used in a certain block). Darker index blocks indicate that fewer indices resulted after remapping process, while the bright

index blocks indicate that more indices are remapped. High details index blocks use more indices than low details indices, and brighter index blocks are found.

Fig (5.5): Index distribution in different index blocks (N=64)

Fig (5.6) shows the average minimum, mean, and maximum percentages of the number of the used indices to the codebook size. This percentage reduces with increasing the codebook size.

5.4. Using Index Compression Algorithms with the Remapped Indices

Generally, Index compression algorithms cannot be cascaded, since the resultant output is stream of bits which does not contain any type of correlation. Index remapping generates a new set of indices, not a stream of bits, thus any index compression algorithm can be cascaded after the remapping process.

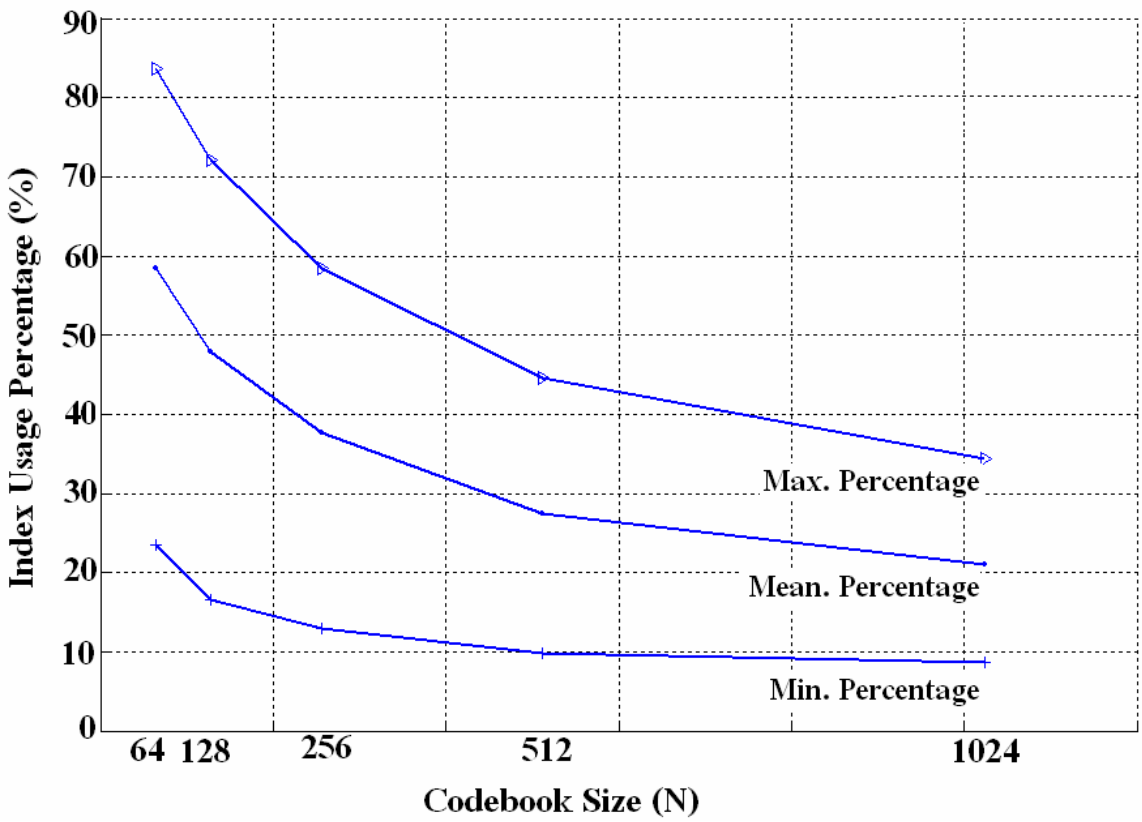


Fig (5.6): The average minimum, mean, and maximum percentages of the number of used indices to the codebook size for different codebook sizes

Many index compression algorithms can be used with the remapped indices to increase index compression efficiency. Fig (5.7) demonstrates the full VQ system with remapping and index compression. The remapping process is inserted between the VQ stage and the index compression stage. The correlation between adjacent indices is still found at the same index block, since the proposed remapping process affects only the absolute value of indices and the average difference between adjacent indices are reduced. At the decoder the remapped indices are used as an offset in the index usage map to retrieve the original indices.

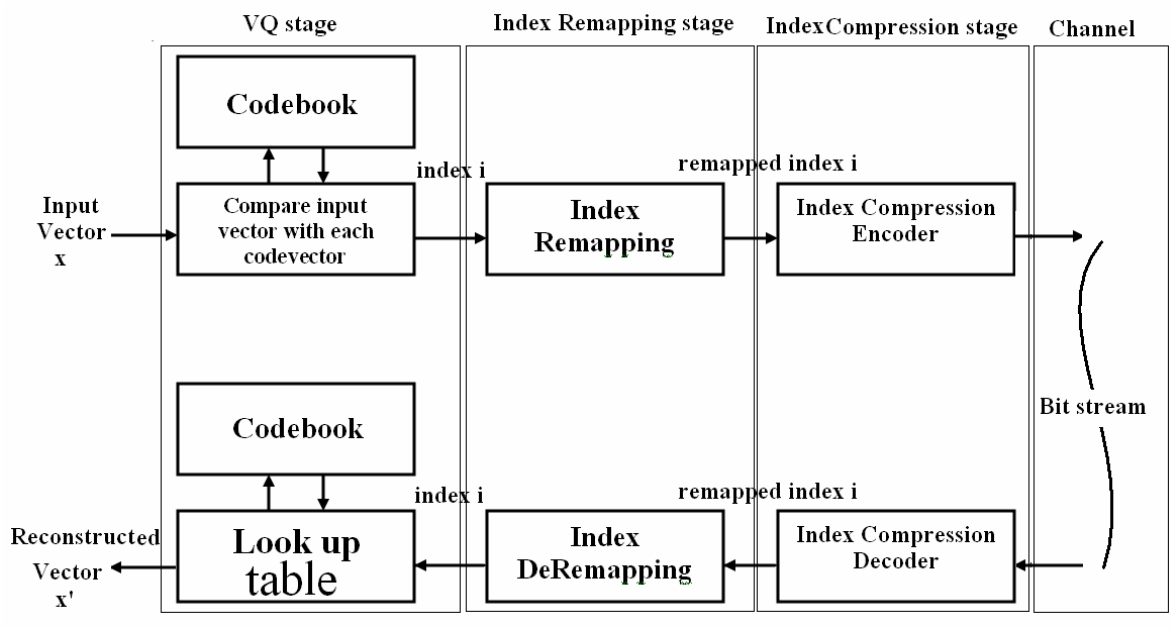


Fig (5.7): Full VQ system with remapping and index compression

CHAPTER VI

Simulation Results of the Proposed Algorithms

In this chapter, the proposed index compression algorithms are compared with other index compression algorithms and evaluated to explore its advantages and disadvantages. The algorithms are implemented using Matlab software package. The memoryless VQ scheme is utilized to generate the VQ indices used in simulations; it is the lossy part of the compression system. In VQ, the image to be compressed is divided into blocks which are converted into vectors with dimensions 1xk, where each having mxn pixels. In our VQ implementation we used the most common block dimensions (m=4, n=4, then k=16).

Five gray images; “Peppers”, “bridge”, “boat”, “harbor”, and “airport” with size 512 x 512 pixels and resolution of 256 gray levels are used to generate the codebooks used in the simulation. The splitting method is used to obtain the initial codebook, and the LBG is used to enhance it. Five codebooks based on 4x4 blocks and having sizes 64, 128, 256, 512 and 1024 codevectors are generated to explore the performance in wide range of bit rates.

The full search algorithm is used to find the best match codevector to each input vector from the previously designed codebook in terms of minimum Euclidean distance between the input vector and the codevectors. The index of the best match codevector is saved as a representative to the input vector. The same procedure is performed to all the blocks of the image and the result is a 2D index map. Each element in the index map corresponds to a pixels block in the image.

Ten standard test images “Lena”, “man”, “gold hill”, “Barbara”, “baboon”, “airplane”, “Elaine”, “peppers”, “bridge”, “Boats”, are used to test the algorithms. Three of these images are used in the training set, namely, “Peppers”, “bridge”, and “boats” images.

Fig (6.1) shows part of the original and reconstructed Lena images using VQ with different codebooks sizes. It is clear that using larger size codebooks increases the reconstructed image quality as represented by the PSNR.

a) Original b) N=64, PSNR=28.65 c) N=128, PSNR=29.42

d) N=256, PSNR=30.24 e) N=512, PSNR=30.92 f) N=1024, PSNR=31.53

Fig (6.1): Part of the original and reconstructed Lena image using VQ with codebooks sizes 64, 128, 256, 512, and 1024, respectively.

In the proposed algorithms the correlation between adjacent blocks is exploited, so that applying index compression algorithms to the output indices results in lowered bit rate without any loss in reconstructed image quality.

Huffman entropy coder [17], the Hu and Chang algorithm [5], and the search order coding (SOC) [6] algorithms are used here for comparison. For Hu and Chang method, the index offset threshold used is 16 which give better results than other values of threshold. In the SOC method, the search order is coded using two bits which give search order up to four different indices.

The compression ratio is represented here by the bit rate which is more suitable in telecommunication systems. The performance of index compression algorithms is measured by the percentage improvement in bit rate over the full index bit rate. The full index bit rate is the bit rate obtained when fixed length codes are used to represent the indices without compression, it is calculated from

$$BR = \log_2(N)/k \qquad \text{(bpp)} \qquad (6.1)$$

Where N is the codebook size and k is the vector dimension (=m×n pixels)

The improvement in bit rate for the index compression algorithms is calculated from

$$\text{The improvement (\%)} = 100 \times (\text{FIBR-BR}) / \text{FIBR} \qquad (6.2)$$

Where BR is the bit rate of the index compression algorithm and FIBR is the full index bit rate.

In the sections 6.1 to 6.3, the results of known algorithms are presented for the sake of comparison and the main factors that affect its performance are clarified. These are followed by the rates of the proposed algorithms and combinations of both in one table for the purpose of comparison.

6.1. Huffman Coding Simulation

The most basic index compression method is the Huffman coder. Since the indices are not equally probable, variable length codes can be assigned to them to reduce the total average bit rate. High probable indices are given short length binary codes while the low probable indices are given longer binary codes. Huffman codes can be generated globally based on the statistics estimated from a large set of images or can be generated locally for each image based on its own statistics. Global codes are found at both the transmitter and receiver, so it does not need to be sent with the image. However, it is not optimized for individual image statistics, so it does not give the optimal variable length codes. On the other hand, local Huffman codes give optimal binary codes for each image. Global codes are used in practical applications to reduce the complexity of generating local codes for each image. To increase coding efficiency, the correlation between adjacent indices can be removed by applying DPCM to them. The bit rate obtained using DPCM cascaded by Huffman coder indicates the minimum bit rate that can be obtained using variable length coding.

Arranging codevectors according to their mean values results in adjacent indices with near values. Using DPCM gives small index differences with high probability, while the large differences values are less probable. Fig (6.2) shows the used global codes bit lengths for codebook size N=64. It is clear that the index difference of value zeros can have as low as 5 bits code while the index difference of value 60 can have 15 bits code.

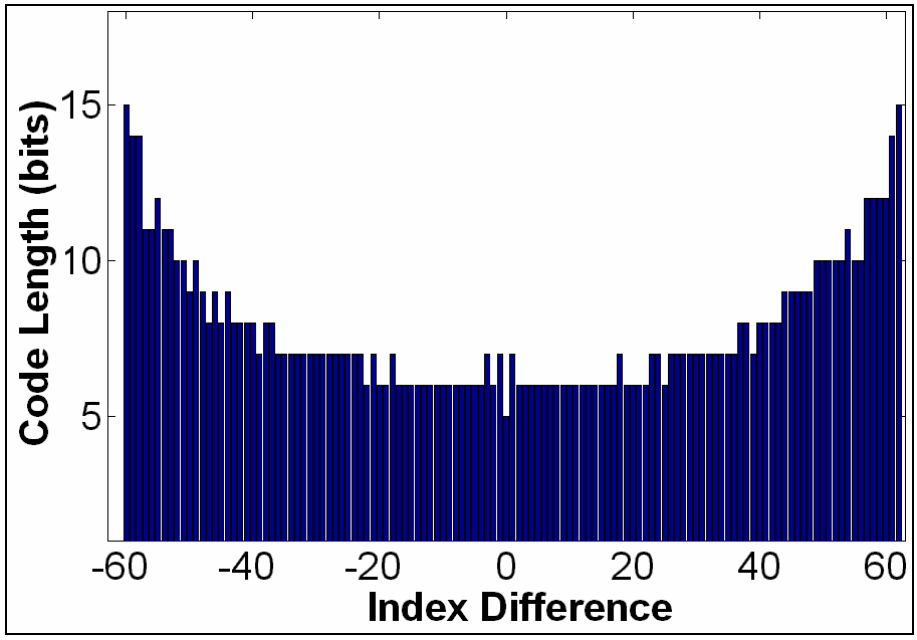


Fig (6.2): Length distribution of the global Huffman codes used for index differences (codebook size N=64)

6.2. Search Order Coding Simulation

For the SOC algorithm, the percentage of indices that are coded using search order gives an idea about the compression ratio where high percentage results in decreased bit rate. In our implementation 3 bits are used to code the search order code. Table (6.1) shows this percentage for different codebook sizes of the different reconstructed images. We note that for small codebook sizes this percentage is high, while with increasing the codebook size the percentage decreases. This is because the smaller codebook sizes result in highly correlated indices, so it is more probable to find a matched index in the nearby area. So that it can be deduced, that the SOC performance is deteriorates with increasing the codebook size. The high detailed images (like baboon image) have less percentage than the low detailed images (like airplane image).

Table (6.1): The percentage of indices that are encoded by search order

Image \ N	64	128	256	512	1024
Lena	75.3854	69.2929	59.5853	52.6915	40.3107
Man	58.9613	48.2567	37.4847	29.9303	20.7426
GH	67.3538	57.7074	46.5562	37.5948	26.3457
Barbara	67.2559	57.6156	47.9386	40.2740	30.7316
Baboon	43.4120	33.1233	22.8958	16.3812	10.0624
Airplane	74.7614	67.9900	59.6587	54.5265	44.1277
Elaine	74.7431	66.8216	54.0800	44.3846	29.9731
Peppers	74.7370	68.5038	58.5148	49.9266	35.9922
Bridge	46.8131	35.3805	24.6697	17.9716	11.4876
Boats	71.0729	63.4206	55.7010	48.6359	39.2770
Average	65.4496	56.8112	46.7085	39.2317	28.9051

6.3. Hu and Chang Method Simulation

For the Hu and Chang method, the percentage of indices, that doesn't need storing/sending the full index, indicates the reduction in the bit rate. With the increase of this percentage the compression ratio increases. Table (6.2) shows the percentage of the different cases for the Hu and Chang method, namely:

UM (upper matched): the current index is the same as the upper index.
LM (left matched): the current index is the same as the left index.
UD (upper difference): the difference between the current index and upper index if it is less than a pre defined threshold.
FI (full index): the current index does not match any of the previous cases and full index will be sent. Again the performance of this method is seen to deteriorate with the increase in the codebook size.

Table (6.2): Average cases' percentages for the Hu and Chang method

case \ N	64	128	256	512	1024
UM	38.5309	31.9012	24.6503	19.9109	13.0212
LM	17.0044	15.6537	13.4399	11.6821	9.1895
UD	40.9241	40.3833	36.0809	26.6711	20.5035
FI	3.5406	12.0618	25.8289	41.7358	57.2858

6.4. Enhanced Hu and Chang Method Simulation

In the proposed enhanced Hu and Chang method the index follows one of five cases. Where the additional fifth case is LD (left difference) is the difference between the current index and left index if it is less than a predefined threshold. These cases occur with different probabilities; a binary prefix is allocated to each case which is generated using Huffman coder. The global Huffman codes lengths generated in simulation are given in table (6.3).

For the small codebook size of 64, the number of indices that are encoded using full index is small fraction of the total number of indices, and this result in longer binary codes (4 bits) than others. This is not the case with higher codebook sizes where the indices correlation decreases and the indices that are encoded with full index are more probable.

Table (6.3): Global Huffman codes' lengths for different cases used for the enhanced Hu and Chang method

Case \ N	64	128	256	512	1024
UM	1	2	2	2	4
LM	3	3	2	3	4
UD	2	1	2	2	2
LD	4	4	3	3	3
FI	4	4	3	2	1

Table (6.4) is the same as table (6.2) but for the enhanced Hu and Chang method. From both tables we note that the percentage of full index coded indices in the Hu and Chang method is higher than the corresponding percentage for the enhanced Hu and Chang method, so that the expected bit rate is less in the enhanced Hu and Chang method than the Hu and Chang method. Fig (6.3) compares between the Hu and Chang method with the enhanced Hu and Chang method (in the cases of global and local tables).

Table (6.4): Average cases percentage for the enhanced Hu and Chang method

Case \ N	64	128	256	512	1024
UM	38.5309	31.9012	24.6503	19.9109	13.0212
LM	17.0044	15.6537	13.4399	11.6821	9.1895
UD	40.9241	40.3833	36.0809	26.6711	20.5035
LD	2.7881	7.3810	11.3385	12.4738	11.4807
FI	0.7526	4.6808	14.4904	29.2621	45.8051

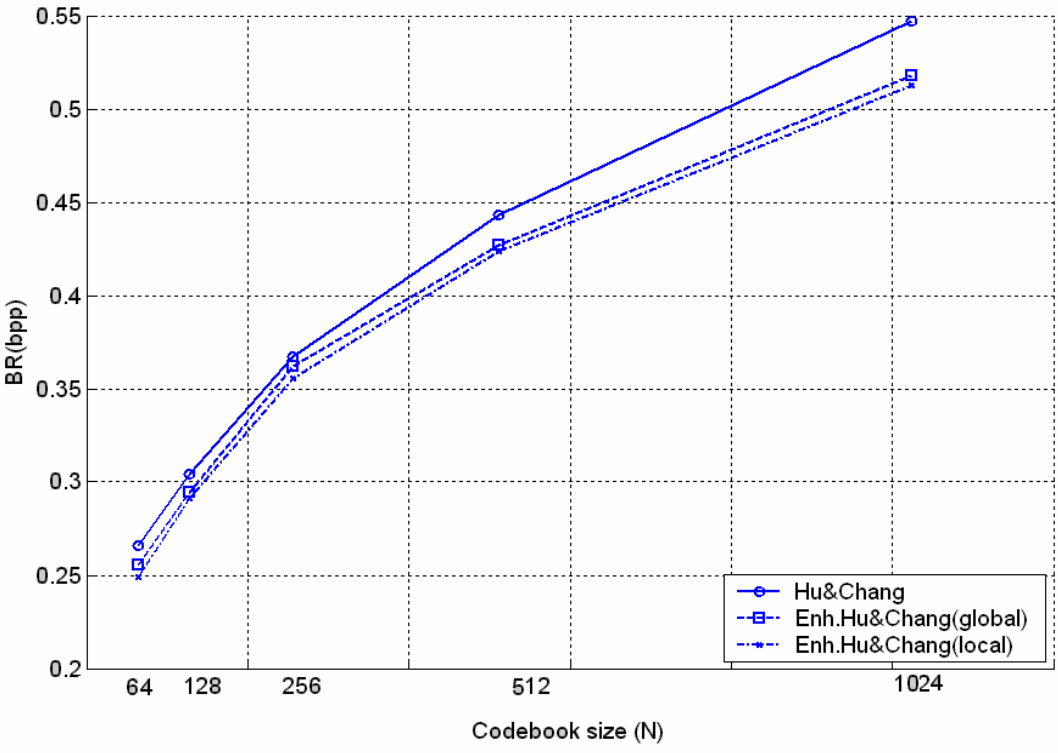


Fig (6.3): The average bit rate (in bpp) against codebook size (N) for the enhanced Hu and Chang method (with global and local Huffman codes) and the original Hu and Chang method

6.5. Repeated Adjacencies Method Simulation

In the proposed repeated adjacencies method the bit rate decreases with decreasing the percentage of the indices that need full index coding. Tables (6.5), (6.6), and (6.7) show the percentage of the different cases for repeated adjacencies method. The three methods ("Next", "Right", and "Next"+"Right") almost have the same percentages for the different cases.

Table (6.5): Average cases' percentages for the Repeated Adjacencies method (the "Next" case)

Case \ N	64	128	256	512	1024
UM	40.3998	34.7687	28.3380	25.4187	20.8099
LM	15.9973	14.3982	12.1967	10.1898	7.7960
UD	39.6954	38.5321	33.8257	24.6198	19.1510
LD	2.9895	7.2760	10.8435	11.4349	9.7345
FI	0.9180	5.0250	14.7961	28.3368	42.5085

Table (6.6): Average cases' percentages for the Repeated Adjacencies method (the "Right" case)

Case \ N	64	128	256	512	1024
UM	41.6956	35.8789	28.9752	25.9601	20.8600
LM	14.9567	13.4216	11.8835	9.8645	7.6465
UD	39.3378	38.2318	33.2355	23.8281	17.9938
LD	3.0756	7.4725	11.2292	12.0746	10.9863
FI	0.9344	4.9951	14.6765	28.2727	42.5134

Table (6.7): Average cases' percentages for the Repeated Adjacencies method (the "Next + Right" case)

Case \ N	64	128	256	512	1024
UM	40.3998	34.7687	28.3380	25.4187	20.8099
LM	16.8103	15.7581	14.4257	13.5876	12.8729
UD	38.9447	37.5287	32.6093	23.5040	18.1189
LD	2.9480	7.1552	10.6586	11.2341	9.8468
FI	0.8972	4.7894	13.9685	26.2555	38.3514

Fig (6.4) shows the distribution of the indices that are full index coded for the image Lena for codebook sizes of 64, 256, and 1024 when encoded with Hu and Chang, enhanced Hu and Chang, and repeated adjacencies method ("Next"). The enhanced Hu and repeated adjacencies method has fewer indices that are encoded using full index than Hu and Chang method and they are, as expected, concentrated at the objects' edges in the image.

When analyzing the "Next" and the "Right" tables for the different images we find that there are some indices which are the most probable next indices or the most probable right indices to themselves. Tables (6.8) and (6.9) show the average percentage of these indices based on ten test images.

Fig (6.5), (6.6), and (6.7) compare between different index compression methods, namely: the Huffman coder (in the cases of global and local tables), the SOC, and the Hu and Chang method with the repeated adjacencies method (“Next”,” Right”, and “Next” + “Right” cases, respectively).

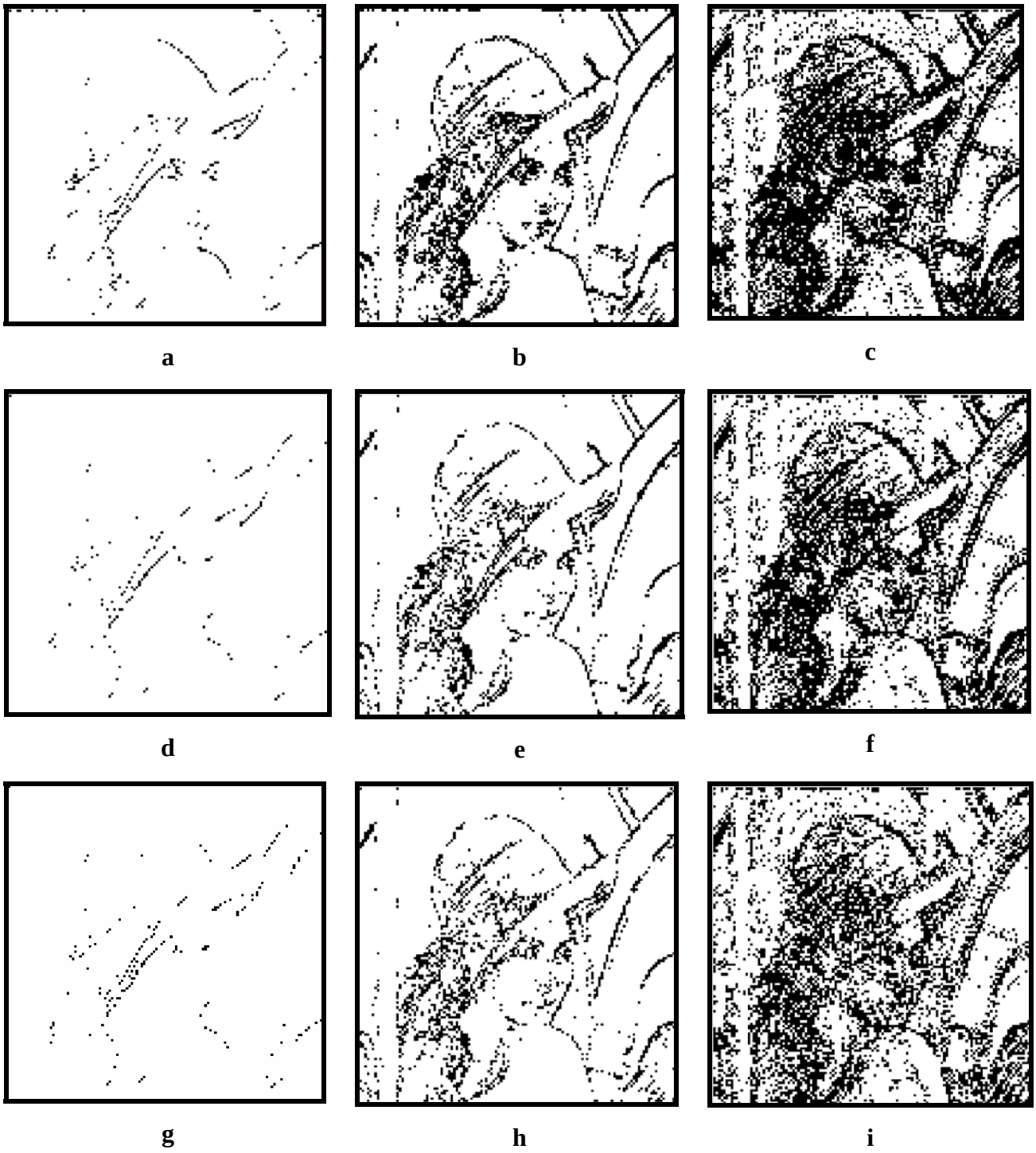


Fig (6.4): Distribution of indices that are full index coded for the Lena image

a), b), and c) Hu and Chang method with codebook sizes 64, 256, and 1024, respectively.

e), f), and g) enhanced Hu and Chang method with codebook sizes 64, 256, and 1024, respectively.

g), h), and i) repeated adjacencies method ("Next") with codebook sizes 64, 256, and 1024, respectively.

Table (6.8): percentage of indices that are the most probable next indices of themselves

Case \ N	64	128	256	512	1024
Lena	32.8125	17.9688	12.5000	5.8594	3.3203
Man	43.7500	27.3438	22.2656	9.5703	4.7852
GH	48.4375	33.5938	26.1719	17.9688	10.5469
Barbara	45.3125	39.8438	28.5156	17.5781	10.2539
Baboon	39.0625	27.3438	14.8438	6.8359	2.7344
Airplane	62.5000	46.0938	29.6875	15.2344	8.5938
Elaine	34.3750	21.0938	13.2813	8.2031	4.0039
Peppers	53.1250	39.0625	24.6094	14.6484	8.3008
Bridge	70.3125	57.0313	38.2813	23.6328	12.9883
Boats	53.1250	42.1875	27.3438	17.7734	11.9141
Ave.	**48.2813**	**35.1563**	**23.7500**	**13.7305**	**7.7441**

Table (6.9): percentage of indices that are the most probable right indices of themselves

Case \ N	64	128	256	512	1024
Lena	50.0000	37.5000	27.3438	16.6016	11.3281
Man	46.8750	35.1563	23.0469	14.6484	7.3242
GH	54.6875	42.1875	27.7344	16.9922	9.5703
Barbara	46.8750	35.1563	27.3438	16.0156	11.2305
Baboon	34.3750	16.4063	11.7188	5.6641	4.0039
Airplane	34.3750	18.7500	14.8438	6.4453	3.1250
Elaine	45.3125	27.3438	17.5781	9.9609	6.8359
Peppers	53.1250	32.8125	23.8281	15.4297	10.6445
Bridge	48.4375	36.7188	20.7031	11.5234	4.8828
Boats	45.3125	35.1563	25.3906	14.8438	11.2305
Ave.	**45.9375**	**31.7188**	**21.9531**	**12.8125**	**8.0176**

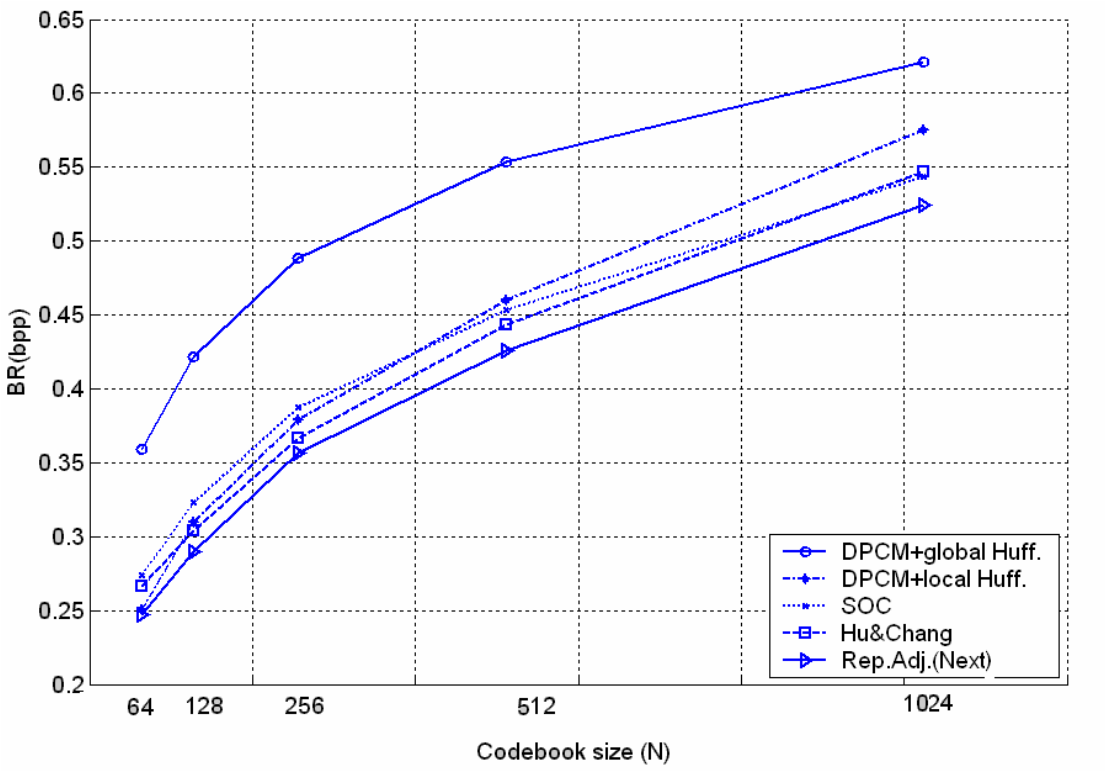


Fig (6.5): The average bit rate (in bpp) against codebook size (N) of repeated adjacencies ("Next" table)

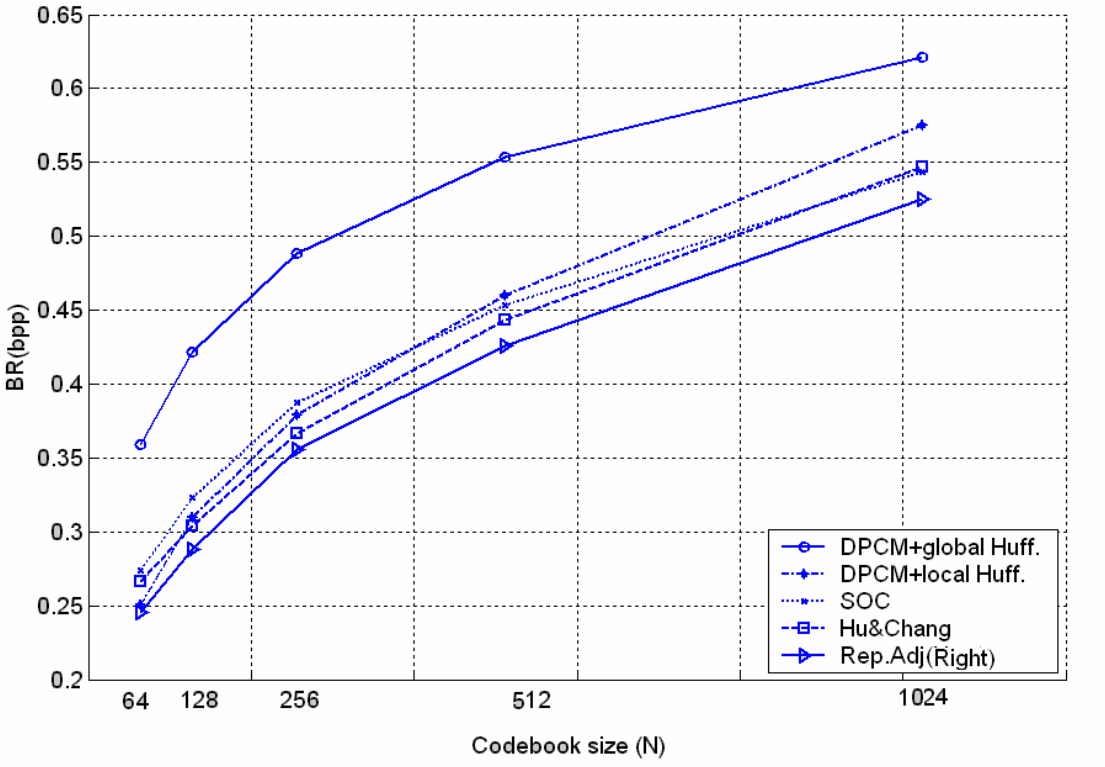


Fig (6.6): The average bit rate (in bpp) against codebook size (N) of repeated adjacencies ("Right" table)

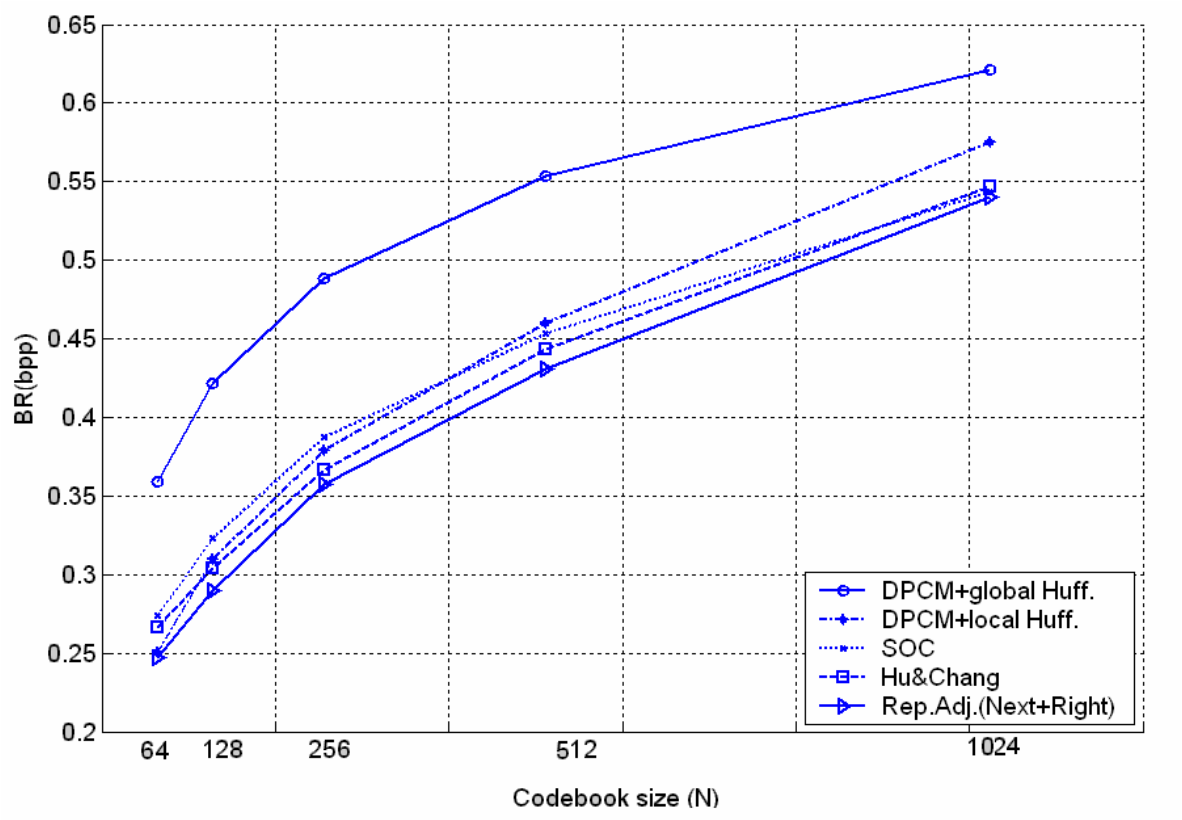


Fig (6.7): The average bit rate (in bpp) against codebook size (N) of repeated adjacencies ("Next" and "Right" tables)

6.6. VQ Index Remapping

After vector quantization of the image, the resulting *2D index map* is constructed and divided into *index blocks* with size 32 x 32 indices each. The used indices in each index block are defined and the corresponding index maps are generated. The remapping process is done based on these maps where the used indices in each index block are renumbered. This limits the maximum number of indices and reduces the bits needed to represent them. Table (6.10) shows the average bit rate overhead needed by the index usage maps for ten images and its percentage to the full index bit rate. The cost of the overhead caused by the index usage maps increases with the codebook size.

Table (6.10): Average index usage maps overhead over bit rate

N	64	128	256	512	1024
BR (bpp)	0.0038	0.0069	0.0131	0.0254	0.0501
BR (%)	1.0133	1.5771	2.6200	4.5156	8.0160

The minimum, mean, and maximum percentage of index usage in the different index blocks of different images is shown on table (6.11). These percentages indicate if the image has many details or not; the image with large percentages has more details than the image with low percentages. Also it indicates that when using codebooks with larger size the codevectors are less repeated than using smaller codebook size. For small codebook sizes the number of used indices is near to the codebook size with an average value of about 64% for codebook size of 64 while the percentage of the used indices in each index block decreases with increasing the codebook size. It reaches 25% for codebook size of 1024.

6.7. Remapped Index Compression

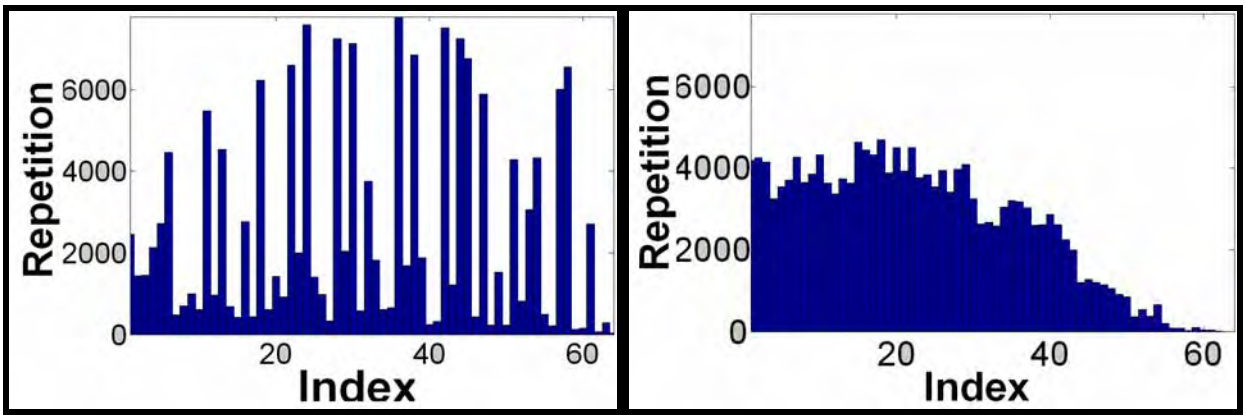
Different index compression algorithms such as SOC and Hu and Chang, enhanced Hu and Chang, and repeated adjacencies methods can with benefit use index remapping. The remapping process affects only the absolute value of indices and reduces the average difference between adjacent indices.

6.7.1. Huffman Coder with Index Remapping

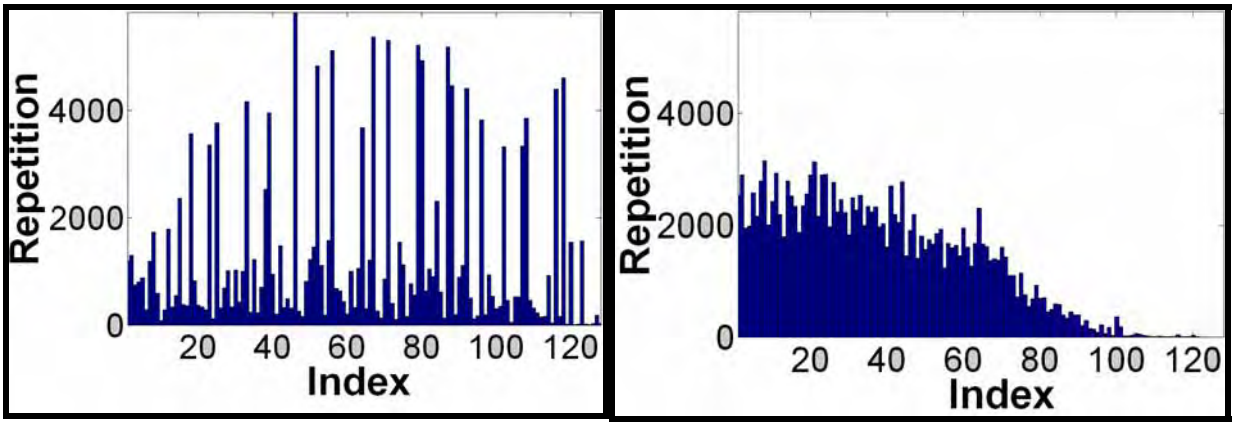
The remapping affects the histogram of the indices. Fig (6.8) shows the repetition of the normal and remapped indices averaged over ten test images. The number of indices is less in the case of the remapped indices than that of the normal indices. It is found that the low value indices have higher probability than high value indices. The probability of an index mostly decreases with increasing its value. This is because the remapping process

Table (6.11): The minimum, mean, and maximum percentages of index usage for the index blocks.

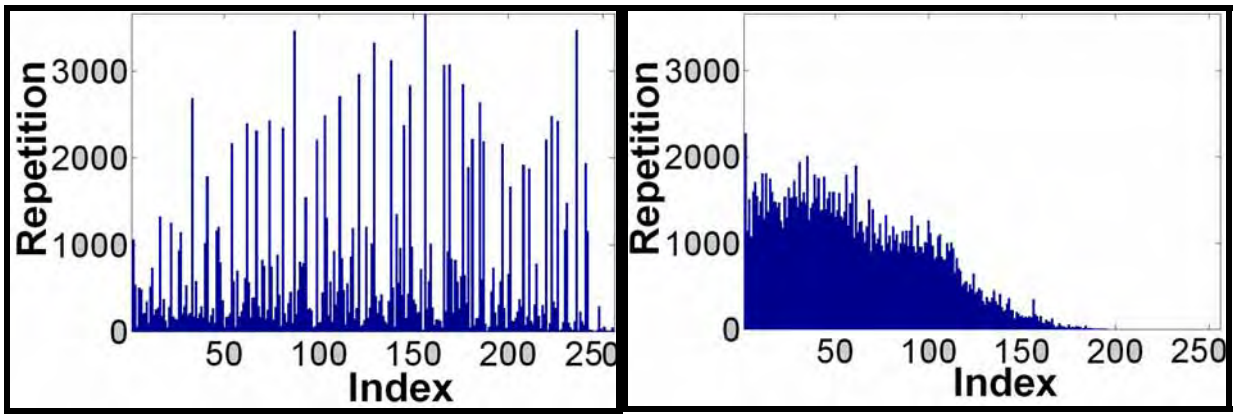
Image \ N		64	128	256	512	1024
Lena	Min.	28.1250	20.3125	15.2344	10.9375	9.3750
	Mean.	61.3281	50.9277	39.0625	28.0518	20.9656
	Max.	81.2500	71.0938	60.1563	46.6797	34.3750
Man	Min.	56.2500	49.2188	37.1094	25.7813	19.1406
	Mean.	75.0977	66.2598	55.2490	43.7622	32.2388
	Max.	89.0625	83.5938	73.4375	63.4766	46.3867
Gold hill	Min.	35.9375	26.5625	20.7031	16.6016	14.4531
	Mean.	59.3750	48.2422	39.8682	30.6030	22.9797
	Max.	78.1250	66.4063	56.2500	44.1406	33.9844
Barbara	Min.	43.7500	33.5938	24.6094	17.7734	13.1836
	Mean.	65.7227	57.4219	46.8506	35.9375	26.7639
	Max.	85.9375	77.3438	65.2344	44.5313	35.1563
Baboon	Min.	45.3125	39.0625	33.9844	27.1484	20.3125
	Mean.	65.2344	58.1543	50.5127	41.3208	31.8298
	Max.	78.1250	77.3438	65.2344	58.7891	44.9219
Airplane	Min.	10.9375	8.5938	6.2500	4.4922	3.3203
	Mean.	56.2500	46.9727	36.8164	26.8555	19.3481
	Max.	89.0625	80.4688	62.8906	49.6094	38.0859
Elaine	Min.	18.7500	10.9375	9.7656	8.0078	7.5195
	Mean.	56.7383	45.1172	35.0830	26.4893	20.5627
	Max.	85.9375	70.3125	54.2969	40.8203	30.9570
Peppers	Min.	59.3750	47.6563	35.9375	25.1953	17.9688
	Mean.	73.0469	61.6699	47.2168	34.6313	24.8901
	Max.	90.6250	78.9063	61.7188	46.8750	32.9102
Bridge	Min.	50.0000	42.9688	35.5469	28.9063	24.3164
	Mean.	73.5352	66.1621	56.8359	45.3735	34.3811
	Max.	96.8750	94.5313	76.1719	60.9375	42.4805
Boats	Min.	14.0625	13.2813	7.0313	5.6641	4.5898
	Mean.	59.3750	50.4883	41.0400	31.1890	22.9797
	Max.	82.8125	75.0000	64.8438	53.1250	40.9180
Average	**Min.**	**36.2500**	**29.2188**	**22.6172**	**17.0508**	**13.4180**
	Mean.	**64.5703**	**55.1416**	**44.8535**	**34.4214**	**25.6940**
	Max.	**85.7813**	**77.5000**	**64.0235**	**50.8985**	**38.0176**



a) Normal indices N=64 g) remapped indices N=64

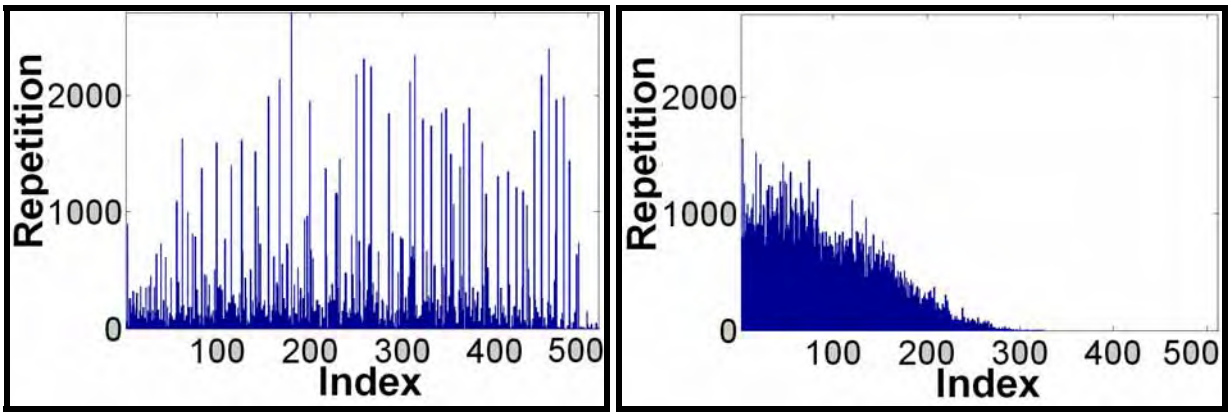


b) Normal indices N=128 h) remapped indices N=128

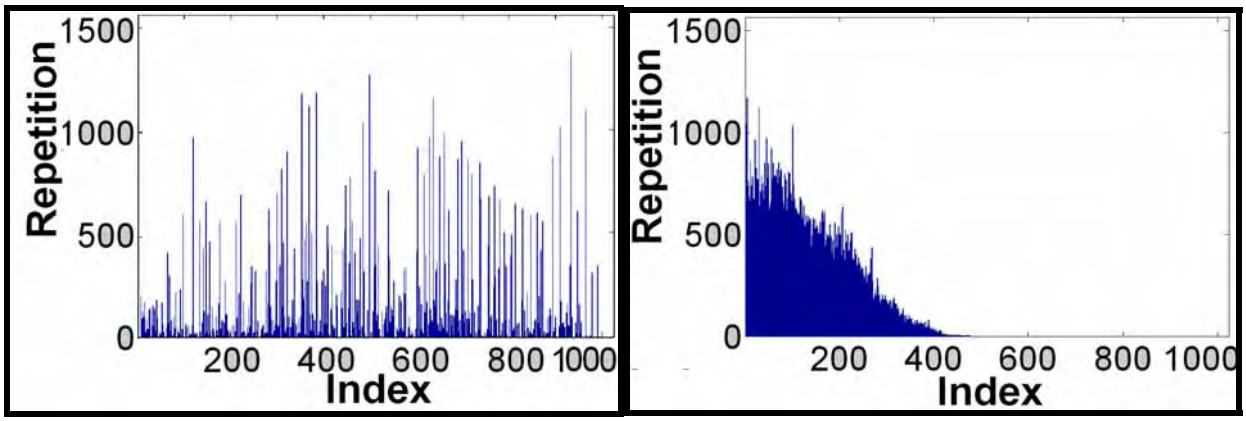


c) Normal indices N=256 i) remapped indices N=256

Fig (6.8): The effect of remapping on the indices histogram of different codebook sizes



d) Normal indices N=512 j) remapped indices N=512



e) Normal indices N=1024 h) remapped indices N=1024

Fig (6.8) (cntd.): The effect of remapping on the indices histogram of different codebook sizes

always starts with index value one and increases with the count of the used indices in each index block. This is not the case with the normal indexing histogram where the index probability does not depend on the index value.

The remapped indices probability distribution affects the global Huffman codes, since the indices with close index values have close probability distribution, so that the average indices' differences is less than normal indices. Huffman codes bit length distribution is more concave than the case of normal indexing, see Fig (6.9).

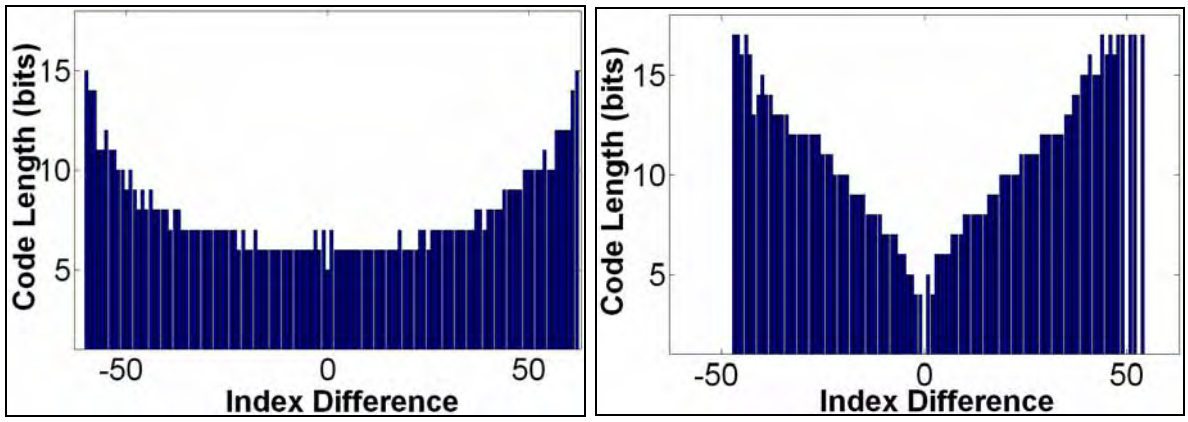


Fig (6.9): Length distribution of global Huffman codes for normal and remapped indices differences (codebook size N=64)

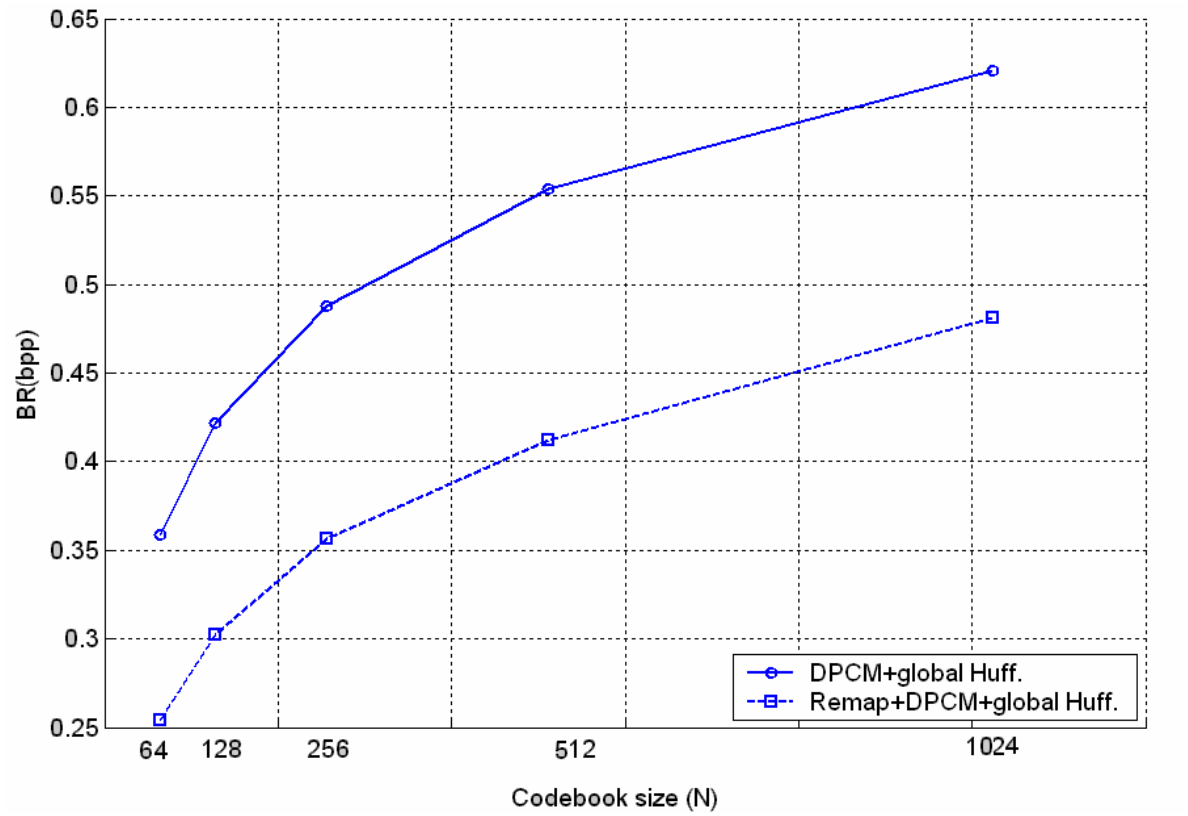


Fig (6.10): The average bit rate (in bpp) against codebook size (N) for the DPCM+Huffman coder (global codes) of the normal and remapped indices

Fig (6. 10) compares the average bit rate of the remapped indices when index compressed with DPCM and global Huffman codes with normal indices. Also Fig (6.11) shows the comparison for the case of using local Huffman codes.

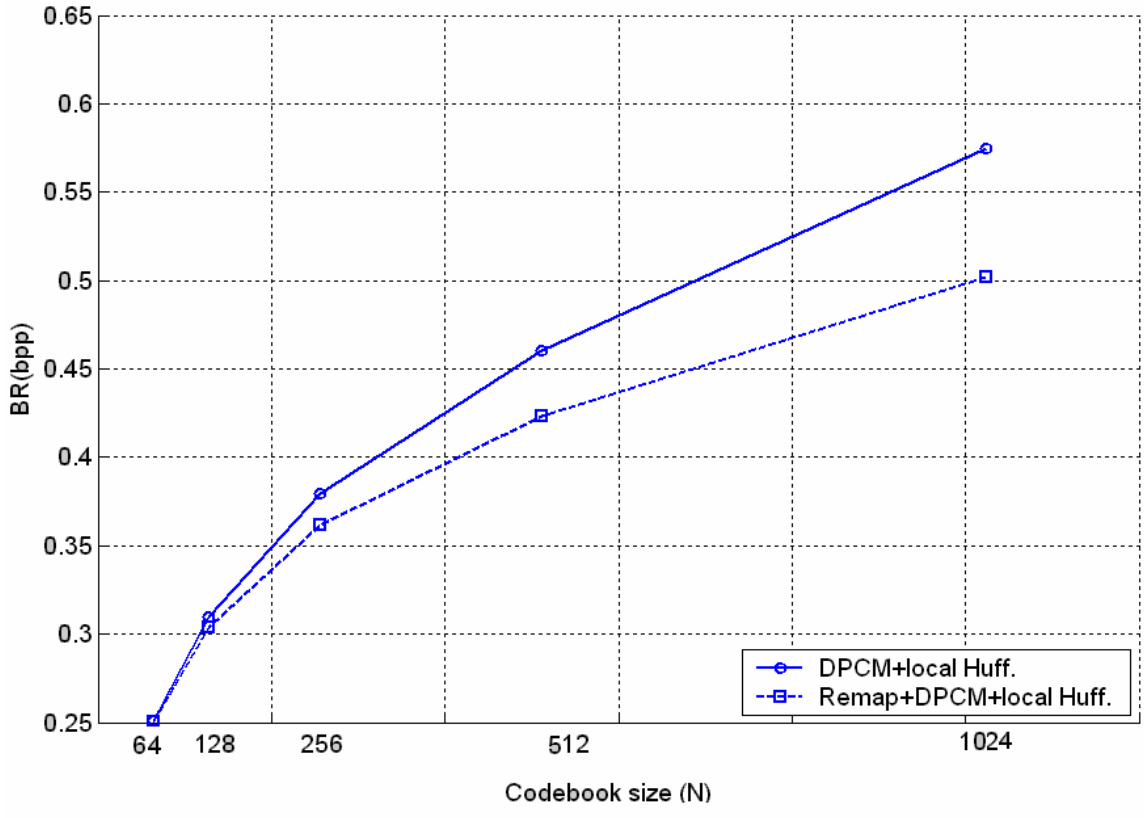


Fig (6.11): The average bit rate (in bpp) against codebook size (N) for the DPCM + Huffman (local codes) of the normal and remapped indices

6.7.2. Hu and Chang Method with Index Remapping

The statistics of image indices are altered after remapping. When comparing table (6.12) with table (6.2), the percentage of indices that are encoded using full index is less in the case of the remapped indices than the case of the normal indices. And the percentage reduction increases with the increase in codebook size. The average difference between adjacent indices is decreased.

Table (6.12): Average cases' percentages for the Hu and Chang method of the remapped indices

Case \ N	64	128	256	512	1024
UM	37.8088	31.2958	24.1675	19.5001	12.7802
LM	17.0856	15.6506	13.3734	11.6174	9.0613
UD	42.2015	43.6853	43.9069	40.4510	39.6930
FI	2.9041	9.3683	18.5522	28.4314	38.4656

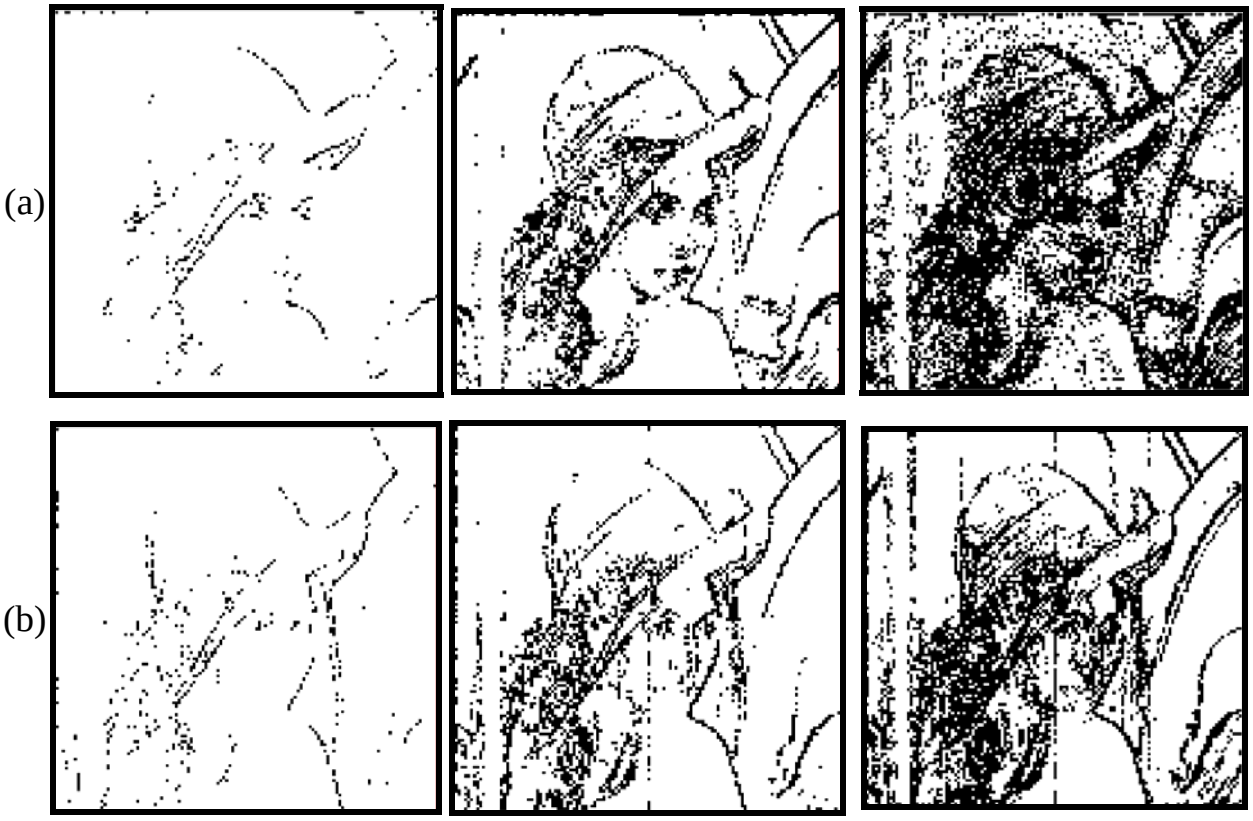


Fig (6.12): Distribution of indices that are full index coded using the Hu and Chang method for the Lena image (a) without remapping (b) with remapping

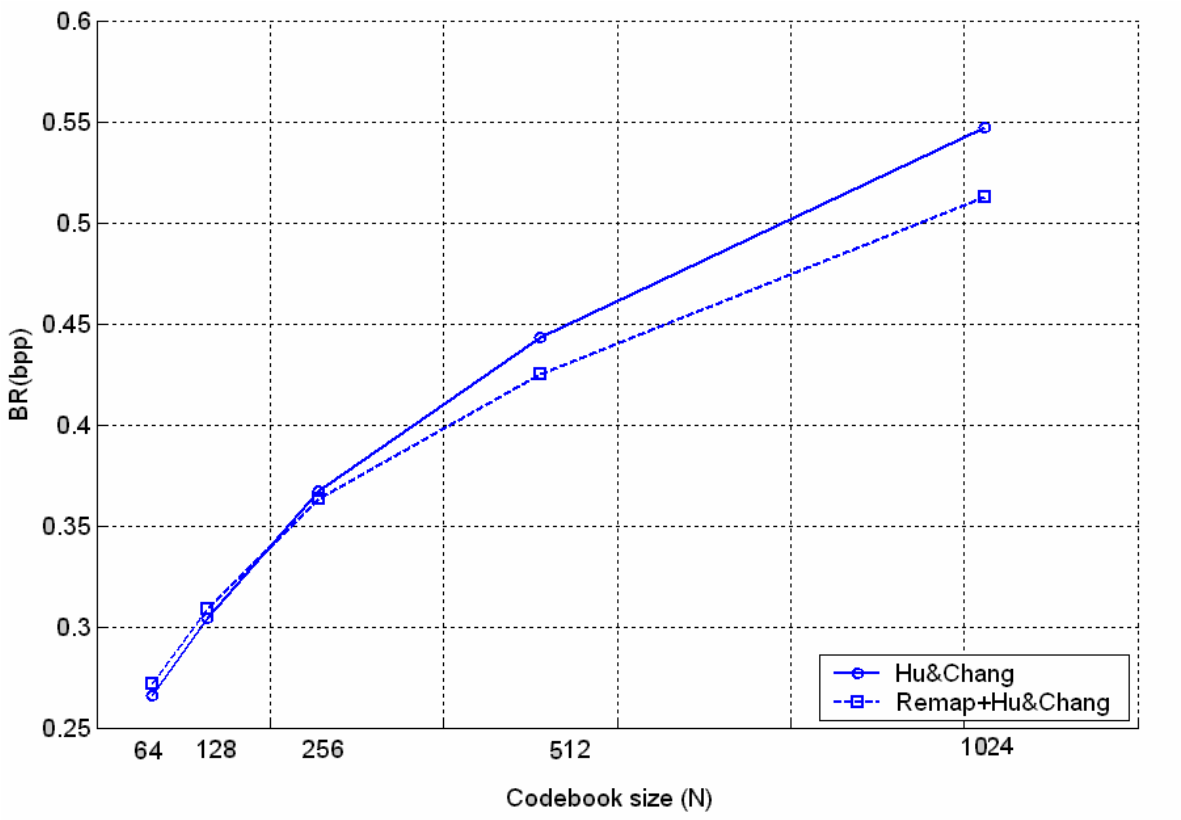


Fig (6.13): The average bit rate (in bpp) against codebook size (N) for the Hu and Chang method for the normal and remapped indices

Fig (6.13) compares the average bit rate of the Hu and Chang method with normal and remapped indices. Note that the overhead for sending the needed remapping information exceeds the improvement due to remapping in case of low bit rates (small codebooks sizes).

6.7.3. Enhanced Hu and Chang Method with Index Remapping

The same effect of remapping found with the Hu and Chang method is present also in the enhanced Hu and Chang method. Compare table (6.13) with table (6.4).

Table (6.13): Average cases' percentages for the enhanced Hu and Chang method of the remapped indices

Case \ N	64	128	256	512	1024
UM	37.8088	31.2958	24.1675	19.5001	12.7802
LM	17.0856	15.6506	13.3734	11.6174	9.0613
UD	42.2015	43.6853	43.9069	40.4510	39.6930
LD	2.3303	6.1249	9.7150	12.1564	13.8287
FI	0.5737	3.2434	8.8373	16.2750	24.6368

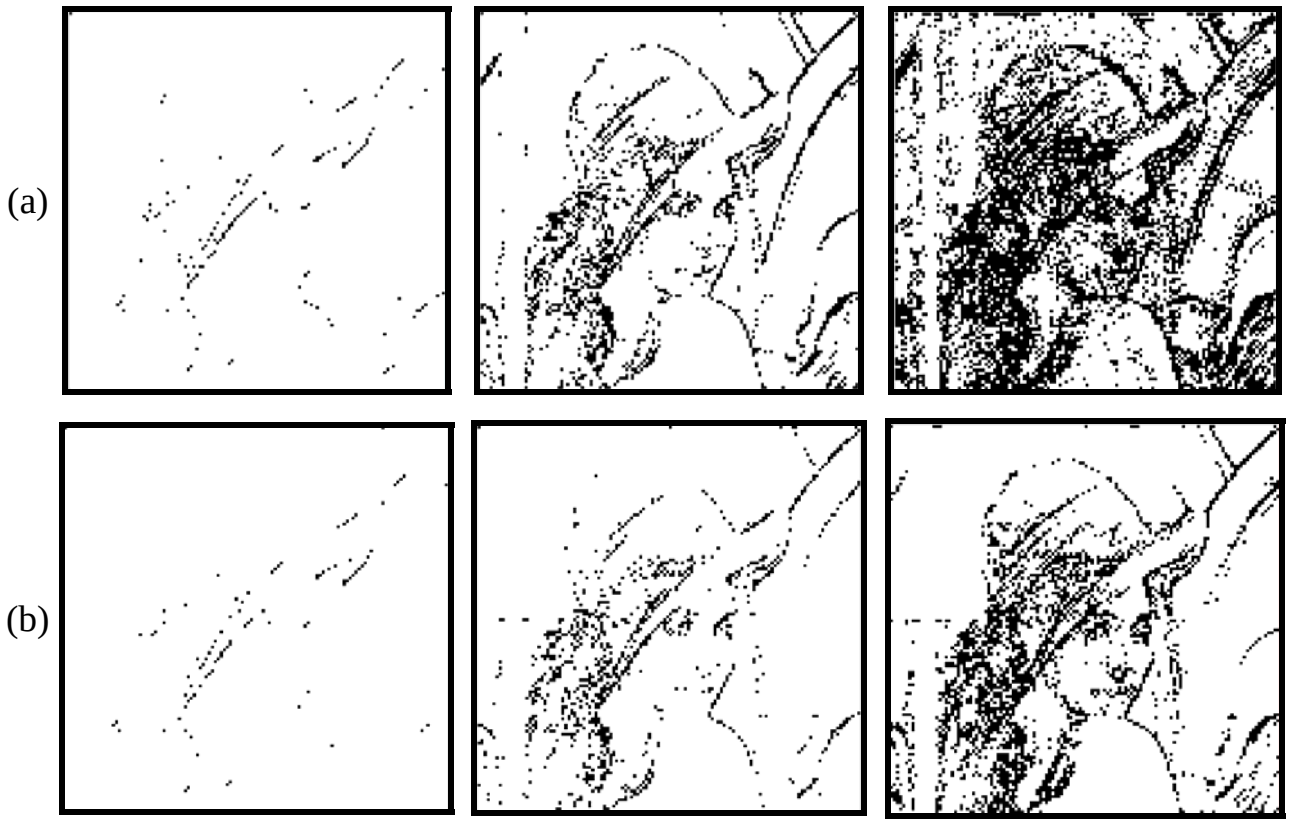


Fig (6.14) Distribution of indices that are full index coded using the enhanced Hu and Chang method for the Lena image (a) without remapping (b) with remapping

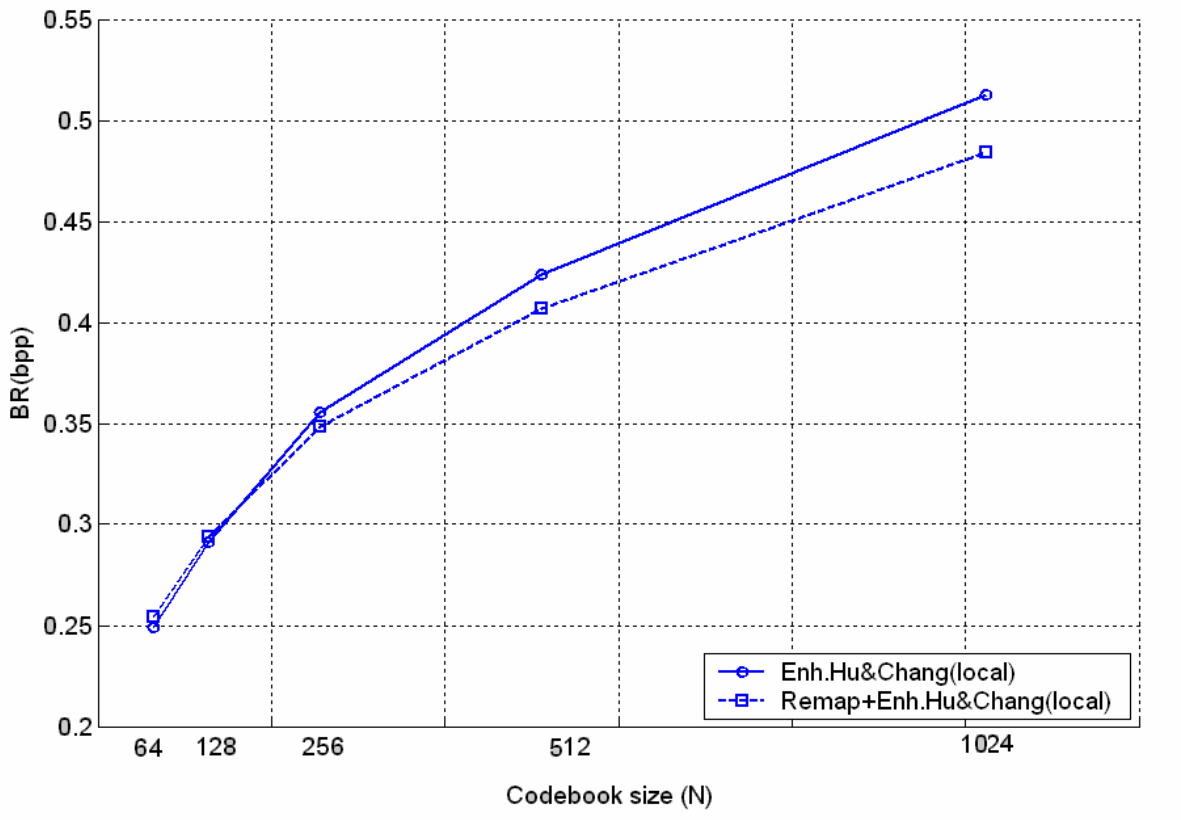


Fig (6.15): The average bit rate (in bpp) against codebook size (N) for the enhanced Hu and Chang method with local Huffman for the normal and remapped indices

Fig (6.15) compares between the average bit rate of the enhanced Hu and Chang method applied to the normal and remapped indices (with local Huffman tables). The same hold as for the Hu and Chang method, the overhead for sending the needed remapping information exceeds the improvement due to remapping in case of low bit rates (small codebooks sizes).

6.7.4. Repeated Adjacencies Method with Index Remapping

Applying the repeated adjacencies method after remapping results in the improvement shown in tables (6.14), (6.15), and (6.16) which should be compared with tables (6.5), (6.6), and (6.7), respectively.

Table (6.14): Average cases' percentages for the Repeated Adjacencies method of the remapped indices (the "Next" case)

Case \ N	64	128	256	512	1024
UM	38.0499	31.9531	25.3052	21.6840	16.0950
LM	16.8359	15.1855	12.8149	10.6976	8.1488
UD	41.7365	42.6178	42.3627	38.4943	37.2205
LD	2.6404	6.5143	9.9097	12.0447	13.3423
FI	0.7373	3.7292	9.6075	17.0795	25.1935

Table (6.15): Average cases' percentages for the Repeated Adjacencies method of the remapped indices (the "Right" case)

Case \ N	64	128	256	512	1024
UM	39.0186	32.9260	25.7056	22.0673	16.3239
LM	15.9271	14.2218	12.4689	10.3668	7.8259
UD	41.6779	42.5909	42.2815	38.3362	37.0306
LD	2.6392	6.5393	9.9371	12.1637	13.5889
FI	0.7373	3.7219	9.6069	17.0660	25.2307

Table (6.16): Average cases' percentages for the Repeated Adjacencies method of the remapped indices (the "Next + Right" case)

	64	128	256	512	1024
UM	38.0499	31.9531	25.3052	21.6840	16.0950
LM	16.8103	15.2856	13.3014	11.7694	10.0604
UD	41.7664	42.5153	41.9910	37.8088	36.1725
LD	2.6367	6.5302	9.8566	11.9476	13.1165
FI	0.7367	3.7158	9.5459	16.7902	24.5557

The same effect can be found by the distribution of indices that are full index coded in the case of Lena image

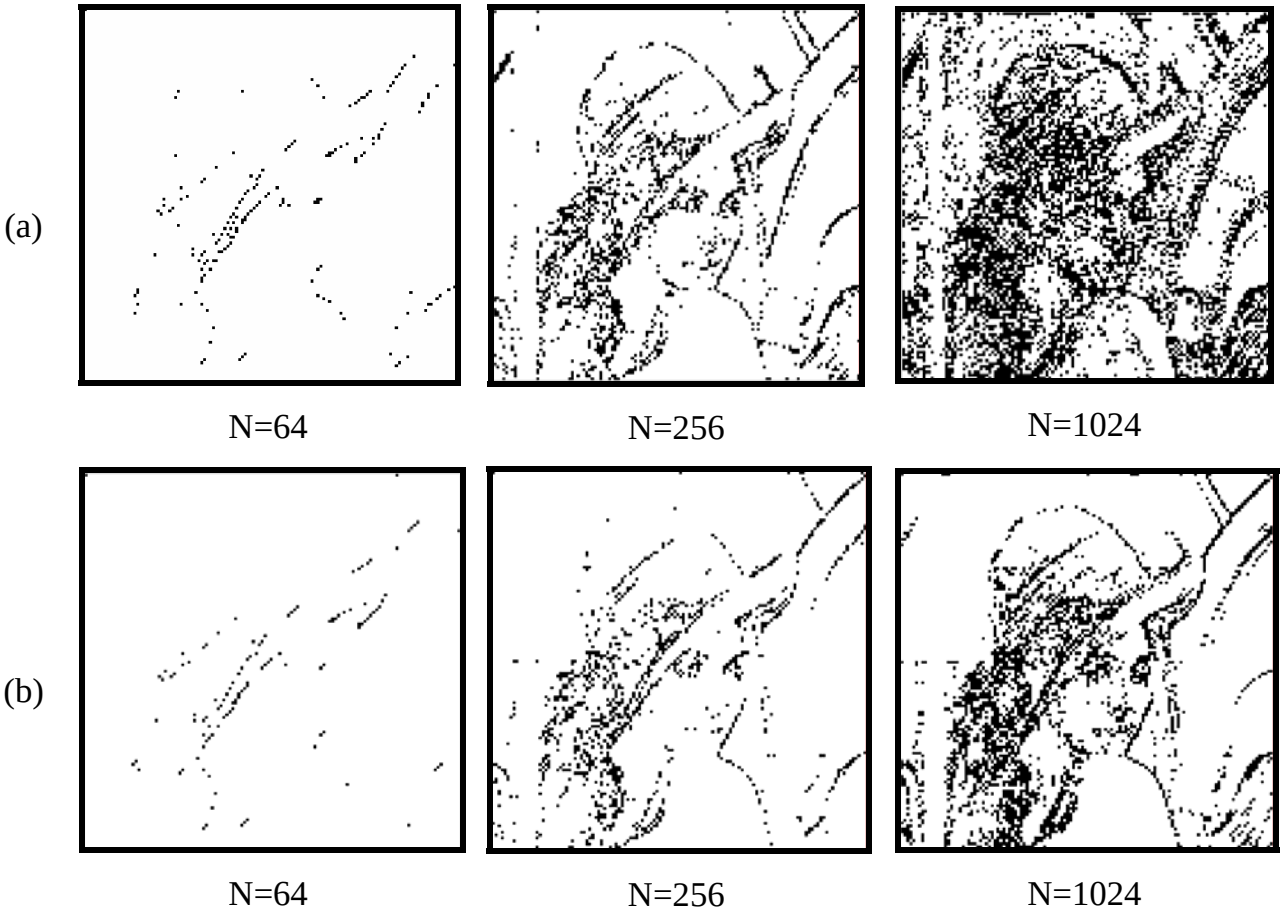


Fig (6.16): Distribution of indices that are full index coded using the repeated adjacencies method for the Lena image (a) without remapping (b) with remapping

The number of the codevectors which are the most probable next index or the most probable right index to themselves is increased in the case of the remapped indices than in the case of the normal indices. This is logic because of the correlation increase in the former method. Tables (6.17) and (6.18) show this percentage for different images that are vector quantized with codebooks of sizes of 64, 128, 256, 512, and 1024. These tables are to be compared with tables (6.8), and (6.9) of the normal case.

Table (6.17): percentage of remapped indices that are the most probable next indices of themselves

Image \ N	64	128	256	512	1024
Lena	77.4194	74.1935	72.5806	79.0323	93.5484
Man	52.8926	52.8926	44.6281	53.7190	84.2975
GH	36.9231	53.3333	31.2821	45.6410	64.6154
Barbara	22.7692	41.2308	20.9231	28.9231	45.8462
Baboon	14.1053	30.9474	8.2105	20.0000	27.3684
Airplane	67.7419	74.1935	85.4839	88.7097	80.6452
Elaine	56.1983	61.1570	65.2893	65.2893	61.9835
Peppers	44.6154	57.9487	53.8462	53.3333	63.5897
Bridge	28.9231	39.0769	36.3077	36.3077	45.8462
Boats	12.8421	31.3684	28.0000	23.5789	36.4211
Ave.	79.3548	59.8347	50.5128	34.6154	23.2842

Table (6.18): percentage of remapped indices that are the most probable right indices of themselves

	64	128	256	512	1024
Lena	79.0323	72.5806	61.2903	83.8710	69.3548
Man	65.2893	54.5455	48.7603	66.9421	50.4132
GH	60.0000	50.2564	28.2051	53.3333	40.0000
Barbara	46.4615	31.6923	17.5385	39.3846	19.6923
Baboon	37.6842	23.7895	11.7895	28.8421	10.7368
Airplane	69.3548	80.6452	77.4194	87.0968	79.0323
Elaine	62.8099	63.6364	45.4545	70.2479	59.5041
Peppers	51.2821	56.9231	38.4615	58.4615	48.7179
Bridge	38.4615	40.9231	24.9231	42.1538	36.6154
Boats	20.6316	30.5263	16.8421	34.5263	28.0000
Ave.	75.9677	58.7603	48.5641	33.7846	24.3368

Fig (6.17), (6.18), and (6.19) compare the repeated adjacencies method ("Next", "Right", and "Next" + "Right" cases respectively) with the remapped indices of the same methods. Note that the overhead for sending the needed remapping information exceeds the improvement due to remapping in case of low bit rates (small codebooks sizes).

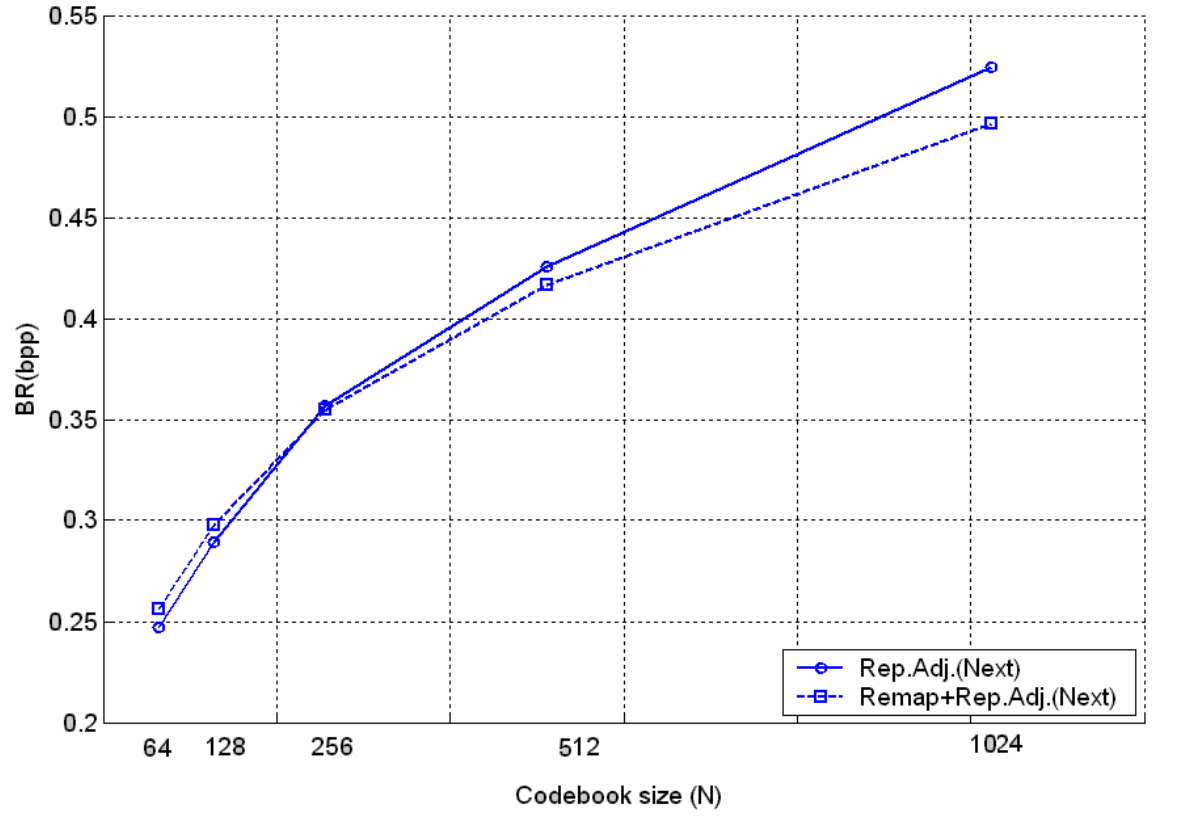


Fig (6.17): The average bit rate (in bpp) against codebook size (N) for repeated adjacencies (“Next” table) of the normal and remapped indices

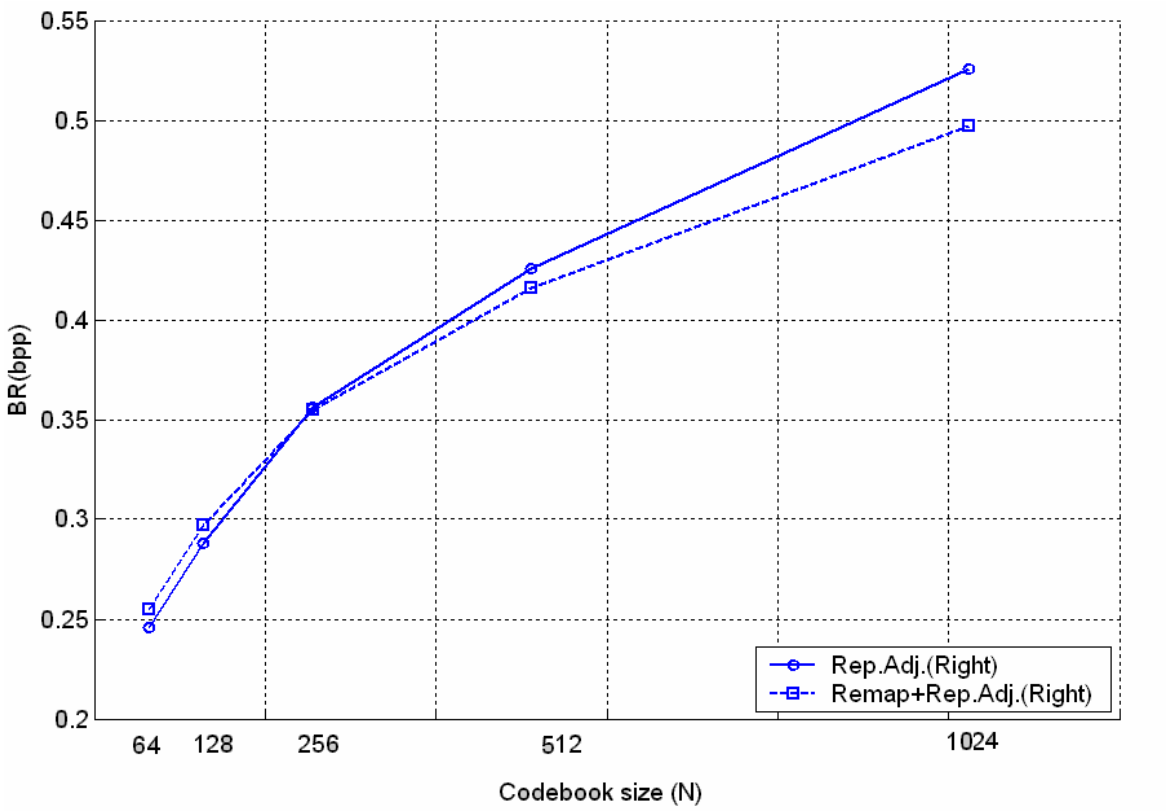


Fig (6.18): The average bit rate (in bpp) against codebook size (N) for repeated adjacencies ("Right" table) of the normal and remapped indices

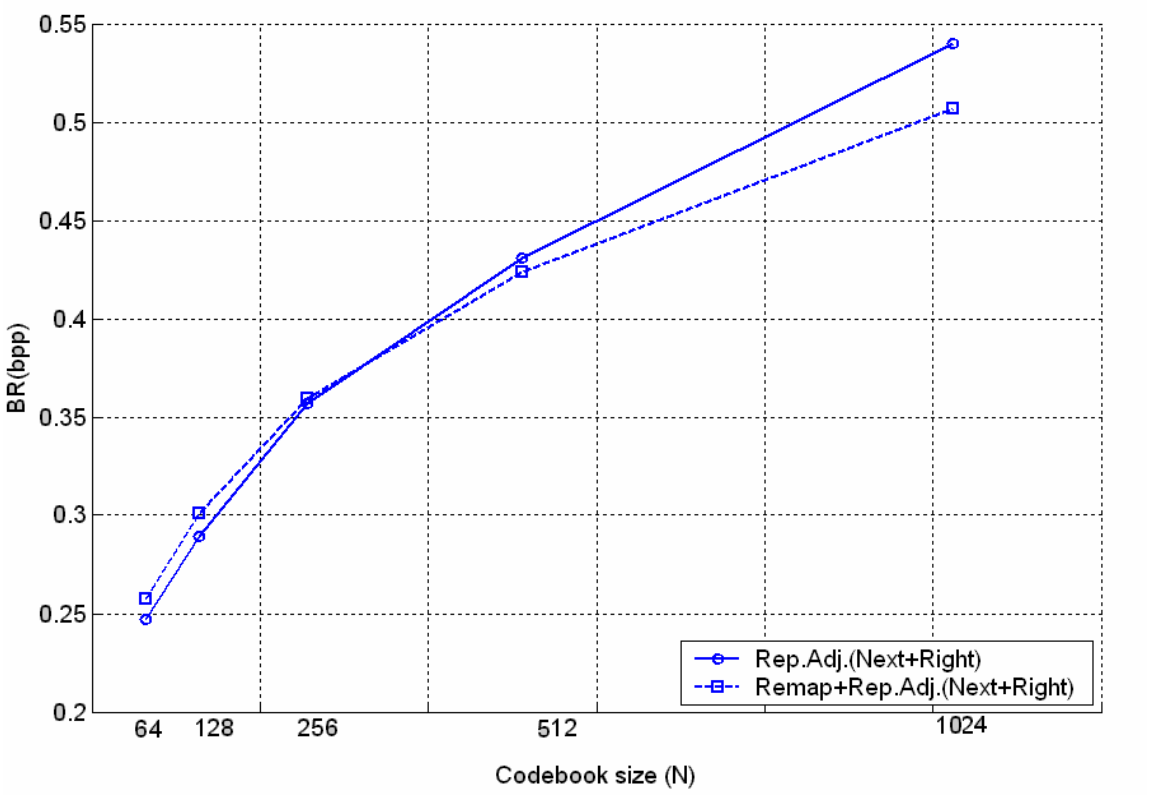


Fig (6.19): The average bit rate (in bpp) against codebook size (N) for repeated adjacencies ("Next" and "Right" table) of the normal and remapped indices

6.7.5. Search Order Coding Method with Index Remapping

The search order coding method can be used with the remapped indices. The number of indices that are encoded with the search order does not change a lot in the case of the remapped indices than that of the normal indices, so the improvement in bit rate between the two cases is negligible. SOC is based on searching an exact match to the current index in the neighboring area. After remapping if the matched index is located in the same index block, it will be coded with the search order with same code as that used for the normal indices, but if it is located in a different index block, it will most probably acquire another index value and no match will be found.

6.8. Comparing Index Compression Algorithms

Tables (6.19), (6.20), (6.21), (6.22), and (6.23) show the bit rates (in bpp) of the encoded indices for five different codebook with sizes of 64, 128, 256, 512, and 1024 covering wide range of practical bit rates. Ten images are used in simulations to cover different image statistics.

Huffman coding, SOC, Hu and Chang method, enhanced Hu and Chang method and the repeated adjacencies applied to normal and remapped indices are presented in one table for the purpose of comparison.

It is clear from the results that best performance of index compression algorithms is found when using codebooks with small size, where indices of inherently high correlation are found. The performance is decreased with increasing the codebook size.

For small codebook sizes the repeated adjacencies method gives the best performance among other algorithms.

The average performance of the new proposed methods is better than other demonstrated methods. For the repeated adjacencies, the three methods are almost the same in the case of the normal indices or in the case of the

remapped indices. For individual images a small difference between the three methods is found and it depends on the image orientation and if the repeated number of blocks in the horizontal or vertical direction is greater. For all images using both the "Next and Right" tables does not improve the performance over the "Next" or "Right" methods due to the need to store two tables instead of one as an overhead.

For large codebook sizes the remapping improves the performance while for small codebook sizes the improvement is limited. The best performance is found when combining the remapping with the enhanced Hu and Chang method.

The index compression performance also depends on the image activity. Comparing the bit rate for individual images for each compression method we note that the high activity image "baboon" gives the worst bit rate for all the compression methods while the low activity image "airplane" gives the best performance.

Table (6.24) indicates the average performance improvement for different index compression methods. For small codebook sizes of 64 and 128 the repeated adjacencies method gives better results than other methods, the improvement achieved is 34% over the full index bit rate. For moderate codebook sizes of 256 and 512 the remapping combined with the enhanced Hu and Chang method gives the best results with 30% and 27% improvement over full index bit rate. For large codebook size of 1024 the remapping combined with global Huffman coding gives the best improvement of 23%.

Table (6.19): Comparing the bit rate (in bpp) of different index compression methods for codebook with size 64

Image	Lena	man	gold hill	Barbara	baboon	airplane	Elaine	peppers	bridge	Boats	**average**
Huff.+DPCM (G)	0.3555	0.3657	0.3544	0.3616	0.3718	0.3480	0.3566	0.3552	0.3686	0.3516	**0.3589**
Huff. + DPCM (L)	0.2509	0.2756	0.2376	0.2675	0.2981	0.2019	0.2357	0.2352	0.2846	0.2218	**0.2509**
SOC	0.2495	0.2904	0.2695	0.2697	0.3292	0.2510	0.2511	0.2511	0.3207	0.2602	**0.2742**
Hu and Chang	0.2358	0.2884	0.2565	0.2683	0.3408	0.2249	0.2487	0.2371	0.3239	0.2373	**0.2662**
Enh. Hu and Chang (G)	0.2278	0.2833	0.2417	0.2614	0.3401	0.1994	0.2391	0.2261	0.3180	0.2171	**0.2554**
Enh. Hu and Chang (L)	0.2279	0.2740	0.2418	0.2579	0.3114	0.1995	0.2392	0.2262	0.2964	0.2173	**0.2492**
Rep. adj. (N)	0.2224	0.2742	0.2396	0.2586	0.3111	0.1966	0.2355	0.2242	0.2967	0.2147	**0.2474**
Rep. adj. (R)	0.2082	0.2708	0.2465	0.2527	0.3097	0.2030	0.2300	0.2173	0.2982	0.2184	**0.2455**
Rep. adj. (N+R)	0.2229	0.2751	0.2394	0.2589	0.3103	0.1949	0.2351	0.2236	0.2950	0.2137	**0.2469**
Remap+ Normal Indexing	0.3789	0.3790	0.3787	0.3788	0.3786	0.3784	0.3788	0.3792	0.3789	0.3785	**0.3788**
Remap+ DPCM +Huff. (L)	0.2464	0.2753	0.2383	0.2637	0.2967	0.2045	0.2325	0.2363	0.2865	0.2216	**0.2502**
Remap+DPCM +Huff. (G)	0.2470	0.2815	0.2381	0.2651	0.3165	0.2064	0.2315	0.2363	0.2997	0.2207	**0.2543**
Remap+ SOC	0.2552	0.2942	0.2748	0.2749	0.3333	0.2558	0.2568	0.2563	0.3248	0.2648	**0.2791**
Remap+ Hu and Chang	0.2414	0.2931	0.2626	0.2733	0.3457	0.2300	0.2547	0.2424	0.3295	0.2421	**0.2715**
Remap+Enh.Hu&Chang(L)	0.2333	0.2778	0.2481	0.2614	0.3150	0.2054	0.2454	0.2317	0.3012	0.2222	**0.2541**
Remap+Rep.adj.(N)	0.2349	0.2798	0.2498	0.2632	0.3180	0.2075	0.2466	0.2337	0.3037	0.2240	**0.2561**
Remap+Rep. adj.(R)	0.2182	0.2772	0.2543	0.2603	0.3172	0.2159	0.2406	0.2262	0.3072	0.2286	**0.2546**
Remap+Rep.adj.(N+R)	0.2364	0.2822	0.2511	0.2644	0.3194	0.2090	0.2480	0.2349	0.3049	0.2252	**0.2576**

Table (6.20): Comparing the bit rate (in bpp) of different index compression methods for codebook with size 128

Image	Lena	man	gold hill	Barbara	baboon	airplane	Elaine	peppers	bridge	Boats	**average**
Huff.+DPCM (G)	0.4196	0.4291	0.4188	0.4245	0.4329	0.4137	0.4204	0.4175	0.4308	0.4136	**0.4221**
Huff. + DPCM (L)	0.3066	0.3407	0.2968	0.3261	0.3625	0.2541	0.2959	0.2909	0.3500	0.2724	**0.3096**
SOC	0.2839	0.3495	0.3201	0.3203	0.3967	0.2880	0.2916	0.2864	0.3897	0.3022	**0.3229**
Hu and Chang	0.2673	0.3358	0.2934	0.3086	0.3859	0.2577	0.2822	0.2670	0.3706	0.2711	**0.3040**
Enh. Hu and Chang (G)	0.2688	0.3237	0.2790	0.3014	0.3660	0.2543	0.2735	0.2647	0.3478	0.2645	**0.2944**
Enh. Hu and Chang (L)	0.2653	0.3238	0.2791	0.3016	0.3629	0.2402	0.2737	0.2606	0.3475	0.2575	**0.2912**
Rep. adj. (N)	0.2582	0.3235	0.2794	0.3018	0.3648	0.2374	0.2705	0.2565	0.3490	0.2539	**0.2895**
Rep. adj. (R)	0.2456	0.3212	0.2815	0.2993	0.3649	0.2412	0.2673	0.2515	0.3514	0.2593	**0.2883**
Rep. adj. (N+R)	0.2576	0.3243	0.2792	0.3022	0.3645	0.2353	0.2706	0.2563	0.3492	0.2543	**0.2894**
Remap+ Normal Indexing	0.4447	0.4450	0.4441	0.4446	0.4441	0.4437	0.4443	0.4452	0.4447	0.4439	**0.4444**
Remap+ DPCM +Huff. (L)	0.2943	0.3363	0.2890	0.3174	0.3572	0.2560	0.2804	0.2856	0.3493	0.2705	**0.3024**
Remap+DPCM +Huff. (G)	0.2912	0.3305	0.2856	0.3126	0.3594	0.2644	0.2797	0.2829	0.3455	0.2719	**0.3036**
Remap+ SOC	0.2923	0.3563	0.3276	0.3282	0.4020	0.2953	0.3005	0.2955	0.3956	0.3092	**0.3302**
Remap+ Hu and Chang	0.2717	0.3407	0.2990	0.3130	0.3887	0.2634	0.2876	0.2734	0.3756	0.2764	**0.3090**
Remap+Enh.Hu&Chang(L)	0.2683	0.3256	0.2814	0.3014	0.3635	0.2454	0.2755	0.2659	0.3500	0.2618	**0.2939**
Remap+Rep.adj.(N)	0.2713	0.3304	0.2857	0.3054	0.3689	0.2489	0.2788	0.2696	0.3558	0.2655	**0.2980**
Remap+Rep. adj.(R)	0.2556	0.3277	0.2898	0.3032	0.3689	0.2556	0.2762	0.2637	0.3583	0.2709	**0.2970**
Remap+Rep.adj.(N+R)	0.2744	0.3336	0.2890	0.3082	0.3712	0.2517	0.2817	0.2727	0.3588	0.2691	**0.3010**

Table (6.21): Comparing the bit rate (in bpp) of different index compression methods for codebook with size 256

Image	Lena	man	gold hill	Barbara	baboon	airplane	Elaine	peppers	bridge	Boats	**average**
Huff.+DPCM (G)	0.4870	0.4941	0.4854	0.4900	0.4979	0.4778	0.4881	0.4856	0.4955	0.4808	**0.4882**
Huff. + DPCM (L)	0.3759	0.4131	0.3643	0.3954	0.4377	0.3207	0.3673	0.3607	0.4211	0.3383	**0.3795**
SOC	0.3395	0.4222	0.3883	0.3831	0.4768	0.3393	0.3601	0.3436	0.4702	0.3541	**0.3877**
Hu and Chang	0.3250	0.4043	0.3538	0.3721	0.4681	0.3084	0.3481	0.3243	0.4462	0.3188	**0.3618**
Enh. Hu and Chang (G)	0.3193	0.3977	0.3480	0.3664	0.4614	0.3075	0.3417	0.3196	0.4404	0.3157	**0.3567**
Enh. Hu and Chang (L)	0.3195	0.3912	0.3435	0.3661	0.4398	0.3008	0.3394	0.3197	0.4231	0.3120	**0.3669**
Rep. adj. (N)	0.3170	0.3944	0.3471	0.3673	0.4458	0.2990	0.3383	0.3197	0.4288	0.3095	**0.3555**
Rep. adj. (R)	0.3103	0.3907	0.3484	0.3645	0.4457	0.3038	0.3383	0.3159	0.4278	0.3140	**0.3559**
Rep. adj. (N+R)	0.3184	0.3954	0.3476	0.3683	0.4457	0.2987	0.3393	0.3195	0.4287	0.3096	**0.3571**
Remap+ Normal Indexing	0.5135	0.5143	0.5125	0.5133	0.5125	0.5117	0.5129	0.5149	0.5137	0.5120	**0.5131**
Remap+ DPCM +Huff. (L)	0.3489	0.4003	0.3454	0.3745	0.4247	0.3103	0.3330	0.3389	0.4135	0.3262	**0.3616**
Remap+DPCM +Huff. (G)	0.3420	0.3916	0.3382	0.3655	0.4294	0.3086	0.3292	0.3313	0.4099	0.3171	**0.3563**
Remap+ SOC	0.3463	0.4232	0.3910	0.3865	0.4720	0.3441	0.3659	0.3514	0.4666	0.3579	**0.3905**
Remap+ Hu and Chang	0.3237	0.4006	0.3514	0.3663	0.4568	0.3104	0.3423	0.3245	0.4403	0.3193	**0.3636**
Remap+Enh.Hu&Chang(L)	0.3188	0.3840	0.3318	0.3554	0.4289	0.2997	0.3234	0.3157	0.4151	0.3093	**0.3482**
Remap+Rep.adj.(N)	0.3231	0.3906	0.3396	0.3625	0.4367	0.3049	0.3289	0.3222	0.4239	0.3156	**0.3548**
Remap+Rep. adj.(R)	0.3157	0.3912	0.3429	0.3609	0.4364	0.3114	0.3274	0.3189	0.4236	0.3198	**0.3548**
Remap+Rep.adj.(N+R)	0.3284	0.3951	0.3446	0.3674	0.4400	0.3086	0.3339	0.3271	0.4280	0.3201	**0.3593**

Table (6.22): Comparing the bit rate (in bpp) of different index compression methods for codebook with size 512

Image	Lena	man	gold hill	Barbara	baboon	airplane	Elaine	peppers	bridge	Boats	**average**
Huff.+DPCM (G)	0.5526	0.5608	0.5518	0.5557	0.5619	0.5442	0.5546	0.5513	0.5601	0.5462	**0.5539**
Huff. + DPCM (L)	0.4553	0.4956	0.4482	0.4792	0.5205	0.3846	0.4547	0.4430	0.5039	0.4156	**0.4601**
SOC	0.3950	0.4943	0.4609	0.4492	0.5535	0.3870	0.4312	0.4071	0.5465	0.4127	**0.4537**
Hu and Chang	0.3907	0.4857	0.4376	0.4489	0.5622	0.3539	0.4355	0.3961	0.5368	0.3836	**0.4431**
Enh. Hu and Chang (G)	0.3815	0.4654	0.4208	0.4325	0.5346	0.3486	0.4176	0.3839	0.5120	0.3731	**0.4240**
Enh. Hu and Chang (L)	0.3817	0.4646	0.4210	0.4327	0.5159	0.3488	0.4178	0.3841	0.5004	0.3733	**0.4270**
Rep. adj. (N)	0.3782	0.4655	0.4267	0.4340	0.5218	0.3482	0.4167	0.3833	0.5085	0.3759	**0.4259**
Rep. adj. (R)	0.3774	0.4640	0.4275	0.4299	0.5220	0.3518	0.4150	0.3826	0.5075	0.3783	**0.4256**
Rep. adj. (N+R)	0.3867	0.4690	0.4317	0.4362	0.5269	0.3532	0.4197	0.3874	0.5131	0.3816	**0.4306**
Remap+ Normal Indexing	0.5263	0.5904	0.5242	0.5257	0.5866	0.5225	0.5250	0.5290	0.5889	0.5858	**0.5504**
Remap+ DPCM +Huff. (L)	0.4061	0.4701	0.4073	0.4343	0.4952	0.3623	0.3912	0.3972	0.4827	0.3838	**0.4230**
Remap+DPCM +Huff. (G)	0.3932	0.4549	0.3948	0.4223	0.4964	0.3506	0.3847	0.3863	0.4753	0.3665	**0.4125**
Remap+ SOC	0.4056	0.4955	0.4627	0.4536	0.5449	0.3938	0.4371	0.4194	0.5408	0.4177	**0.4571**
Remap+ Hu and Chang	0.3757	0.4712	0.4147	0.4307	0.5336	0.3522	0.3996	0.3822	0.5166	0.3747	**0.4251**
Remap+Enh.Hu&Chang(L)	0.3668	0.4510	0.3947	0.4148	0.4983	0.3452	0.3764	0.3702	0.4842	0.3664	**0.4068**
Remap+Rep.adj.(N)	0.3738	0.4606	0.4063	0.4262	0.5085	0.3507	0.3855	0.3805	0.4955	0.3760	**0.4164**
Remap+Rep. adj.(R)	0.3712	0.4595	0.4069	0.4256	0.5078	0.3547	0.3853	0.3787	0.4940	0.3783	**0.4162**
Remap+Rep.adj.(N+R)	0.3825	0.4673	0.4156	0.4345	0.5137	0.3578	0.3945	0.3888	0.5011	0.3842	**0.4240**

Table (6.23): Comparing the bit rate (in bpp) of different index compression methods for codebook with size 1024

Image	Lena	man	gold hill	Barbara	baboon	airplane	Elaine	peppers	bridge	Boats	**average**
Huff.+DPCM (G)	0.6207	0.6253	0.6194	0.6223	0.6265	0.6123	0.6223	0.6205	0.6249	0.6150	**0.6209**
Huff. + DPCM (L)	0.5745	0.6059	0.5643	0.5922	0.6301	0.4997	0.5724	0.5645	0.6138	0.5315	**0.5749**
SOC	0.4864	0.5840	0.5561	0.5342	0.6373	0.4673	0.5380	0.5079	0.6302	0.4915	**0.5433**
Hu and Chang	0.5007	0.5902	0.5472	0.5493	0.6621	0.4381	0.5610	0.5144	0.6378	0.4703	**0.5471**
Enh. Hu and Chang (G)	0.4818	0.5452	0.5181	0.5192	0.5990	0.4471	0.5227	0.4906	0.5828	0.4689	**0.5175**
Enh. Hu and Chang (L)	0.4743	0.5454	0.5162	0.5174	0.5991	0.4259	0.5228	0.4879	0.5829	0.4561	**0.5128**
Rep. adj. (N)	0.4813	0.5569	0.5344	0.5279	0.6129	0.4384	0.5352	0.4938	0.5957	0.4686	**0.5245**
Rep. adj. (R)	0.4835	0.5572	0.5362	0.5256	0.6148	0.4376	0.5364	0.4970	0.5969	0.4707	**0.5256**
Rep. adj. (N+R)	0.5003	0.5712	0.5552	0.5423	0.6254	0.4532	0.5512	0.5068	0.6077	0.4872	**0.5400**
Remap+ Normal Indexing	0.6141	0.6173	0.6103	0.6135	0.6098	0.6066	0.6119	0.6197	0.6147	0.6085	**0.6126**
Remap+ DPCM +Huff. (L)	0.4849	0.5515	0.4849	0.5107	0.5751	0.4369	0.4693	0.4764	0.5622	0.4657	**0.5018**
Remap+DPCM +Huff. (G)	0.4637	0.5209	0.4637	0.4879	0.5600	0.4210	0.4535	0.4592	0.5422	0.4404	**0.4812**
Remap+ SOC	0.5000	0.5856	0.5555	0.5402	0.6234	0.4756	0.5422	0.5242	0.6220	0.4987	**0.5468**
Remap+ Hu and Chang	0.4666	0.5645	0.5022	0.5169	0.6200	0.4279	0.4918	0.4790	0.6050	0.4556	**0.5130**
Remap+Enh.Hu&Chang(L)	0.4463	0.5298	0.4748	0.4929	0.5684	0.4175	0.4537	0.4512	0.5610	0.4427	**0.4838**
Remap+Rep.adj.(N)	0.4576	0.5412	0.4919	0.5057	0.5762	0.4282	0.4685	0.4668	0.5725	0.4532	**0.4962**
Remap+Rep. adj.(R)	0.4585	0.5420	0.4930	0.5043	0.5772	0.4300	0.4701	0.4677	0.5711	0.4548	**0.4969**
Remap+Rep.adj.(N+R)	0.4707	0.5493	0.5053	0.5160	0.5831	0.4393	0.4810	0.4796	0.5800	0.4638	**0.5068**

Method \ N	64	128	256	512	1024
Huff.+DPCM (G)	4.2934	3.5260	2.3570	1.5248	0.6530
Huff. + DPCM (L)	33.0958	29.2322	24.1083	18.2126	8.0166
SOC	26.8705	26.2046	22.4544	19.3354	13.0732
Hu and Chang	29.0226	30.5216	26.6206	21.2246	12.4625
Enh. Hu and Chang (G)	31.8948	32.7170	27.6439	24.0845	17.1931
Enh. Hu and Chang (L)	33.5550	33.4357	28.8978	24.6200	17.9499
Rep. adj. (N)	34.0370	33.8238	28.6638	24.2877	16.0779
Rep. adj. (R)	34.5357	34.1030	28.8111	24.3378	15.9063
Rep. adj. (N+R)	34.1609	33.8606	28.5798	23.4557	13.5933
Remap+ Normal Indexing	-1.0083	-1.5834	-2.6266	2.1450	1.9767
Remap+ DPCM +Huff. (L)	33.2859	30.6061	27.6871	24.7959	19.7188
Remap+DPCM +Huff. (G)	32.1896	30.8841	28.7439	26.6683	23.0001
Remap+ SOC	25.5808	24.5155	21.9013	18.7370	12.5190
Remap+ Hu and Chang	27.6085	29.3817	27.2899	24.4233	17.9274
Remap+Enh.Hu&Chang(L)	32.2284	32.8267	30.3564	27.6788	22.5900
Remap+Rep.adj.(N)	31.7045	31.8787	29.0391	25.9803	20.6124
Remap+Rep. adj.(R)	32.1169	32.1164	29.0332	26.0076	20.5022
Remap+Rep.adj.(N+R)	31.3196	31.1924	28.1378	24.6254	18.9092

Table (6.24): Comparing the improvement of bit rate over full index bit rate of different index compression algorithms for different codebook sizes

CHAPTER VII

Conclusion and Future Work

7.1. Conclusion

This thesis presents three new contributions for VQ index compression, namely, the enhanced Hu and Chang method, the repeated adjacencies method, and the remapping process. Simulations for the new methods are implemented and compared with the already found algorithms.

The enhanced Hu and Chang method is found to achieve lower bit rate than the original method of Hu and Chang. For codebook sizes 64, 128,256,512, and 1024, respectively, the average bit rates achieved using the enhanced Hu and Chang method are 0.2554, 0.2944, 0.3567, 0.4240, and 0.5175 (bpp) when using global prefixes and 0.2492, 0.2912, 0.3669, 0.4270, and 0.5128 (bpp) when using local prefixes, while the original Hu and Chang achieved 0.2662, 0.3040, 0.3618, 0.4431, 0.5471 (bpp).

The second proposed contribution is the repeated adjacencies method. When compared with Huffman coding, the Hu and Chang method, and SOC, the new method is found to achieve lower bit rates. Using "Next" table, "Right" table, or "Next"+"Right" tables achieve almost the same bit rates of 0.24, 0.28, 0.35, 0.42, and 0.52 (bpp) for codebook sizes of 64, 128, 256, 512, and 1024, respectively.

The third contribution is the remapping process. The remapping process is used in conjunction with different index compression methods. When using Huffman coding with remapped indices, a reduction in bit rate is achieved

over that achieved when using normal indices. The remapping process accomplish similarity between the local indices statistics and global statistics, so that the bit rates achieved with global Huffman codes for codebook sizes of 64, and 128, respectively, are 0.25, and 0.30 (bpp) and are almost equal to that achieved with local codes. For the higher codebook sizes the global codes achieve bit rates of 0.3563, 0.4125, and 0.4812 (bpp) which is a little lower than that obtained with the local codes due to the overhead cased by Huffman tables.

The SOC is not statistics based compression algorithm, but it depends on finding a matched index in the nearby area which is not much affected by the remapping process, this in turn does not improve the average bit rates when using normal indices or remapped indices.

The remapping process reduces the average differences between adjacent indices, and this improves the performance of the Hu and Chang, enhanced Hu and Chang, and repeated adjacencies method. The improvement in bit rate caused by the remapping process increases with increasing the codebook size, while it is canceled by the overhead at small codebook size of 64 and 128, it appears at larger codebook sizes of 256, 512, and 1024.

When comparing the different compression algorithms, it is found that the lower bit rates achieved for codebook of sizes 64 and 128 when using the repeated adjacencies method. The remapping with the enhanced Hu and Change gives the best result for codebook with sizes 256 and 512, while remapping with DPCM + global Huffman gives the best results for codebook size of 1024.

7.2. Future Work

Further work can be done to explore the performance of the proposed index compression algorithms with other VQ variants. Finite-state VQ, Adaptive VQ, and Subband VQ are good candidates because of the competitive bit rate that could be achieved.

The concept of repeated adjacent indices and search order coding can be used to implement fast encoding algorithms to allow more efficient VQ systems.

Applying VQ with index compression to video signals can achieve competitive bit rate while keeping simple algorithms which is a demand in low bit rate applications such as internet multimedia.

Hardware implementation of the proposed index compression algorithms using VLSI can be done and used in practical telecommunication systems.

References

1. A. Gersho and R. M. Gray, "Vector Quantization and Signal Compression," Kluwer Academic Publishers, 1992.

2. R. M. Gray, “Vector quantization,” IEEE Acoust., Speech, and Signal Process. mag., pp. 4–29, Apr. 1984.

3. N. M. Nasradbadi and R. A. King, “Image coding using vector quantization:A review,” IEEE Trans. Comm., vol. 39, pp. 957–971, Aug.1988

4. N. M. Nasrabadi and Y. Feng, “Image compression using address-vector quantization,” IEEE Trans. Comm., vol. 38, pp. 2166–2173, Dec. 1990.

5. Yu-Chen Hu and Chin-Chen Chang, “Low complexity index-compressed vector quantization for image compression”, IEEE Trans. on Consumer Electronics, vol. 45, No. 1 pp. 219–224, Feb. 1999.

6. C. H. Hsieh and J. C. Tsai, “Lossless compression of VQ index with search order coding”, IEEE Trans. on Image processing, vol. 5, No. 11, pp. 1579–1582, 1996.

7. M. H. Sheu, S.C. Tsai, and M. D. Shieh, “A lossless index coding algorithm and VLSI design for vector quantization,” The First IEEE Asia Pacific Conference on ASICs, 1999.

8. C. H. Hsieh, J. C Tsai, and P. C. Lu, “Noiseless Coding of VQ Index Using Index Grouping Algorithm,” IEEE Trans. On Comm., Vol. 44, No. 12, pp. 1643–1648, Dec. 1996.

9. X. Wu, W. Jiang and W. Wong, “Conditional Entropy Coding of VQ Indexes for Image Compression”, Proc. of 1998 IEEE Data Compression Conference, IEEE Computer Society Press, pp.347-356, March 1998.

10. Yun Gong, Michael K. H. Fan, “Image compression using lossless coding on VQ indexes” Proc. of IEEE Data Compression Conference, p.583, 2000.

11. K. T. Lo and J. Feng, “Predictive mean search algorithms for fast VQ encoding of images,” IEEE Trans. Consumer Electronics, vol. 41, No.2, pp. 327–331, 1995.

12. Y. Linde, A. Buzo, and R. M. Gray, “An algorithm for vector quantizer design,” IEEE Trans. Comm., vol. COM-28, pp. 84–95, Jan. 1980.

13. Equitz W. H.,"A new vector quantization clustering algorithm" IEEE Trans. on Acoustics, Speech and Signal Processing 37, 1568-1575. 1989.

14. A. Buzo, A. H. Gray, Jr. , R. M. Gray, and J. D. Markel, “Speech coding based upon vector quantization,” IEEE Trans. ASSP., vol. 28 , pp. 562–574, Oct. 1980.

15. B. Ramamurthi and A. Gersho, “Classified vector quantization of images,” IEEE Trans. Comm., vol. 34, pp. 1105–1115, Nov., 1986.

16. P. Fränti, "Digital Image Processing," University of Joensuu, Dept. of Computer Science. Lecture notes, 1998.

17. D. A. Huffman, "A method for the construction of minimum redundancy codes," Proc. IRE, Vol. 40, pp. 1098-1101, 1952.

18. G. K. Wallace, “The JPEG still picture compression standard," Comm. ACM, pp. 31-43, Apr. 1991.

19. http://www.acm.org/crossroads/xrds6-3/sahaimgcoding.html

20. http://www.debugmode.com/imagecmp/index.htm

21. http://www-isl.stanford.edu/~gray/compression.html

22. N. S. Jayant and P. Noll, "Digital Coding of Waveforms," Prentice-Hall, Englewwood Cliffs, New Jersey, 1984.

23. Ziv, J. & Lempel, A., "A universal algorithm for sequential data compression", IEEE Trans. on Information Theory, pp. 337-343, 1977.

24. Ziv, J. & Lempel, A., "A compression of individual sequences via variable-rate coding," IEEE Trans. on Information Theory, pp. 530-536, 1978.

25. W. B. Pennebaker, J. L. Mitchell, et. al., “Arithmetic coding articles;’ IBM J. Res. Develop., vol. 32, no. 6, pp. 717-774, Nov. 1988

26. http://www.netnam.vn/unescocourse/index.htm

27. Woods, J.,"Subband Image Coding", Kluwer Academic Publishers, 1991.

28. Rafael C. Gonzalez and Richard E. Woods, "Digital Image Processing," Addison-wesley Publishing Company, Inc., 1992.

29. M. Barnsley and L. Hurd," Fractal Image Compression," AK Peters, Wellesley, Mass., 1993.

30. Delp, E. & Mitchell, O., "Image compression using block truncation coding", IEEE Trans. on Comm., pp. 1335-1342,1997.

31. Guy Blelloch, draft of “algorithms in the real world” book, http://www_2.cs.cmu.edu/afs/cs/project/pscico_guyb/realworld/www/compression.pdf, 2001

32. Y. Yamada, K. Fujita, and S. Tazaki, " Vector Quantization of Video Signals", In proceeding of annual conference of IECE, pp. 1031, Japan, 1980.

33. Pan, J. S., McInnes, F. R., and Jack, M. A. "VQ codebook design using genetic algorithm." Electron. Lett., Vol. 31, No. 17, pp.18-19, 1995.

34. Delport, V., and Koschorreck, M. Genetic algorithm for codebook design in vector quatization. Electron. Lett., Vol. 31, No. 2, pp.84-85, 1995.

35. Pasi Franti,"Genetic algorithm with deterministic crossover for Vector Quantization," Pattern Recognition Letters 21, pp.61-68, 2000.

36. Juha Kivijarvi, Pasi Franti, Olli Nevalainen, "Self-Adaptive Genetic Algorithm for Clustering", Journal of Heuristics, 9, pp. 113–129, 2003.

37. J. Vaisey and A. Gersho. "Simulated Annealing and Codebook Design," Proceedings ICASSP, pp.1176-l 179, Apr. 1988.

38. Robert D. Dony, and Simon Haykin, "Neural Network Approaches to Image compression," Proc. Of the IEEE, Vol. 83, No. 2, Feb. 1995.

39. M. A. Robbers, and A. G. Katsaggelos, "Reducing blocking artifacts within Vector Quantization algorithms," IEEE Trans. Consumer Electronics, vol. 43, No.4 pp. 1118–1123, Nov. 1997.

References

40. P.C. Cosman, K. I. Oehler, E. A. Riskin, and R. M. Gray,"Using Vector Quantization for Image Processing," Proc IEEE, Vol. 81, No. 9, pp. 1326-1341, Sep. 1993.

41. A. Buzo, A. H. Gray, R. M. Gray, and J. D. Markel, "Speech coding based upon vector quantization," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-28, pp. 562–574, Oct. 1980.

42. K. Rose, E. Gurewitz, and G. C. Fox, "Vector quantization by deterministic annealing," IEEE Trans. Inform. Theory, vol. 38, July 1992.

43. Zeger, K., Gersho, A., "Stochastic relaxation algorithm for improved vector quantiser design". Electronics Letters 25, pp. 896-898, 1989.

44. C. W. Chao, C. C. Chiu, P. C. Lu, and C. H. Hsieh, "Codebook design for vector quantization of images based on fuzzy c-means clustering algorithm," Opt. Eng., vol. 36, pp. 580–587, Feb. 1997.

45. Byung Cheol Song and Jong Beom Ra," a fast search algorithm for Vector Quantization using L2 norm pyramid of code words", IEEE Trans. On Image processing, Vol. 11, No. 1, Jan. 2002.

46. James McNames, "Rotated partial distance search for fast VQ encoding ", IEEE signal processing letters, Vol. 7, No. 9, Sep. 2000.

47. J. S. Pan, Z. M. Lu, ands. H. Sun, "Fast codeword search algorithm for image coding based on mean-variance pyramids of code words", Electronics letters, Vol. 36, No. 3, Feb. 2000.

48. Wenhua Li, Ezzatollah Salari, "A fast Vector Quantization encoding method for image compression", IEEE Trans. on circuits and systems for video technology, Vol. 5, No. 2, Apr. 1995.

49. Chang-Hsing Lee, Ling-Hwei Chen, "A fast search algorithm for Vector Quantization using mean pyramids of code words", IEEE Trans. on comm. Vol. 43, No. 2/3/4, Feb./Mar./Apr. 1995.

50. S. W. Ra, J. K. Kim, "Fast mean distance ordered partial codebook search algorithm for image Vector Quantization", IEEE Trans. on circuits on circuits and systems-II analog and digital signal processing, Vol. 40, No. 9, Sep. 1993.

References

51. K. K. Paliwal and V. Ramasubramanian, "Effect of ordering the code book on the efficiency of the partial distance search algorithm for Vector Quantization", IEEE Trans. on comm., Vol. 37, No. 5, May 1989.

52. C. H. Lee and L. H. Chen, “A fast vector search algorithm for vector quantization using mean pyramids of codevectors,” IEEE Trans. Comm., vol. 43, pp. 1697–1702, Feb. 1995.

53. Averbuch, A., Lazar, D., and Israeli, M. "Image compression using wavelet transform and multireslution decomposition." Trans. On IEEE Image Proc., vol. 5, pp. 4-15, 1996.

54. Pamela C. Cosman, Robert M. Gray, Martin Vetterli ,"Vector Quantization of image subbands : A survey", IEEE Trans. on Image processing, Vol. 5, No. 2, Feb. 1996.

55. J. Foster, R. M. Gray, and M. O. Dunham, “Finite-state vector quantization for waveform coding,” IEEE Trans. Inform. Theory, vol. IT-31, pp. 348–359, May 1985.

56. A. Gersho, “Adaptive vector quantization,” Ann. TeleComm., vol. 41, pp. 470–480, Sep. 1986.

57. M. Goldberg, P. R. Boucher, and S. Shlien, "Image compression using adaptive vector quantization," IEEE Trans. Comm. 34, 180–187, 1986.

LIST OF PUBLISHED PAPERS

1. M. Farouk, Tarik K. Abdel-Hamid, H. Selim, and Magdy M. Doss," A New VQ Index Remapping Process Based on Index Usage Activity for Image Compression", submitted for publication in Journal of Engineering Sciences, JES, Assiut University, Jan. 2005.

2. M. Farouk, Tarik K. Abdel-Hamid, H. Selim, and Magdy M. Doss," Utilizing Repeated Adjacencies of Vector Quantization Indices in Image Compression", submitted for publication in the proc. Of IEEE symp. On signal processing and informayion theory, ISSPIT2004, Rome, Italy, Dec. 2004.

3. M. Farouk, Tarik K. Abdel-Hamid, H. Selim, and Magdy M. Doss," Utilizing Index Usage Maps for VQ Index Compression", submitted for publication in the proc. Of IEEE symp. On signal processing and informayion theory, ISSPIT2004, Rome, Italy, Dec. 2004.

الملخص العربى

فى الأنظمه العمليه للتكويد الكمى بالمتجهات المستخدمه لإشارة الصوره يتم تقسيم الصوره إلى أجزاء صغيره لتقليل زمن العمليات الحسابيه. هذا بالتالى يؤدى إلى أجزاء شديدة الترابط وبالتالى سترث أدلة التكويد الكمى بالمتجهات هذا الترابط حيث يمكن إستغلاله لزيادة نسبة الضغط لعملية التكويد الكمى بالمتجهات عن طريق إضافة مرحلة ضغط للأدله بعد عملية التكويد الكمى بالمتجهات . خوارزميات ضغط الأدله تعمل فى نطاق الأدله وبالتالى يكون لها خوارزميات أبسط من تلك الخاصه بالتكويد الكمى بالمتجهات ذو الذاكره.

قدم هذا البحث ثلاث مساهمات فى مجال ضغط أدلة التكويد الكمى بالمتجهات لزيادة نسبة الضغط للتكويد الكمى بالمتجات الأساسى لضغط إشارة الصوره. المساهمه الأولى هى تعديل فى طريقه هوو و شانج. فى الطريقه الأصليه يتم تصنيف العلاقه بين الدليل الحالى وكلا من الدليل الأعلى منه والدليل على يساره إلى أربع حالات. فى الطريقه المعدله تضاف حاله خامسه لتطبق فى حالة كون الفرق بين الدليل الحالى و الدليل على اليسار أقل من قيمه محدده مسبقا. تستخدم أكواد ذات أطوال متغيره كمحددات للخمس حالات المختلفه.

المساهمه الثانيه هى خوارزميه جديده لضغط الأدله حيث تستخدم الإحصاءات الخاصه بكل صوره والنسق المتكرر للأدله المتجاوره. ينشأ لكل صوره جدول "التالى" أو "اليمين" وهوعباره عن مصفوفه بأبعاد 1XN ؛حيث ان N هو حجم كتاب الشفره؛ تحتفظ لكل دليل بمؤشر للدليل الأكثر إحتماليه أن يأتى بعده (أو يمينه). هذا الجدول يستخدم لتقليل وحدات البيانات اللازمه لتمثيل الأدله اذا وجد نسق متكرر.

المساهمه الثالثه هى طريقه لإعادة ترقيم أدله التكويد الكمى بالمتجهات لزيادة نسبة الضغط لطرق ضغط الأدله المختلفه وتعتمد هذه الطريقه على توزيع الأدله طبقا لمكان إستخدامها لتقليل العدد الكلى للأدله حيث أن العدد الكلى للأدله فى مختلف أجزاء الصوره يكون محدود وتوظف هذه العمليه لترقيم الأدله المستخدمه وإهمال الأدله الغير مستخدمه وقد تم تطبيق طرق ضغط الأدله على الأدله المعاد ترقيمها وتم دراسه تأثير إعادة الترقيم فوجد أن عملية إعادة الترقيم تزيد من نسبة ضغط الأدله لطرق ضغط الأدله المختلفه .

جامعة أسيوط
كلية الهندسة

ضغط إشارة الصوره بالتكويد الكمى بالمتجهات

رسالة ماجستير مقدمة من

المهندس / محمد فاروق عبداللاطيف أحمد

قسم الهندسة الكهربائية
كلية الهندسة - جامعة أسيوط

لجنة الإشراف:

أ.د / هانى سليم جرجس
(كلية الهندسة – جامعة أسيوط)

أ.د / مجدى مفيد دوس
(كلية الهندسة – جامعة أسيوط)

د / طارق كمال عبد الحميد
(كلية الهندسة – جامعة أسيوط)

لجنة المناقشة:

أ.د / عبد الفتاح إبراهيم عبدالفتاح
(كلية الهندسة – جامعة المنصوره)

أ.د / محمد أبوزهاد أبوزيد
(كلية الهندسة – جامعة أسيوط)

أ.د / هانى سليم جرجس

أ.د / مجدى مفيد دوس

جامعة أسيوط
كلية الهندسة

ضغط إشارة الصوره بالتكويد الكمى بالمتجهات

رسالة ماجستير مقدمة من

المهندس / محمد فاروق عبداللاطيف أحمد

قسم الهندسة الكهربائية
كلية الهندسة - جامعة أسيوط

October 2004