جـامـعـة الــشـرق الأوسـط
MIDDLE EAST UNIVERSITY

# Developing a High-Performance Adjustable-Quality Data Compression Scheme for Multimedia Messaging

**تطوير نظام ضغط بيانات الرسائل متعددة الوسائط  بأداء عالي الجودة والتقنية**

By

**Tamer Waleed Alsous**

Supervisor

**Dr. Hussein Al-Bahadili**

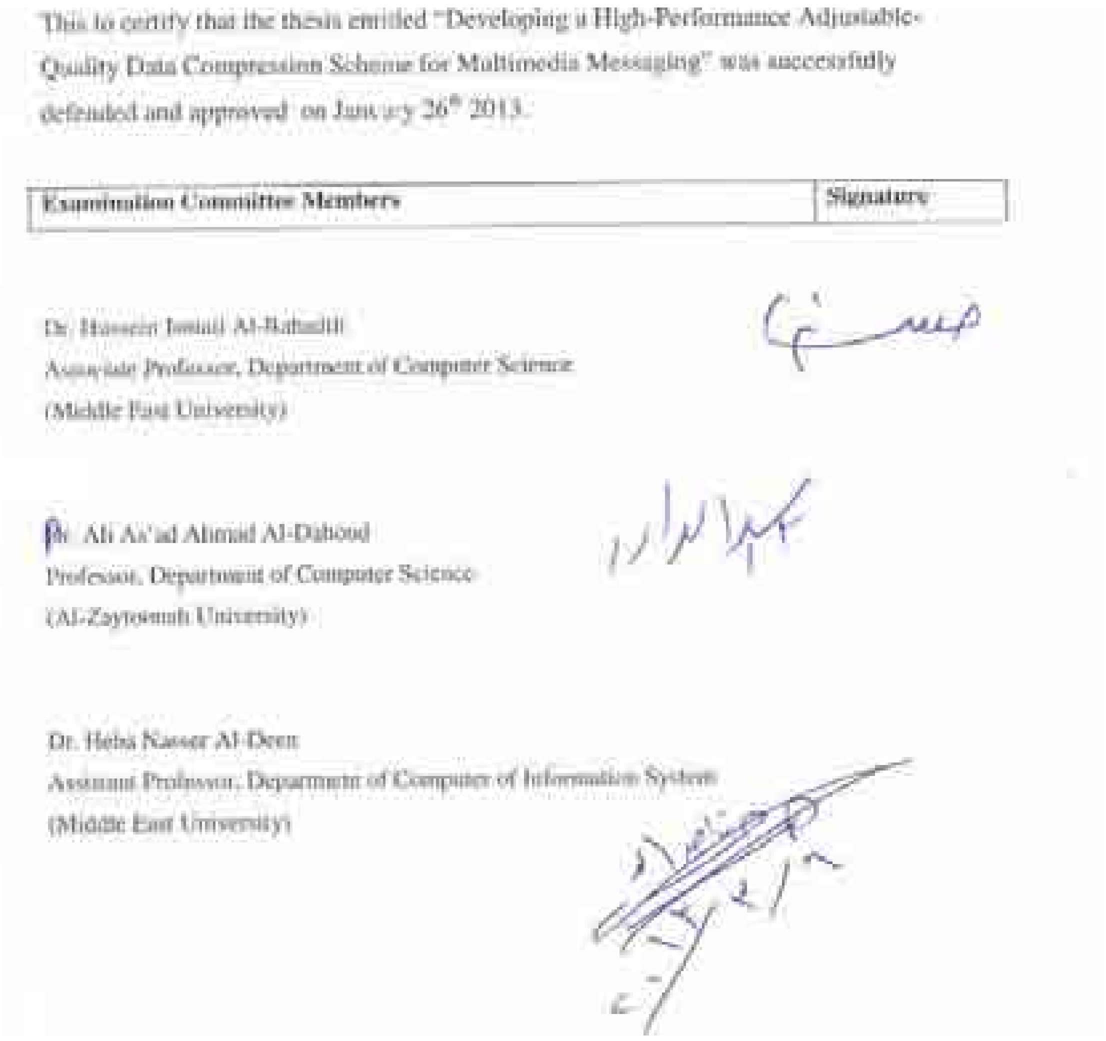
This thesis is submitted to the Department of Computer Science, Graduate College of Computing Studies, Middle East University in partial fulfillment for the requirement for the degree of Master of Science in Computer Science.

**Faculty of Information Technology**

**January, 2013**

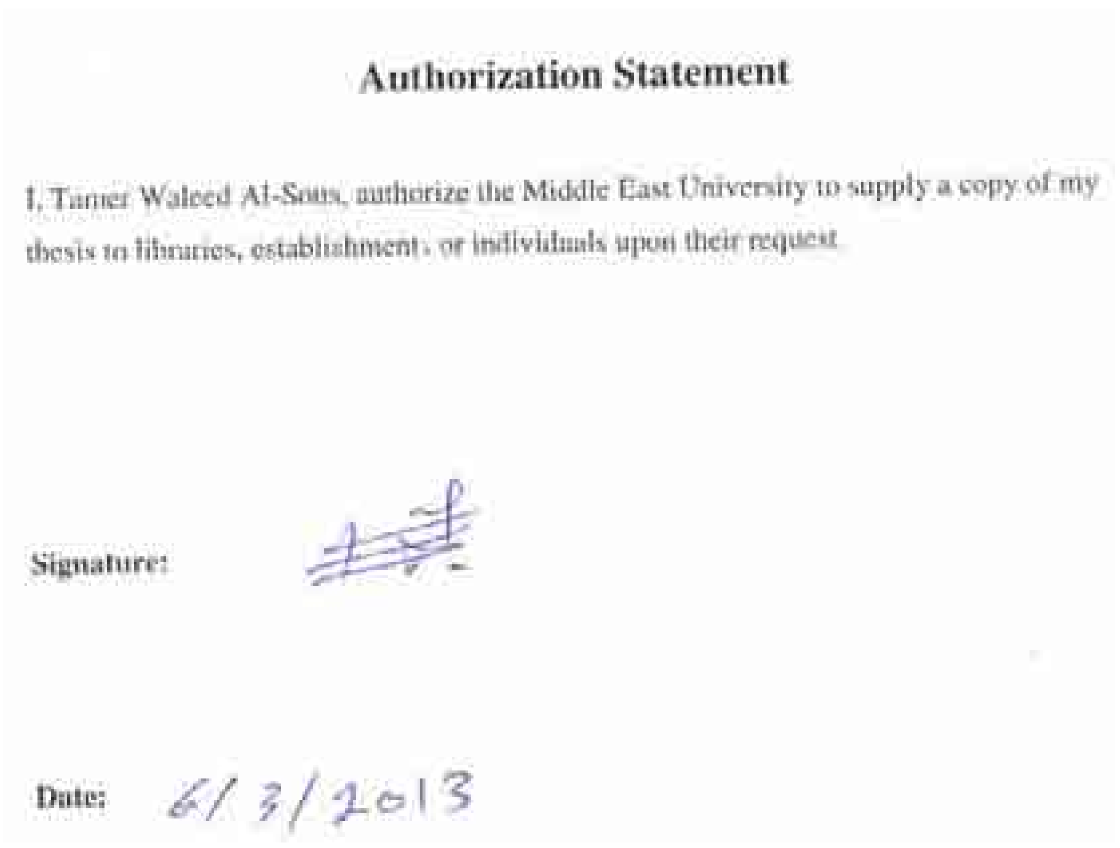
**Middle East University**

## Examination Committee Decision



This to certify that the thesis entitled "Developing a High-Performance Adjustable-Quality Data Compression Scheme for Multimedia Messaging" was successfully defended and approved on January 26th 2013.

| Examination Committee Members | Signature |
|---|---|
| Dr. Hussein Ismail Al-Bahadili<br>Associate Professor, Department of Computer Science<br>(Middle East University) | [illegible] |
| Dr. Ali As'ad Ahmad Al-Dahoud<br>Professor, Department of Computer Science<br>(Al-Zaytoonah University) | [illegible] |
| Dr. Heba Nasser Al-Deen<br>Assistant Professor, Department of Computer of Information System<br>(Middle East University) | [illegible] |



## Authorization Statement

I, Tamer Waleed Al-Sous, authorize the Middle East University to supply a copy of my thesis to libraries, establishment, or individuals upon their request.

**Signature:** [illegible]

**Date:** 6/3/2013

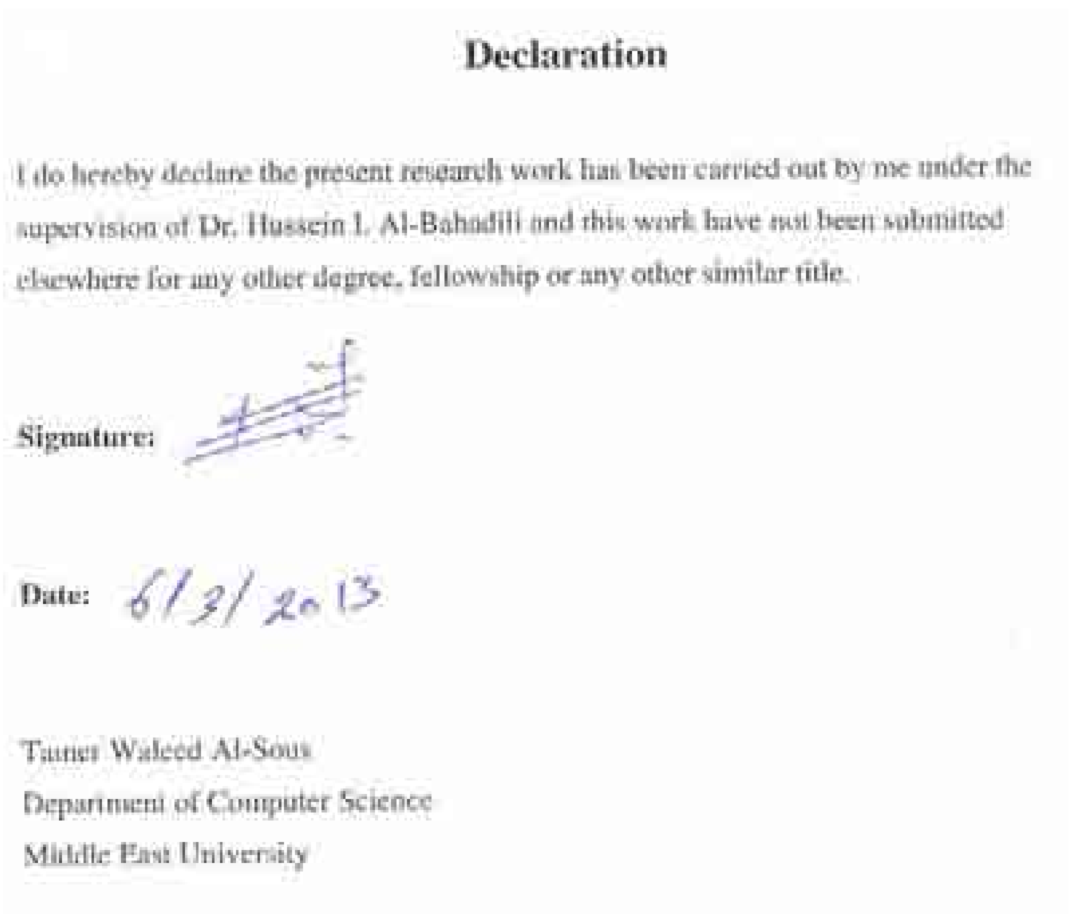
## Authorization Form

## إقرار تفويض

أنا ثامر وليد الصوص أفوض جامعة الشرق الأوسط بتزويد نسخ من رسالتي للمكتبات أو المؤسسات أو الهيئات أو الأفراد عند طلبها.

التوقيع: [illegible]

التاريخ: ٦ / ٢ / ٢٠١٢



## Declaration

I do hereby declare the present research work has been carried out by me under the supervision of Dr. Hussein I. Al-Bahadili and this work have not been submitted elsewhere for any other degree, fellowship or any other similar title.

**Signature:** [illegible]

**Date:** 6/3/2013

Tamer Waleed Al-Sous
Department of Computer Science
Middle East University

# Dedication

This thesis is dedicated to my father, who taught me that the best kind of knowledge to have is that which is learned for its own sake. It is also dedicated to my mother, who taught me that even the largest task can be accomplished if it is done one step at a time. To my brothers and lovely sister, and to all friends who they supported me till finish this work. I lovingly dedicate this thesis to my wife, who supported me each step of the way.

## Acknowledgment

I would like to specially thank my supervisor Dr. Hussein Al-Bahadili, who taught me everything that I know about research and the way it should be done. I would like to thank him for his guidance during all stages of this research, for answering endless questions, for his great support, professional advice, and profound understanding. Through this work, Dr. Hussein has shown me how to attack a problem from different angels, how to approach it, and how to find the most suitable solution.

I also would like to thank all members of staff at Middle East University, in particular, the members of staff at the Graduate College of Computing Studies.

Finally, it would be unthinkable of me not to thank my parents, wife, children, brothers and friends for their support and encouragement over the years. I am thankful for anyone who supported me during my master study.

# Table of Contents

# List of Tables

# List of Figures

## Abbreviations

| | |
|---|---|
| ACW | Adaptive Character Wordlength. |
| AQ-HCDC | Adjustable-Quality Hamming Codes Data Compression. |
| ASCII | American Standard Code for Information Interchange. |
| BMP | Bitmap. |
| Bpc | bit per character. |
| BWT | Binary Wavelet Transform. |
| CBWTC | Context-based Binary Wavelet Transform Coding. |
| CMYK | Cyan, Magenta, Yellow and Key. |
| dB | decibel. |
| DCT | Discrete Cosine Transform. |
| DIB | Device Independent Bitmap. |
| DMT | Discrete Modal Transform. |
| DPCM | Differential Pulse-Code Modulation. |
| DVM | Directional Vanishing Moments. |
| EXIF | Exchangeable Image File Format. |
| FRAR | Full Range Auto Regressive. |
| GCR | Gray Code Representation. |

| | |
|---|---|
| GIF | Graphics Interchange Format. |
| GIS | Geographic Information Systems. |
| HCDC | Hamming Codes Data Compression. |
| HDTV | High Definition Television. |
| ICT | Information and Communication Technology. |
| ID-CCT | Image Dependent Color space Transform. |
| IEEE | Institute of Electrical and Electronics Engineers. |
| IESG | Internet Engineering Steering Group. |
| JBIG2 | Joint Bi-level Image Experts Group |
| JFIF | JPEG File Interchange Format. |
| JPEG | Joint Photographic Experts Group. |
| LAN | Local Area Network. |
| LSB | Least Significant Bit. |
| LZ | Lempel Ziv. |
| LZW | Lempel Ziv Welch. |
| MIME | Multipurpose Internet Mail Extensions. |
| MMS | Multimedia Messaging Service. |
| MSE | Mean Square Error. |
| MSRCR | Multi-Scale Retinex with Color Restoration. |

| | |
|---|---|
| OBDD | Ordered Binary-Decision Diagram. |
| OS | Operating System. |
| PAN | Personal Area Networks. |
| PCDC | Polynomial Codes Data Compression. |
| PCM | Pulse Code Modulation. |
| PCX | Personal Computer Exchange. |
| PDA | Personal Digital Assistant. |
| PNG | Portable Network Graphics. |
| PSNR | Peak Signal to Noise Ratio. |
| QoSs | Quality-of-Services. |
| RGB | Red Green Blue. |
| RLC | Run Length Coding. |
| RLE | Run Length Encoding. |
| RWT | Real Wavelet Transform. |
| SFQ | Space-Frequency Quantization. |
| SFX | SelF-eXtracting archive. |
| SMS | Short Message Service. |
| SPIHT | Set Partitioning In Hierarchical Trees |
| TIFF | Tagged Image File Format. |

| | |
|---|---|
| TVQ | Transformed Vector Quantization. |
| USB | Universal Serial Bus. |
| WAN | Wide Area Network. |
| WT | Wavelet Transform. |

# English Abstract

Data compression algorithms are designed to reduce the size of data so that it requires less disk space for storage and less bandwidth to be transmitted over data communication channels. An additional benefit of data compression is in wireless communications devices, where it reduces the transmission power consumption of the device as the power consumption is directly proportional to the size of the transmitted data. Data compression can play a big role in improving the Quality-of-Service (QoS) in Multimedia Messaging Service (MMS) on mobile phones, where MMS is facing three main challenging: limited message size, high power consumption, and long delay.

In this thesis, we develop a new data compression algorithm for stand still image compression, which can be used for MMS and Internet applications on mobile networks. The new algorithm is a modified version of the lossless bit-level Hamming Codes based Data Compression (HCDC) algorithm, and it is characterized as an image-based adjustable-quality algorithm; therefore, it is referred to as Adjustable-Quality HCDC (AQ-HCDC) algorithm.

The AQ-HCDC algorithm is implemented using VB.NET, and to evaluate the performance of the algorithm, two types of experiments are carried-out for large-size and small-size images compression. The images used in these experiments are widely-used as a test images by many researchers due to their standard features. In particular, six images are selected, namely, AirPlane, Baboon, CornField, Flowers, Girl, and Monarch.

The performance of the AQ-HCDC algorithm is evaluated in terms of Compression ratio (*C*), Mean Square Error (MSE), and Peak Signal to Noise Ratio (PSNR). In all experiments, the performance of the AQ-HCDC algorithm is compared against the performance of a number of lossless compressed image formats (e.g., GIF and PNG), lossy image format (e.g., JPEG), and lossless compression tools (e.g., ZIP and WinRAR).

For both experiments the AQ-HCDC algorithm provides a compression ratio of ≈1.6 and a PNSR of more than 30 dB, which ranks the algorithm between the most widely-used formats PNG and JPEG.

## Arabic Abstract

## الخلاصة

صممت خوارزميات ضغت البيانات للتقليل من جحم البيانات وبالتالي تقليل المساحة المطلوبة لتخزين البيانات وللتقليل أيضا من عرض النطاق الترددي(bandwidth) المطلوب لنقلها خلال قنوات الاتصال. ويمكن الاستفادة من ضغط البيانات خلال اجهزة الاتصال اللاسلكية، حيث تحتاج طاقة اقل لنقلها من خلال قنوات الاتصال. كما يمكن لضغط البيانات ان تلعب دورا مهما في تحسين جودة خدمة الرسائل المتعددة الوسائط في الاجهزة النقالة، حيث هناك ثلاث تحديات رئيسية تواجه خدمة الرسائل المتعددة الوسائط: محدودية حجم الرسالة، الاستهلاك العالي للطاقة، والتأخر في وصول الرسالة.

في هذه الاطروحة، سوف نقوم بتطوير خوارزمية جديدة لضغط الصور يمكن تطبيقها على الرسائل متعددة الوسائط وتطبيقات الانترنت على شبكات الاجهزة النقالة. الخوارزمية الجديدة هي نسخة معدلة من خوارزمية the lossless bit-level Hamming Codes based Data Compression (HCDC) وتتميز بقابليتها على تعديل جودة الصورة، لذلك تم تسميتها Adjustable-Quality HCDC (AQ-HCDC).

تم تنفيذ الخورازمية بإستخدام لغة VB.net ولتقييم ادائها تمت تجربتها على نوعين من الصور: كبيرة الحجم وصغيرة الحجم. وهذه الصور التي تم تطبيق الخوارزمية عليها جرى استخدامها من قبل العديد من الباحثين حول العالم بناءً على المميزات التي تتمتع بها. وعلى وجه الخصوص تم إختيار ستة صور، وهي:,AirPlane, Baboon CornField, Flowers, Girl, Monarch.

أداء الخوارزمية تم تقييمه من خلال قياس نسبة الضغط Compression ratio (C)، متوسط الخطأ التربيعي Mean Square Error (MSE)، وإشارة الذروة إلى نسبة الضوضاء Peak Signal to Noise Ratio (PSNR).
في كل التجارب تم مقارنة أداء الخوارزمية مع عدد من خوارزميات ضغط الصور المعروفة جيدا مثل:,JPEG, PNG GIF وأيضا مع عدد من الادوات المستخدمة في ضغط البيانات مثل:ZIP and WinRAR .

في كلا التجربتين استطاعت الخوارزمية الوصل الى معدل ضغط يساوي 1.6 تقريبا، أما مقياس إشارة الذروة إلى نسبة الضوضاء فتم تحقيق أكثر من 30 ديسيبيل، وذلك يضع الخوارزمية بين صفوف الخوارزميات الاشهر JPEG, PNG.

# Chapter One

# Introduction

## 1.1 Overview

The last two decades have witnessed a tremendous revolution and growth in the field of Information and Communication Technology (ICT), and still more and more to emerge. The revolution has been notably recorded in a number of related fields of ICT; these may include:

(1) Computer technology: Development of very high performance computing resources, especially those of mobile characteristics, such as: laptops, netbooks, notebooks, smart phones (e.g., IPhones), Personal Digital Assistants (PDAs), IPads, etc. (Patterson & Hennessy, 2011).

(2) Communications technology: Emergent of various wire/wireless networks designs, such as: Local Area Networks (LANs), Wide Area Network (WAN), Personal Area Networks (PAN), wireless networks, cellular networks, satellite communications (Forouzan, 2007) (Stallings, 2010).

(3) Internet technology: Development of masses of applications and tools (e.g., World Wide Web (Web)) open the door towards an amazing world of all kinds of information and knowledge exchange. Internet facilities are easily accessible in almost every parts of the world (Duffy, 2012) (Bentley & Barrett, 2012).

(4) Multimedia technology: Production of very high resolution and high quality multimedia devices, such as: digital cameras, display screens, audio systems, etc. (Ze-Nian Li & Mark, 2004).

(5) Mobile technology: Emergent of high performance, small-size, low-power, and mobile devices (e.g., laptops, notebooks, smart phones, etc.), which have led to the emergent of a wide range of tools and applications (Duffy, 2012) (Bentley & Barrett, 2012).

In addition to its many advanced characteristics (e.g., small size, high processing speed, relatively large memories, low power consumption, long life batteries), digital cameras nowadays can produce very high quality or high resolution images. The term resolution is often used for a pixel count in digital imaging, which gives an indication on the digital image quality. The higher the resolution, the better the image is. A low resolution digital image (640x480 pixels), for instance, may look acceptable for displaying on the Internet but it can appear fuzzy when printed or enlarged. By comparison, high resolution images, such as those at 1280x1024 pixels, contain enough pictorial information (sharp contrasts, rich colors, and picture details) to look good on the internet as well as when printed or enlarged.

Storing high resolution images require high storage capacities (memories); for example, a 1280x1024 image size using RGB color requires 3.894 MB space. Fortunately, modern digital cameras are supplied with reasonably huge memories of Giga Bytes (GB) that can store large number of high resolution images or videos. Digital cameras are also equipped with wire and wireless interface technologies. Wire interface technologies like the Universal Serial Bus (USB), and wireless interface technologies like Bluetooth (IEEE 802.15 protocol) and Wireless LAN (WLAN) (IEEE 802.11 protocol) (Forouzan, 2007) (Stallings, 2010). Thus, digital cameras memories can be cleared by copying images' data into more permanent or external storage media.

Most modern mobile phones are equipped with high technology digital camera(s), encouraging users taking more and more pictures or videos while they are on the move; and despite the high storage capacity of modern mobile phones, the memory may not be enough to accommodating all users' generated data, therefore, users may need to clear or flash-out their mobile's memory to a remote end. The most obvious way to exchange these pictures or videos is through the available cellular bandwidth.

In this direction, researchers developed a standard service to send messages that include multimedia content to and from mobile phones, namely, the Multimedia Messaging Service (MMS). It extends the core Short Message Service (SMS) capability that allowed exchange of text messages only up to 160 characters in length. Comparing the current cellular bandwidth cannot meet the growing demands for satisfactory performance for multimedia exchange through cellular systems, and more powerful algorithms or protocols are required.

MMS can be divided into two main interrelated tasks; these are: data processing and data communication. Data processing may include the following main activities: data capturing, encoding, compression, decompression, decoding, storing, and displaying. Data communication may include the following main activities: data packetization, routing, error-detection, and data re-assembling.

One of the influential factors on the performance of MMS is the maximum size of the message that can be exchanged. Although the standard does not specify a maximum size for a message, 300 KB is the current recommended size used by networks due to some limitations on the available bandwidth. It at the end depends on the performance of the local cellular infrastructure. Furthermore, exchanging large amount of data facing two more challenges, these are high power consumption and long delay. One obvious approach to address the concerns of MMS is to compress the exchange data using powerful reversible data compression algorithms, while maintaining image quality. Which is the main concept discussed in this thesis.

## 1.2 Data Compression

Data compression aims to reduce the size of data so that it requires less disk space for storage and less bandwidth to be transmitted over data communication channels (Sayood, 2012) (Salomon, 2007). Data compression also reduces the amount of accumulated errors during data transmission over error-prone data communication channels by decreasing the size of information to be exchanged over such channels (Adiego, Navarro, & de la Fuente, 2007) (Freschi & Bogliolo, 2004). An additional benefit of data compression is in wireless communication devices where it may introduce a significant power saving. Power savings is possible by compressing data prior to transmission, where the power consumed is directly proportional to the size of the transmitted data. In fact, in wireless devices, transmission of a single bit requires over $10^3$ times more power than a single 32-bit processing.

Data compression is usually obtained by substituting a shorter symbol for an original symbol in the source data, containing the same information but with a smaller representation in length. The symbols may be characters, words, phrases, or any other unit that may be stored in a dictionary of symbols and processed by a computing system (Witten I. H., 2004). The data compression process requires efficient algorithmic transformations of a source message to produce representations (also called codewords) that are more compact. Such algorithms are known as data compression algorithms or data encoding algorithms. On the other hand, the data compression process must be reversible process, such that each data compression algorithm needs to be complemented by its inverse, which is known as a data decompression algorithm (or data decoding algorithm), to restore an exact or an approximate form of the original data (Al-Bahadili & Hussain, 2008) (Gryder & Hake, 1991).

### 1.2.1 Data compression models

Different data compression models have been recommended and used throughout the years. These data compression models can be classified into four major models; these are (Sayood, 2012):

1) Substitution data compression model
2) Statistical data compression model
3) Dictionary-based data compression model
4) Bit-level data compression model

A substitution model involves the swapping of repeating characters by a shorter representation. Algorithms that are based on this model include: Null Suppression, Run-Length Encoding (RLE), Bit Mapping and Half Byte Packing (Pandya, 2000).

A statistical model involves the generation of the shortest average code length based on an estimated probability of the characters. Examples of algorithms that are based on this model include: Shannon-Fano coding (Rueda & Oommen, 2006), static/dynamic Huffman coding (Huffman, 1952) (Knuth, 1985) (Vitter, 1989), and arithmetic coding (Howard & Vitter, 1994) (Witten & Neal, 1987).

A dictionary-based model involves the substitution of substrings by indices or pointer code, relative to a dictionary of the substrings; algorithms that can be classified as a dictionary-based model include the LZ compression technique and its variations (Ziv & Lempel, 1977) (Ziv & Lempel, 1978) (Nelson, 1989) (Brittain & El-Sakka, 2007).

Finally, since data files could be represented in binary digits, a bit-level processing can be performed to reduce the size of data. In bit-level data compression algorithms, the binary sequence is usually divided into groups of bits that are called minterms, blocks, subsequences, etc. These minterms might be considered as representing a Boolean function. Then, algebraic simplifications are performed on these Boolean functions to reduce the size or the number of minterms, and hence, the number of bits representing the output (compressed) data is reduced as well.

Examples of bit-level data compression algorithms include: The Polynomial Codes Data Compression (PCDC) (Sirbu & Cleju, 2011), the Hamming Codes based Data Compression (HCDC) algorithm (Al-Bahadili H. , 2008) (Al-Bahadili & Rababa'a, 2010), the Adaptive Character Wordlength (ACW(*n*)) algorithm (Al-Bahadili & Hussain, 2008), the Adaptive Character Wordlength (ACW(*n,s*)) scheme (Al-Bahadili & Hussain, 2010), the logic-function simplification algorithm (Nofal, 2007), the neural network based algorithm (Mahoney, 2000).

### 1.2.2 Categorization of data compression algorithms

There are many data compression algorithms that have been developed throughout the years utilizing the different models discussed in previous section and are designed for various applications and purposes. Therefore, it is necessary to have some methodologies to classify these algorithms in a way that helps academicians, researchers, developers and professionals to understand these algorithms. In this direction, data compression algorithms are categorized by several characteristics, such as (Sayood, 2012) (Salomon, 2004) (Salomon, 2002):

(1) Data compression fidelity

(2) Length of data compression symbols

(3) Data compression symbol table

(4) Data compression cost

In what follows, I shall provide a brief explanation for each of them:

**(1) Data compression fidelity**

One of the most important characteristics is the fidelity with which the original and the decompressed data agree with each other. The decompressed (restored) data can either represent an exact or an approximate form of the original data set (Shapira & Daptardar, 2006). Therefore, two fundamentally different styles of data compression can be recognized, depending on the fidelity of the restored data, these are: lossless and lossy data compression.

- *Lossless data compression*

It involves a transformation of the representation of the original data set such that it is possible to reproduce exactly the original data (exact copy). Lossless compression is used in compressing text files, executable codes, word processing files, database files, tabulation files, and whenever it is important that the original and the decompressed files must be identical. Lossless compression is used in many applications, e.g., the popular ZIP file format and in the UNIX tool gzip. It is also used as a component within lossy data compression technologies. Lossless compression algorithms can usually achieve a 2 to 8 compression ratio (Rueda & Oommen, 2006) (Brittain & El-Sakka, 2007).

- *Lossy data compression*

It involves a transformation of the representation of the original data set such that it is impossible to reproduce exactly the original data set, but an approximate representation is reproduced by performing a decompression transformation. This type of compression is used frequently on the Internet and especially in streaming media and telephony applications. Because some information is discarded, it achieves better data compression ratios that reach 100 to 200, depending on the type of information being compressed. In addition, higher compression ratio can be achieved if more errors are allowed to be introduced into the original data (Brittain & El-Sakka, 2007) (Witten I. H., 2004).

### (2) Length of data compression symbols

Data compression algorithms are characterized by the length of the symbols an algorithm process, regardless of whether the algorithm uses variable length symbols in the original data or in the compressed data, or both. For example, RLE uses fixed length symbols in both the original and the compressed data. Huffman encoding uses variable length compressed symbols to represent fixed-length original symbols. Other methods compress variable-length original symbols into fixed-length or variable-length encoded data.

### (3) Data compression symbol tables

Another distinguishing feature of the compression algorithms is the source of the symbol table. According to this feature, compression algorithms can be classified into: static or fixed, dynamic or adaptive, and semi-adaptive compression algorithms

- *Static or fixed data compression algorithms*

Some data compression algorithms operate on a static symbol table, or a fixed dictionary of compression symbols. Because the dictionary is fixed, it needs not be combined with the compressed data. Such algorithms are dependent on the format and content of the data to be compressed. However, a fixed dictionary is usually optimized for a particular data type,   whereas if the same dictionary used for other types of information the efficiency of the algorithm suffers and provides a lower compression ratio.

- *Dynamic or adaptive data compression algorithms*

Some data compression algorithms are relatively independent, and some make two passes at the data. The first pass determines the frequency of the symbols that will be processed; and builds a symbol table based on that frequency. The custom symbol table needs to combine the compressed data, and the second pass uses the custom symbol table to encode and decode data. Adaptive compression algorithms build a custom symbol table as they compress the data. Such algorithms encode each character based on the frequency of preceding characters in the original data file. The decompression algorithm builds an identical dynamic table as the information is decompressed. Adaptive methods usually start with a minimal symbol table to bias the compression algorithm toward the type of data they are expecting.

- *Semi-adaptive data compression algorithms*

In a semi-adaptive algorithm the data to be compressed are first analyzed in their entirety, an appropriate model is then built, afterwards the data is encoded. The model is stored as part of the encoded data, as it is required by the decompressor to reverse the encoding. Static schemes are similar to this, but a representative selection of data is used to build a fixed model, which is hard-coded into compressors and decompressors. This has the advantage that no model must be explicitly stored with the compressed data, but the disadvantage that poor compression will result if the model is not representative of data presented for compression. Concerning the type of adaptively being adopted, focus has remained on static and semi-adaptive techniques, and little attention has been paid to the class of adaptive algorithms (Klein, 2000) (Xie, Wolf, & Lekatsas, 2003) (Gilbert & Abrahamson, 2006).

**(4) Data compression cost**

The cost of data compression is an important feature that can be used to distinguish between the different data compression algorithms. Most importantly is that compression algorithms should be performed in as minimum as possible cost. This cost is measured in terms of time and storage requirement; however, in many applications, and with the revolutionary advancement in computer technology, the time is the most important factor. For example, on-the-fly compression algorithms, such as between application programs and storage devices, the algorithm should operate as quickly as the storage devices themselves. Likewise, if a compression algorithm is built into a hardware data communications component, the algorithm should not prevent the full bandwidth of the communication media from being continuously utilized.

The data compression cost for a particular algorithm consists of the time required by the algorithm to compress the original data and the time it takes to decompress the data back to its original form. In the context of the data compression for minimal storage applications, the cost of compression can be viewed as a one-time cost and hence as relatively less significant than the cost of decompression, which must be incurred every time the data is to be retrieved from storage. In the context of compression designed for fast data transmission applications, the relative costs of compression at one end and decompression at the other may be equally significant.

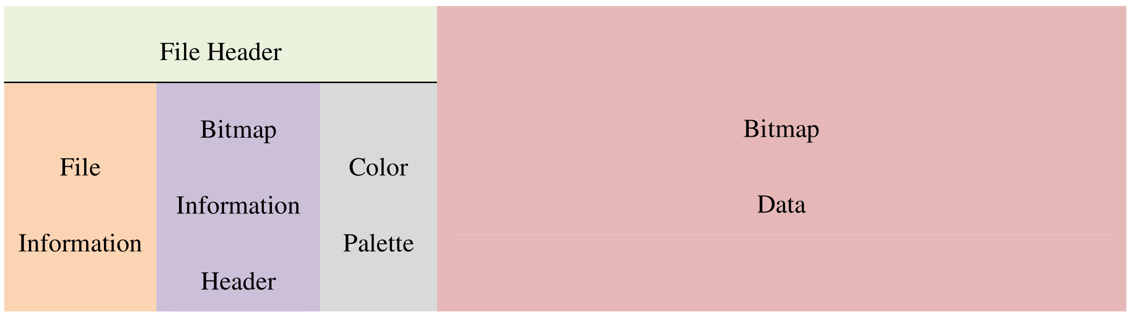
According to the compression-decompression processing time, data compression algorithms are classified into two classes, these are: symmetric and asymmetric data compression algorithms. In symmetric algorithms, the processing times are almost the same for compression and decompression processes; while in asymmetric algorithms the compression processing time is more than the decompression processing time.

## 1.3 Image Compression

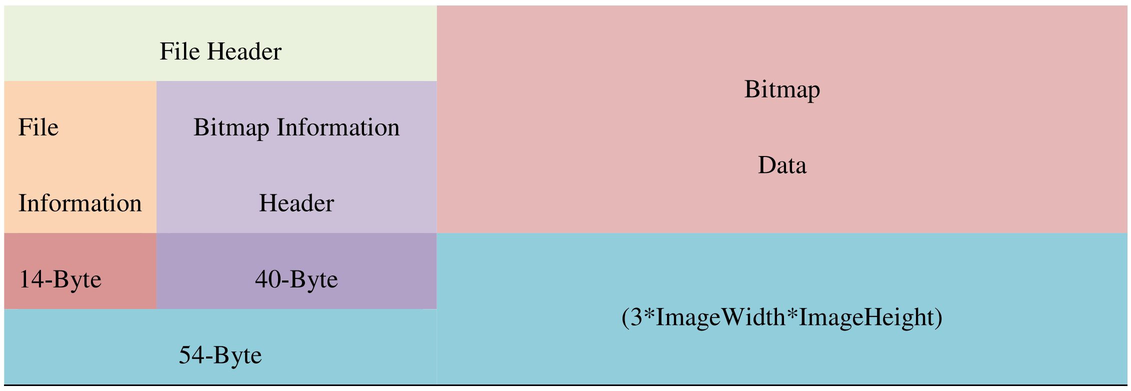
Before I proceed with our discussion on image compression, let us discuss first the standard uncompressed image file format, namely, the bitmap (BMP) image file format, which is used as the standard bitmap storage format in the Microsoft Windows environment. Although it is based on Windows internal bitmap data structures, it is supported by many non-Windows and non-PC applications.

The BMP, also known as Device Independent Bitmap (DIB) file format, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter), especially on Microsoft Windows and OS/2 operating systems. The BMP file format is capable of storing 2D digital images of arbitrary width, height, and resolution, both monochrome and color, in various color depths, and optionally with data compression, alpha channels, and color profiles (BMP file format, 2012).

The standard BMP files contain four sections, these are: file information, bitmap information header, color palette, and bitmap data, as shown in Figure (1.1). Of these four sections, only the palette information may be optional, depending on the bit depth of the bitmap data. The BMP file header is 14 Bytes in length. The file header is followed by a second header (called the bitmap information header), a variable-sized palette, and the bitmap data. There are four versions of BMP formats, they all have the same file information length (14 Bytes) but have various lengths for the remaining sections.



(a) Structure of a generic BMP image.



(b) Structure of Version 3.0 24-bit BMP image.

Figure (1.1). Structure of BMP image.

However, in this work I consider BMP Version 3 (Microsoft Windows 3.x), in which for 24-bit (RGB) color depth, the file size in Bytes is calculated as:

$$\text{ImageSize} = 54 + 3 * \text{ImageWidth} * \text{ImageHeight} \tag{1.1}$$

| | | |
|---|---|---|
| Where | ImageSize | the size of the image file in Bytes. |
| | ImageWidth | the image width in pixels. |
| | ImageHeight | the image height in pixels. |

The 54 is the total file header length, 14 Bytes of which is for file information, and 40 Bytes for bitmap information header.

A number of compressed image formats have been developed in the past for various applications, some of these algorithms are lossless (such as the Graphics Interchange Format (GIF) and the Portable Network Graphics (PNG) and other are lossy (such as the Joint Photographic Experts Group (JPEG)). The reduction in file size allows more images to be stored in a given amount of disk or memory space. It also reduces the time required for images to be sent over communication channels or downloaded through the Internet. Furthermore, it reduces the power consumption during transmission and also reduces accumulated transmission errors (Talukder & Harada, 2007).

Lossless compression is preferred for archival purposes and often for medical imaging, technical drawings, clip art, or comics. This is because lossy compression methods, especially when used at low bit rates, introduce compression artifacts. Lossy methods are especially suitable for natural images such as photographs in applications where minor (sometimes imperceptible) loss of fidelity is acceptable to achieve a substantial reduction in bit rate. The lossy compression that produces imperceptible differences may be called visually lossless (Bandyopadhyay, Paul, & Raychoudhury, 2011).

### 1.3.1 Image Compression Tools

While most BMP files have a relatively large file size due to lack of any compression (or generally low-ratio RLE on palletized images), many BMP files can be considerably compressed with well-known and widely-used lossless data compression tools, such as ZIP (http://www.winzip.com) and WinRAR (http://www.win-rar.com) (BMP file format, 2012).

ZIP is a compression and file packaging utility for Unix, VMS, MSDOS, OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, MVS, z/OS, Amiga, Acorn RISC, and other Operating Systems (OS). It is analogous to a combination of the Unix commands *tar* and *compress* (or *tar* and *gzip*) and is compatible with PKZIP (Phil Katz's ZIP for MSDOS systems) and other major ZIP utilities (ZIP, 2011). ZIP is not a single algorithm; there are many possible compression algorithms which may be used in ZIP archives (sometimes even several different algorithms in the same archive). The PK in PKZIP stands for the author Phil Katz, but the “ZIP” is just the name of the product and it means “spead” (i.e., not an acronym).

The second tool that is considered in this work is the WinRAR, which is a Windows compression utility developed by Alexandar Roshal to handle RAR (Roshal Archive) files and other compression formats; therefore, it is abbreviated as WinRAR. It supports the following features:

- Complete support of RAR and ZIP archives.
- Highly sophisticated, original lossless compression algorithm.
- Special algorithms optimized for text, audio, graphics, 32-bit and 64-bit Intel executable compression.
- Shell interface including drag-and-drop facility and wizard.
- Command line interface.

- Non-RAR archives (7Z, ACE, ARJ, BZ2, CAB, GZ, ISO, JAR, LZH, TAR, UUE, Z) management.
- Solid archiving, which can raise compression ratio by 10%-50% over common methods, particularly when packing a large number of small, similar files.
- Multivolume archives.
- Creation of self-extracting archives (also multivolume) using the default or optional SFX modules.
- Recovering physically damaged archives and it has recovery volumes allowing reconstructing missing parts of multivolume archives.
- Unicode support in file names.
- Other service functions, such as encryption, archive comments, error logging, etc.

### 1.3.2 Standard Compressed Image Formats

As it has been discussed above that there are a number of standard compressed image formats, such as GIF, PNG, and JPEG. The GIF and PNG are lossless, while the JPEG is lossy. In what follows, I provide a brief description for each of them:

- *Graphics Interchange Format (GIF)*

It is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the Web due to its wide support and portability. The format supports up to 8 bits per pixel thus allowing a single image to reference a palette of up to 256 distinct colors. The colors are chosen from the 24-bit RGB color space. It also supports animations and allows a separate palette of 256 colors for each frame. The color limitation makes the GIF format unsuitable for reproducing color photographs and other images with continuous color, but it is well-suited for simpler images such as graphics or logos with solid areas of color.

GIF images are compressed using the Lempel-Ziv-Welch (LZW) lossless data compression technique to reduce the file size without degrading the visual quality. This compression technique was patented in 1985. Controversy over the licensing agreement between the patent holder, Unisys, and CompuServe in 1994 spurred the development of the Portable Network Graphics (PNG) standard; however, all the relevant patents have now expired (Graphics Interchange Format, 2012).

- *Portable Network Graphics (PNG)*

It is a bitmapped image format that employs lossless data compression. PNG was created to improve upon and replace GIF as an image-file format not requiring a patent license. PNG supports palette-based images (with palettes of 24-bit RGB or 32-bit RGBA colors), grayscale images (with or without alpha channel), and full-color non-palette-based RGB images (with or without alpha channel). PNG was designed for transferring images on the Internet, not for professional-quality print graphics, and therefore does not support non-RGB color spaces such as Cyan, Magenta, Yellow, and Key (CMYK). PNG files nearly always use file extension PNG or png and are assigned Multipurpose Internet Mail Extensions (MIME) media type image/png; it was approved for this use by the Internet Engineering Steering Group (IESG) on 14 October 1996. PNG was published as an ISO/IEC standard in 2004 (Portable Network Graphics, 2013).

- *Joint Photographic Experts Group (JPEG)*

It is a commonly used method of lossy compression for digital photography (image). The degree of compression can be adjusted, allowing a selectable trade-off between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality. JPEG compression is used in a number of image file formats. JPEG/Exif is the most common image format used by digital cameras and other photographic image capture devices; along with JPEG/JFIF, it is the most common format for storing and transmitting photographic images on the Web. These format variations are often not distinguished, and are simply called JPEG.

The compression method is usually lossy, meaning that some original image information is lost and cannot be restored, possibly affecting image quality. There is an optional lossless mode defined in the JPEG standard; however, this mode is not widely supported in products.

There is also an interlaced "Progressive JPEG" format, in which data is compressed in multiple passes of progressively higher detail. This is ideal for large images that will be displayed while downloading over a slow connection, allowing a reasonable preview after receiving only a portion of the data. However, progressive JPEGs are not as widely supported, and even some software which does support them (such as versions of Internet Explorer before Windows 7) only displays the image after it has been completely downloaded. There are also many medical imaging and traffic systems that create and process 12-bit JPEG images, normally grayscale images. The 12-bit JPEG format has been part of the JPEG specification for some time, but again, this format is not as widely supported (JPEG, 2012).

## 1.4 Performance Measures

The performance of image compression algorithms is mainly measured in terms of the following performance metrics.

### (1) Compression ratio ($C$)

It is the most important and widely-used performance metric. It is defined as the ratio between the size of the original image file ($S_o$) and the size of the compressed image file ($S_c$). It is expressed mathematically as (Sayood, 2012):

$$C = \frac{S_o}{S_c} \tag{1.2}$$

| | | |
|---|---|---|
| Where | $C$ | Compression ratio. |
| | $S_o$ | size of the original image file. |
| | $S_c$ | size of the compressed image file. |

For lossy compression algorithms, two more performance metrics are introduced, namely, the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR).

### (2) Mean Square Error (MSE)

The MSE is the cumulative squared error between the compressed and the original images. A lower value for MSE means lesser error, and it is expressed as:

$$MSE = \frac{1}{X \cdot Y} \sum_{y=1}^{Y} \sum_{x=1}^{X} [I(x,y) - K(x,y)]^2 \tag{1.3}$$

| | | |
|---|---|---|
| Where | $MSE$ | Mean Square Error. |
| | $I(x,y)$ | Value of pixel at position $(x,y)$ of the original image. |
| | $K(x,y)$ | Value of pixel at position $(x,y)$ of the decompressed image. |

### (3) Peak Signal to Noise Ratio (PSNR)

The PSNR is a measure of the peak error. The PSNR is most commonly used as a measure of quality of reconstruction of lossy compression images, and it is usually expressed in terms of the logarithmic decibel (dB) scale as follows (Huynh-Thu & Ghanbari, 2008):

$$PSNR = 20 \cdot log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \tag{1.4}$$

Where $MAX_I$ the maximum possible pixel value of the image.

$MSE$ Mean Square Error.

When the pixels are represented using 8-bit per pixel, this is 255. More generally, when samples are represented using linear Pulse Code Modulation (PCM) with *n*-bit per pixel, $MAX_I$ is $2^n-1$. For color images with three RGB values per pixel, the definition of PSNR is the same except the MSE is the sum over all squared value differences divided by image size and by three. Alternately, for color images the image is converted to a different color space and PSNR is reported against each channel of that color space.

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 original image, and the noise is the error in reconstruction. So, if a compression scheme having a lower MSE (and a high PSNR), it can be recognized as a better one.

A higher PSNR would normally indicate that the reconstruction is of higher quality image. Typical values for the PSNR in lossy image and video compression are between 30 and 50 dB, where higher is better. Acceptable values for wireless transmission quality loss are considered to be about 20 dB to 25 dB (Thomos, Boulgouris, & Strintzis, 2006). In lossless compression, the two images are identical, and thus the MSE is zero. In this case the PSNR is undefined (Salomon, 2007).

In this thesis, the performance metrics for evaluating the performance of an image compression algorithm are represented in a performance triangle as shown in Figure (1.2). These metrics include: *C*, PSNR, and the processing resources (processor speed and memory). The first two metrics should be at their highest values, while the third should be minimized to its lowest value.

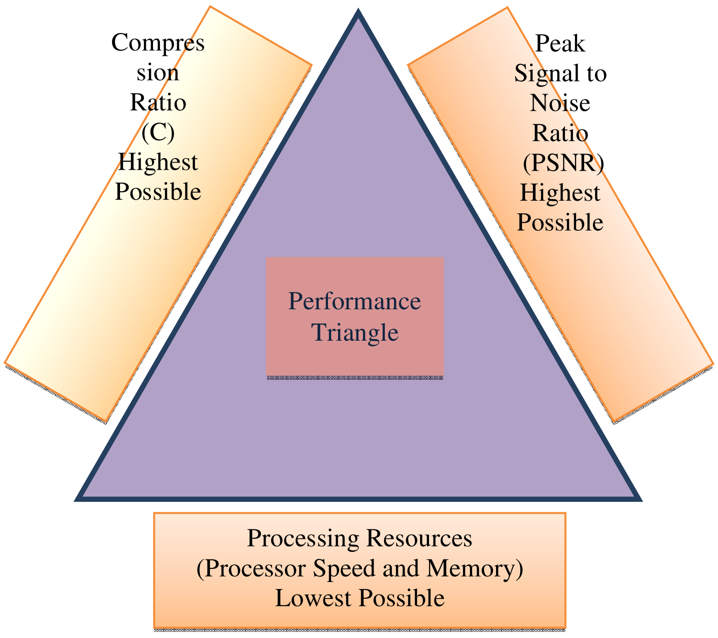

Compres sion Ratio (C) Highest Possible
Peak Signal to Noise Ratio (PSNR) Highest Possible
Performance Triangle
Processing Resources (Processor Speed and Memory) Lowest Possible

Figure (1.2). Performance metrics triangle.

## 1.5 The HCDC Data Compression Algorithm

A novel bit-level lossless data compression algorithm that is based on the error correcting Hamming codes, namely the Hamming Codes based Data Compression (HCDC) algorithm was initially developed and investigated for text compression (Al-Bahadili H. , 2008) (Al-Bahadili & Rababa'a, 2010) . In this algorithm, the binary sequence to be compressed is divided into blocks of *n*-bit length. To utilize the Hamming codes, the block is considered as a Hamming codeword that consists of *p* parity bits and *d* data bits ($n=d+p$). Then each block is tested to find if it is a valid or a non-valid Hamming codeword. For a valid block, only the *d* data bits preceded by 0 are written to the compressed file, while for a non-valid block all *n* bits preceded by 1 are written to the compressed file. These additional 0 and 1 bits are used to distinguish the valid and the non-valid blocks during the decompression process.

In this work, I propose a new high performance algorithm for image compression that is based on the novel lossless bit-level HCDC algorithm (Al-Bahadili H. , 2008). The HCDC algorithm has demonstrated an excellent performance and got the attention by many researchers around the world ( (Wang, Chang, Chen, & Li, 2010) (Al-Bahadili & Rababa'a, 2010) (Al-Saab, 2011) (AlZboun, 2011) (Amro, Zitar, & Bahadili, 2010) (Khancome, 2011) and many others)) due to the tremendous performance of the algorithm and the potential it has. This encourages us to take a step ahead to investigate its performance for image compression, and introduce any necessary modification to achieve the optimum performance, in particular, high compression ratio and high image quality. A detail description of the HCDC algorithm will be presented in Chapter 2.

## 1.6 Multimedia Messaging

Camera-equipped mobile phones nowadays can produce very high quality or resolution images or videos, which encourage mobile users to take more and more pictures or videos while they are on the move and exchange them as MMS. This of course requires high storage capacity to store the generated image data and also wide bandwidth to exchange data across the network. However, with tremendous advancement in processor, memory, and storage media technologies, most mobiles are supplied with large storage capacities of GB that can store large number of high resolution images or videos. Most mobiles are also equipped with wire and wireless interface technologies to flash-out images into more permanent or external storage media.

Despite the high storage capacity of current mobile devices, the device storage media may not be enough accommodating all generated data, or for other reasons users may need to clear or flash-out their mobile's storage media to a remote end while they are on the move or using wireless technologies. There are two possible ways: either utilizing the relatively expensive cellular network, which is usually available within range of commercial cell phone towers (nearly 10 Km range) and it has limited bandwidth, or utilize the high speed wireless LAN, which is free but it has very limited range (less than 200 m range), and also rarely available in urban areas. The most obvious way is to exchange these pictures or videos through the limited cellular bandwidth.

Furthermore, one important service provided for mobile users is MMS, which involves exchange of multimedia data. To improve the quality of this service, it is necessary to overcome problems of mobile memory requirement, mobile power consumption and life time, cellular network limited communication bandwidth, and meet all others users and application needs.

In this sense, one important factor to be considered is the mobile limited power and bandwidth recourses; so it is vital to manage power and bandwidth utilization efficiently, by identifying ways to use less energy and bandwidth preferably with no impact on the Quality-of-Services (QoSs).

The standard MMS can be divided into two main interrelated tasks: data processing and data communication. However, comparing the current cellular bandwidth in many areas around the world cannot meet the growing demands for satisfactory performance for multimedia exchange through cellular systems. Thus, more powerful compression algorithms and tools are required to minimize the size of the generated images while maintaining their quality.

## 1.7 Problem Statement

In order to enhance the performance of multimedia messaging, in particular stand-still images, through MMS across a mobile phone system (cellular network), it is necessary to reduce the size of the exchanged image to the minimum possible size while maintaining the maximum possible image quality. As a result of that more images can be stored in less storage space; reduces the communication time required for the image to be sent over communication networks or downloaded through the Internet (i.e., reduces bandwidth resources); and reduces power consumption. The same problem faces other applications, such as: satellite image exchange, Geographical Information System (GIS) applications, satellite and TV broadcast, and other image applications.

As it has been mentioned above that the HCDC algorithm has been successfully used for English text compression due to the almost exponential distribution of characters frequencies, where the sum of the frequencies of the first sixteen most common characters usually forms more than 85% of the total text. On the other side, the characters frequencies of color photographs and other images with continuous color are almost flat or equally distributed, as a result of that the original HCDC algorithm cannot achieve significant compression ratio and sometimes inflated. However, it may be well-suited for simpler images such as graphics or logos with solid areas of color.

In order to enhance the compression ratio of the HCDC algorithm for image compression, the algorithm needs to be modified, in particular, by increasing the frequencies of the most common characters to ensure satisfactory compression ratios, and also by decreasing the range of colors without affecting the compressed image color quality.

## 1.8 Objectives of this Work

This thesis develops a modified version of the HCDC algorithm to be used for standstill image compression. The modified version of the HCDC algorithm is characterized as an image-based adjustable-quality algorithm with high compression ratio; therefore, it is referred to as Adjustable-Quality HCDC and abbreviated as AQ-HCDC algorithm. It can be used efficiently and effectively for MMS or Internet applications on mobile cellular devices to provide high performance, where it can provide high compression ratio while maintaining high image quality, which means reduces MMS session bandwidth requirement and power consumption.

The main objectives of this work can be summarized as follows:

(1) Develop a new version of the HCDC algorithm for image compression, namely, the Adjustable-Quality HCDC (AQ-HCDC) algorithm.

(2) Evaluate the performance of the AQ-HCDC algorithm in terms of *C*, MSE, and PNSR by using the algorithm to compression a number of standard images of large and small sizes images.

(3) Compare the performance (*C*, MSE, and PNSR) of the AQ-HCDC algorithm against the performance of a number compressed image formats (GIF, PNG, JPEG) and a number of lossless compression tools (ZIP and WinRAR).

(4) Discuss the results obtained, draw some conclusions and point-out some recommendation for further development and future research.

## 1.9 Organization of the Thesis

The thesis is divided into five chapters. This chapter provides an introduction to the concept of data compression, data compression algorithms classification methodologies, data compression model, and data compression performance algorithms measures. It also introduces the problems statement and the objectives of this research. The rest of this thesis is organized as follows. Chapter 2 presents a literature review that provides a description of the core algorithm in this work, namely, the HCDC algorithm. Chapter 2 also summarizes the most recent and related work.

Chapter 3 provides a detail description of the AQ-HCDC algorithm. It also discusses the compressed image file header, the decompression process of the algorithm, and an analytical analysis of the performance of the AQ-HCDC algorithm. Chapter 4 presents a number of experiments that are performed to evaluate the compression ratio of the AQ-HCDC algorithm over a number of large and small images. Chapter 4 also presents a comparison between the compression ratios achieved by the AQ-HCDC algorithm against the compression ratios of a number of well-known compressed image formats and compression tools. In Chapter 5 and based on the results obtained conclusions are drawn and a number of recommendations for future work are pointed-out.

# Chapter Two
# Literature Review and Previous Work

There are a large number of data compression algorithms that have been developed and used throughout the years. Some of which are of general use, i.e., can be used to compress files of different types (e.g., text files, image files, video files, etc.). Others are developed to compress efficiently a particular type of files (Sayood, 2012) (Salomon, 2004) (Salomon, 2002). In this work, I concern with image compression, which has been developed using the two modes of compression, namely, lossless and lossy. Examples of lossless compression include: the Graphics Interchange Format (GIF), and the Portable Network Graphics (PNG), while as an example of lossy compression is the Joint Photographic Experts Group (JPEG). Furthermore, there are a number of general purpose tools that have been developed for general data compression including images, such as ZIP and WinRAR, which all have been discussed in Chapter 1.

This thesis is mainly concerned with the development and performance evaluation of a new algorithm for standstill image compression. The new algorithm is based on the lossless bit-level Hamming Codes based Data Compression (HCDC) algorithm (Al-Bahadili H. , 2008). The algorithm is originally developed and used for text compression (Al-Bahadili & Rababa'a, 2010). It also used by many researchers for compressing Web applications (Al-Saab, 2011), for speech compression (Amro et al., 2010) (AlZboun, 2011), and for lossless image compression (Wang et al., 2009).

The new algorithm is neither lossless nor lossy, as it is designed to act as adjustable-quality algorithm based on the range colors within the image. Therefore, it is referred to as Adjustable-Quality HCDC (AQ-HCDC) algorithm.

This chapter presents a detail description of the HCDC algorithm as it is the core of this research and also it provides a literature review on most recent research on image compression.

## 2.1 The HCDC Algorithm

This section presents a detail description of the HCDC algorithm, which is classified as a lossless, bit level, and asymmetric data compression algorithm (Al-Bahadili H. , 2008). Then it provides an analytical analysis of the performance of the algorithm. However, let us first discuss the concept of bit-level data compression.

In bit-level data compression, first, the data file should be represented in binary digits (bits). A data file can be represented in bits by concatenating the binary sequences of the characters within the file using a specific mapping or coding format, such as ASCII codes, Huffman codes, adaptive codes. Afterwards, a bit-level processing can be performed to reduce the size of the data files. The coding format has a huge influence on the entropy of the generated binary sequence and consequently the compression ratio ($C$) or the coding rate ($C_r$) that can be achieved.

Usually, in bit-level data compression algorithms, the binary sequence is subdivided into groups of bits that are called minterms, blocks, sub sequences, etc. In this thesis, I shall use the term blocks to refer to each group of bits. These blocks might be considered as representing a Boolean function. Then, algebraic simplifications for bit-reduction are performed on these Boolean functions to reduce the size or the number of blocks, and hence, the number of bits representing the data file is reduced as well.

### 2.1.1 Description of the HCDC algorithm

The error-correcting Hamming code has been widely used in computer networks and digital data communication systems as a single bit error correcting code or two bits errors detection code. It can also be tricked to correct burst errors. The key to Hamming code is the use of extra parity bits ($p$) to allow the identification of a single bit and a detection of two bits errors.

Thus, for a message having *d* data bits and to be coded using Hamming code, the coded message (also called codeword) will then have a length of *n*-bit, which is given by:

$$n = d + p \tag{2.1}$$

|  |  |  |
|---|---|---|
| Where | $n$ | length of codeword. |
|  | $d$ | length of data bits. |
|  | $p$ | length parity bits. |

This would be called a (*n*,*d*) Hamming code or simply code. The optimum length of the codeword (*n*) depends on *p*, and it can be calculated as:

$$n = 2^p - 1 \tag{2.2}$$

The data and the parity bits are located at particular locations in the codeword. The parity bits are located at positions $2^0$, $2^1$, $2^2$, …, $2^{p-1}$ in the coded message, which has at most *n* positions. The remaining positions are reserved for the data bits, as shown in Figure (2.1). Each parity bit is computed on different subsets of the data bits, so that it forces the parity of some collection of data bits, including itself, to be even or odd.

A lossless binary data compression algorithm based on the error correcting Hamming codes, namely the HCDC algorithm, was proposed by (Al-Bahadili H. , 2008). In this algorithm, the data symbols (characters) of a source file are converted to binary sequence by concatenated the individual binary codes of the data symbols.

The binary sequence is, then, subdivided into a number of blocks, each of *n*-bit length as shown in Figure (2.1b). The last block is padded with 0s if its length is less than *n*. For a binary sequence of $S_o$ bits length, the number of blocks *B* (where *B* is a positive integer number) is given by:

$$B = \left\lceil \frac{S_o}{n} \right\rceil \tag{2.3}$$

| | | |
|---|---|---|
| Where | $B$ | number of blocks. |
| | $S_o$ | size of the original image file. |
| | $n$ | length of codeword. |

The number of padding bits (*g*), which may be added to the last block is calculated by:

$$g = B \cdot n - S_o \tag{2.4}$$

The number of parity bits (*p*) within each block is given by:

$$p = \left\lceil \frac{\ln(n+1)}{\ln(2)} \right\rceil \tag{2.5}$$

For a block of *n*-bit length, there are $2^n$ possible binary combinations (codeword) having decimal values ranging from 0 to $2^n$-1, only $2^d$ of them are valid codewords and $2^n$-$2^d$ are non-valid codewords.

Each block is then tested to find if it is a valid block (valid codeword) or a non-valid block (non-valid codeword). During the compression process, for each valid block the parity bits are omitted, in other words, the data bits are extracted and written into a temporary compressed file. However, these parity bits can be easily retrieved back during the decompression process using Hamming codes. The non-valid blocks are stored in the temporary compressed file without change.

In order to be able to distinguish between the valid and the non-valid blocks during the decompression process, each valid block is preceded by 0, and each non-valid block is preceded by 1 as shown in Figure (2.1c). Figures (2.2) and (2.3) summarize the flow of the compressor and the decompressor of the HCDC algorithm.

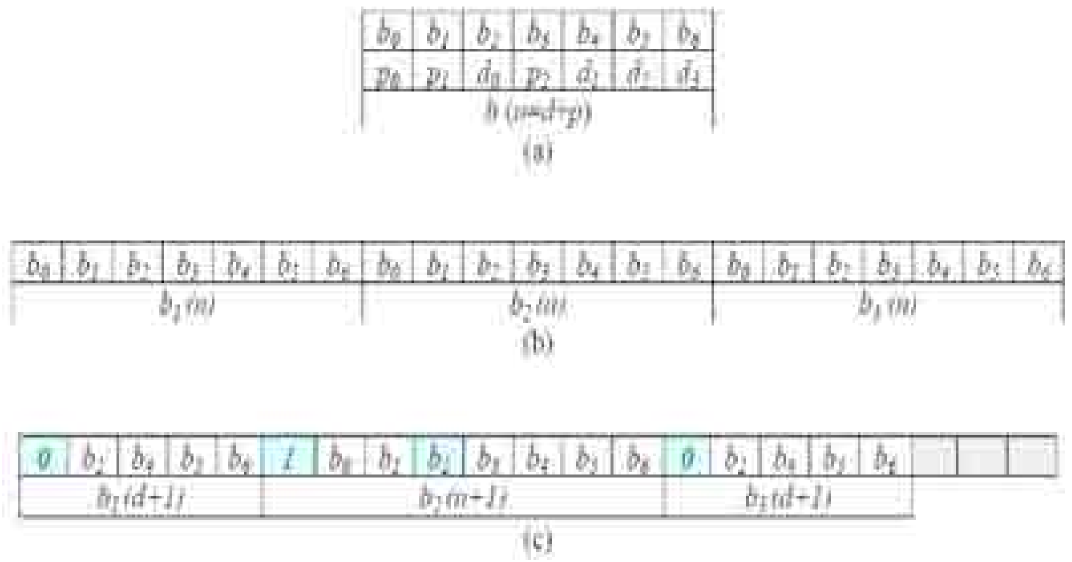


Figure (2.1). (a) Locations of data and parity bits in 7-bit codeword, (b) an uncompressed binary sequence of 21-bit length divided into 3 blocks of 7-bit length, where $B_1$ and $B_3$ are valid blocks, and $B_2$ is a non-valid block, and (c) the compressed binary sequence (18-bit length).

```
Initialization
Select $p$
Calculate $n = 2^p - 1$
Calculate $d = n - p$
Calculate $B$ = ceiling($S_o/n$)
Calculate $g = B * n - S_o$
Initialize $i = 0$
Reading binary data
While ($i<B$)
```

```
{
Read a block of n-bit length
Add 1 to i
Check for block validity
If  (block = valid codeword) then
Add 1 to v   // v is the number of valid blocks
Extract the data bits (d-bit)
Write 0 followed by the extracted d-bits to the temporary compressed file
Else (block = non-valid codeword)
Add 1 to w  // w is the number non-valid blocks
Write 1 followed by all n-bits to the temporary compressed file
End if
}
```

Figure (2.2). The main steps of the HCDC compressor.

```
Initialization
Select p
Calculate $n = 2^p - 1$
Calculate $d = n - p$
Initialize $i = 0$
Reading binary data
While (not end of data)
{
Read one bit (h)
Add 1 to i
Check for block validity
If  (h = 0) then
Add 1 to v // v is the number of valid blocks
Read the following $d$ data bits
Compute the Hamming codes for these $d$ data bits
Write the coded block the temporary decompressed binary sequence
Else ($h$ = 1) then
Add 1 to $w$ // $w$ is the number of non-valid blocks
Read a block of $n$ bits length
Write $n$ bits block to the  temporary decompressed  binary sequence
End if
}
```

Figure (2.3). The main steps of the HCDC decompressor.

### 2.1.2 Derivation and analysis of HCDC algorithm compression ratio

This section presents the analytical derivation of a formula that can be used to compute the compression ratio achievable using the HCDC algorithm. The derived formula can be used to compute *C* as a function of two parameters:

(1) The block size (*n*).

(2) The fraction of valid blocks (*r*).

In the HCDC algorithm, the original binary sequence is divided into *B* blocks of *n*-bit length. These *B* blocks are either valid or non-valid blocks; therefore, the total number of blocks is given by:

$$B = v + w \tag{2.6}$$

| | | |
|---|---|---|
| Where | $B$ | number of blocks. |
| | $v$ | the number of valid blocks. |
| | $w$ | the number non-valid blocks. |

For a valid block only the *d* data bits preceded by 0 are appended to the compressed binary sequence (i.e., *d*+1 bits for each valid block). So that the length of the compressed valid blocks ($S_v$) is given by:

$$S_v = v\,(d + 1) \tag{2.7}$$

| | | |
|---|---|---|
| Where | $S_v$ | length of the compressed valid blocks. |
| | $v$ | the number of valid blocks. |
| | $d$ | Length of data bits. |

For a non-valid block all bits are appended to compressed binary sequence (i.e., *n*+1 bits for each non-valid block). The number of bits appended to the compressed binary sequence is given by:

$$S_w = w\,(n + 1) \tag{2.8}$$

| | | |
|---|---|---|
| Where | $S_w$ | length of the compressed non-valid blocks. |
| | $w$ | the number of non-valid blocks. |
| | $n$ | length of the codeword. |

Thus, the length of the compressed binary sequence ($S_c$) can be calculated by:

$$S_c = S_v + S_w = v\,(d + 1) + w\,(n + 1) \tag{2.9}$$

Using Eqns. (2.6) and (2.7), Eqn. (2.9) can be simplified to

$$S_c = Bn + B - v \cdot p \tag{2.10}$$

Substituting $S_o$=$nB$ and $S_c$ as it is given by Eqn. (2.10) into the equation of the compression ratio ($C$) yields:

$$C = \frac{S_o}{S_c} = \frac{n}{n + 1 - r\ p} \tag{2.11}$$

Where *R* *v*/*B*.

*r* it represents the fraction of valid blocks. Substitute Eqn. (2.2) into Eqn. (2.11) gives:

$$C = \frac{2^p - 1}{2^p - r\ p} \tag{2.12}$$

It is clear from Eqn. (2.12) that, for a certain value of $p$, $C$ is inversely proportional to $r$, and $C$ is varied between a maximum value ($C_{max}$) when $r$=1 and a minimum value ($C_{min}$) when $r$=0. It can also be seen from Eqn. (2.12) that for each value of $p$, there is a value of $r$ at which $C$=1. This value of $r$ is referred to as $r_1$, and it can be found that $r_1$=1/$p$.

Table (2.1) lists the values of $C_{max}$, $C_{min}$, and $r_1$ for various values of $p$. These results are also shown in Figures (2.4) and (2.5), where Figure (2.4) shows the variation of $C_{max}$ and $C_{min}$ with $p$, and Figure (2.5) shows the variation of $r_1$ with $p$.

Table (2.1)
Variation of $C_{min}$, $C_{max}$, and $r_1$ with number of parity bits ($p$).

| $P$ | $C_{min}$ | $C_{max}$ | $r_1$ |
|---|---|---|---|
| 2 | 0.750 | 1.500 | 0.500 |
| 3 | 0.875 | 1.400 | 0.333 |
| 4 | 0.938 | 1.250 | 0.250 |
| 5 | 0.969 | 1.148 | 0.200 |
| 6 | 0.984 | 1.086 | 0.167 |
| 7 | 0.992 | 1.050 | 0.143 |
| 8 | 0.996 | 1.028 | 0.125 |

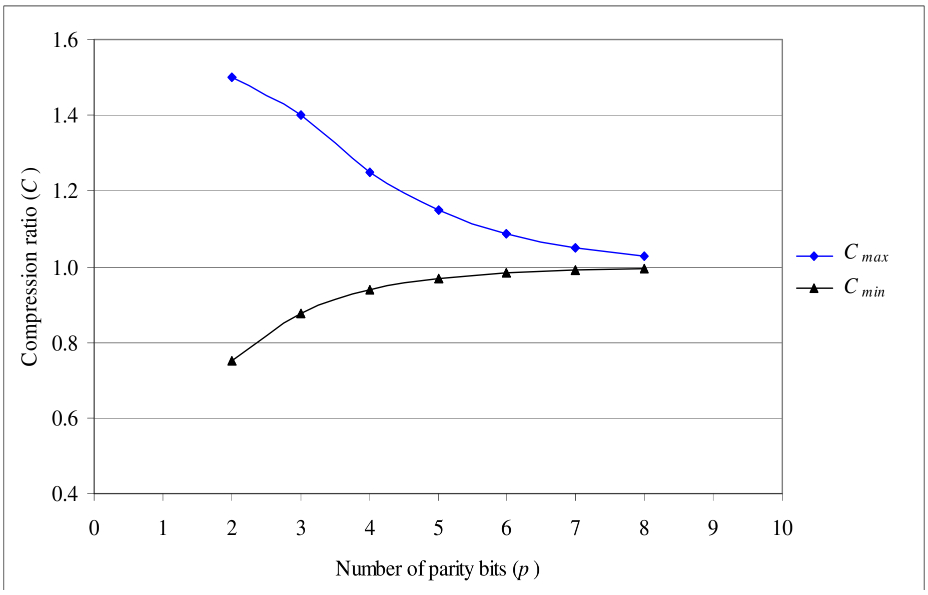


Figure (2.4). Variation of $C_{min}$ and $C_{max}$ with $p$.

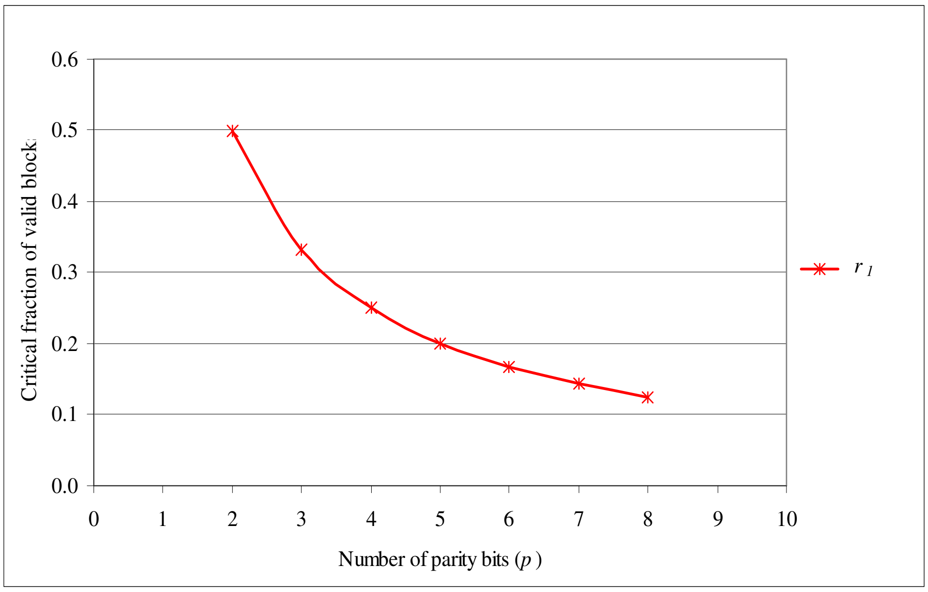


Figure (2.5). Variation of $r_1$ with $p$.

Furthermore, in order to demonstrate the effect of $r$ on $C$ for various $p$, Table (2.2) lists the values for $C$ when $r$ varies between 0 and 1 in step of 0.1 and $p$ varies between 2 to 8 in step of 1. These values are also plotted in Figure (2.6). It can be deduced from Table (2.2) and Figure (2.6) that satisfactory values of $C$ can be achieved when $p \leq 4$ and $r > r_1$.

Table (2.2)
Variations of *C* with respect to *r* for various values of *p*.

| *R* | Number of the parity bits (p) | | | | | | |
|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0.0 | 0.750 | 0.875 | 0.938 | 0.969 | 0.984 | 0.992 | 0.996 |
| 0.1 | 0.789 | 0.909 | 0.962 | 0.984 | 0.994 | 0.998 | 0.999 |
| 0.2 | 0.833 | 0.946 | 0.987 | 1.000 | 1.003 | 1.003 | 1.002 |
| 0.3 | 0.882 | 0.986 | 1.014 | 1.016 | 1.013 | 1.009 | 1.006 |
| 0.4 | 0.938 | 1.029 | 1.042 | 1.033 | 1.023 | 1.014 | 1.009 |
| 0.5 | 1.000 | 1.077 | 1.071 | 1.051 | 1.033 | 1.020 | 1.012 |
| 0.6 | 1.071 | 1.129 | 1.103 | 1.069 | 1.043 | 1.026 | 1.015 |
| 0.7 | 1.154 | 1.186 | 1.136 | 1.088 | 1.054 | 1.032 | 1.018 |
| 0.8 | 1.250 | 1.250 | 1.172 | 1.107 | 1.064 | 1.038 | 1.022 |
| 0.9 | 1.364 | 1.321 | 1.210 | 1.127 | 1.075 | 1.044 | 1.025 |
| 1.0 | 1.500 | 1.40 | 1.250 | 1.148 | 1.086 | 1.050 | 1.028 |

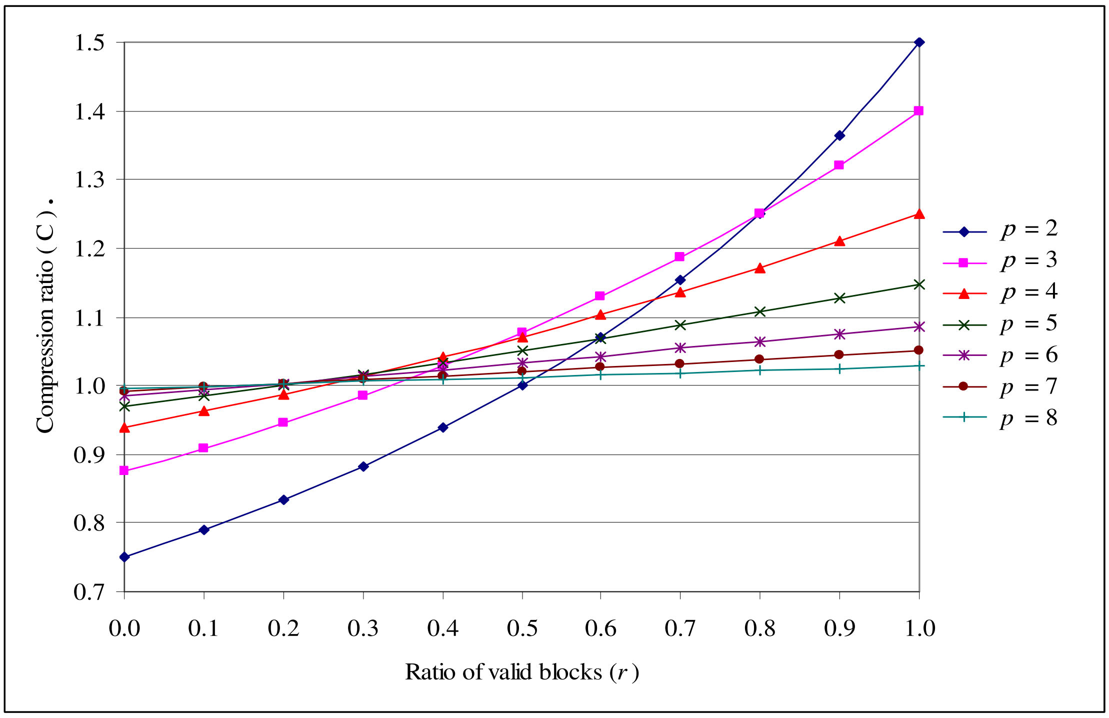


Figure (2.6). Variations of *C* with respect to *r* for various values of *p*.

One important feature of the HCDC algorithm is that it can be repeatedly applied on the binary sequence, and an equation can be derived to compute, what I refer to as the accumulated compression ratio ($C_k$):

$$C_k = \prod_{i=1}^{k} C_i = \prod_{i=1}^{k} \frac{S_{i-1}}{S_i} \tag{2.13}$$

| | | |
|---|---|---|
| Where | $k$ | number of repetitions. |
| | $S_i$ and $S_{i-1}$ | sizes of the binary file after and before the $i^{th}$ compression loop. |
| | $C_i$ | is the compression ratio of the $i^{th}$ compression loop. |

For $i$=1, $S_o$ represents the size of the original file.

## 2.2 Previous Work

The section reviews some of the researches that have been developed during the last two decades. The reviewed work is ordered chronologically from the most recent to the oldest and for the same year it is ordered alphabetically using the first author family name.

- (Douak, Benzid, & Benoudjit, 2011) designed a lossy image compression algorithm dedicated to color still images. After a preprocessing step (mean removing and RGB to YCbCr transformation), the DCT transform is applied and followed by an iterative phase (using the bisection method) including the thresholding, the quantization, dequantization, the inverse DCT, YCbCr to RGB transform and the mean recovering. This is done in order to guarantee that a desired quality (fixed in advance using the well-known PSNR metric) is checked. To obtain the best possible compression ratio, the next step is the application of a proposed adaptive scanning providing, for each (*n*, *n*) DCT block a corresponding (*n*×*n*) vector containing the maximum possible run of zeros at its end. The last step is the application of a modified systematic lossless encoder. The efficiency of the proposed scheme is demonstrated by results, especially, when faced to the method presented in recent research based on the block truncation coding using pattern fitting principle.

Image enhancement and data compression methods arose from the distinct and largely separate disciplines of image processing and communications respectively, yet both are important components of current and future digital imaging systems technology. (Rahman, Jobson, & Woodell, 2011) examined the relationship of these two components with special emphasis on image enhancement and lossy JPEG image compression. When transmission channel capacity is limited, image/data compression is often performed to increase the data throughput. However, this compression has a significant impact on the quality of the final data that is received. In most cases, image enhancement performed *after* image compression tends to bring out the artifacts injected into the data due to the compression. However, if image enhancement is performed *before* image compression, there are two issues that arise: (i) image enhancement typically increases the contrast, amount of observable detail, in an image which leads to poorer compression ratios, and (ii) the radiometric information in the original data is typically irretrievably lost.

(Rahman et al., 2011) addressed the impact of image enhancement specifically that of the Multi-Scale Retinex with Color Restoration (MSRCR) on image compression, and vice versa. We also look at the impact of compression on recovering original data from enhanced imagery given certain parameters about the enhancement process. In this context, we also develop an inversion process for the MSRCR.

- (Singh & Kumar, 2011) developed an Image Dependent Color space Transform (ID-CCT), exploiting the inter-channel redundancy optimally and which is very much suitable compression for large class of images. The comparative performance evaluated and a significant improvement was observed, objectively as well as subjectively over other quantifiable methods.

- (Telagarapu, 2011) analyzed the performance of a hybrid data compression scheme that uses the Discrete Cosine Transform (DCT) and Wavelet transform. They carried-out extensive experimentation and concluded that selecting proper threshold method, better result for Peak Signal to Noise Ratio (PSNR) can be obtained.

- Binary Wavelet Transform (BWT) has several distinct advantages over the Real Wavelet Transform (RWT), such as the conservation of alphabet size of wavelet coefficients, no quantization introduced during the transform and the simple Boolean operations involved. Thus, less coding passes are engaged and no sign bits are required in the compression of transformed coefficients. However, the use of BWT for the embedded grayscale image compression is not well established.

  (Pan, Jin, Yuan, Xia, & Xia, 2010) proposed a novel Context-based Binary Wavelet Transform Coding approach (CBWTC) that combines the BWT with a high-order context-based arithmetic coding scheme to embedded compression of grayscale images. In the CBWTC algorithm, BWT is applied to decorrelate the linear correlations among image coefficients without expansion of the alphabet size of symbols. In order to match up with the CBWTC algorithm, they employed the Gray Code Representation (GCR) to remove the statistical dependencies among bi-level bit-plane images and developed a combined arithmetic coding scheme.

  In the proposed combined arithmetic coding scheme, three high-pass BWT coefficients at the same location are combined to form an octave symbol and then encoded with a ternary arithmetic coder. In this way, the compression performance of the CBWTC algorithm is improved in that it not only alleviate the degradation of predictability caused by the BWT, but also eliminate the correlation of BWT coefficients in the same level sub-bands. The conditional context of the CBWTC is properly modeled by exploiting the characteristics of the BWT as well as taking advantages of non-causal adaptive context modeling. Experimental results show that the average coding performance of the CBWTC is superior to that of the state-of-the-art grayscale image coders, and always outperforms the JBIG2 algorithm and other BWT-based binary coding technique for a set of test images with different characteristics and resolutions.

- (Ameer & Basir, 2009) described a simple scheme to compress images through plane fitting. The scheme can achieve better than 60:1 compression ratio, while maintaining acceptable image quality. The results are superior to those of JPEG at comparable compression ratios. The scheme does not require any multiplication or division operations, making it a perfect candidate for online and/or progressive compression. The scheme is scalable in the context of computations required to magnify the image. Blocking effects were reduced up to 0.85 dB of PSNR through simple line fitting on block boundaries. The performance of the scheme is further improved by optimizing its predicted model parameters based on previously coded neighboring blocks. It is found that less than 2 bits (on average) are enough to index the position of the candidate neighbor, making a 100:1 compression ratio possible. The improvement in the compression ratio came at the expense of moderate to small quality degradations.

- (Dudek, Borys, & Grzywna, 2007) reported on the new method of image compression. The method is based on LZ77 dictionary algorithm. They introduced two modifications such as quantization and noise levels. Their experimental results prove that the new method of image compression gives promising results as compared with original LZ77 dictionary algorithm and JPEG2000.

- (Krinidis, Nikolaidis, & Pitas, 2007) This paper introduces the Discrete Modal Transform (DMT), a 1D and 2D discrete, non-separable transform for signal processing, which, in the mathematical sense, is a generalization of the well-known Discrete Cosine Transform (DCT). A 3D deformable surface model is used to represent the image intensity and the introduced discrete transform is a by-product of the explicit surface deformation governing equations. The properties of the proposed transform are similar to those of the DCT. To illustrate these properties, the proposed transform is applied to lossy image compression and the obtained results are compared to those of a DCT-based compression scheme. Experimental results show that DMT, which includes an embedded compression ratio selection mechanism, has excellent energy compaction properties and achieves comparable compression results to DCT at low compression ratios, while being in general better than DCT at high compression ratios.

- (Krishnamoorthi & Seetharaman, 2007) proposed a family of stochastic models for image compression, where images are assumed to be Gaussian Markov random field. This model is based on stationary Full Range Auto Regressive (FRAR) process. The parameters of the model are estimated with the Monte-Carlo integration technique based on Bayesian approach. The advantage of the proposed model is that it helps to estimate the finite number of parameters for the infinite number of orders. They used arithmetic coding to store seed values and parameters of the model as it gives furthermore compression. They also studied the use of Metropolis–Hastings algorithm to update the parameters, through which some image contents such as untexturedness are captured. Different types–both textured and untextured images–are used for experiment to illustrate the efficiency of the proposed model and the results are encouraging.

- (Velisavljevic, Beferull-Lozano, & Vetterli, 2007) combined the directionlets with the Space-Frequency Quantization (SFQ) image compression method, originally based on the standard 2-D wavelet transform. They showed that their compression method outperforms the standard SFQ as well as the state-of-the-art image compression methods, such as SPIHT and JPEG-2000, in terms of compressed image quality, especially in a low-rate compression regime. They also showed that the order of computational complexity remains the same, as compared to the complexity of the SFQ algorithm.

- A very interesting comparison of different image compression formats can be found in (Aguilera, 2006), in which she made a comparison of some of the most used image representation formats on a set of different types of images: true color, greyscale, scanned documents and high resolution photographs. She performed many investigations to find-out how well the different formats work for each of the images, and concluded that some formats that match some images better than others depending in what users are looking for, and the type of image they are working with.

- (Cagnazzo, Cicala, Poggi, & Verdoliva, 2006) considered the class-based multispectral image coder originally proposed in (Gelli & Poggi, 1999), and modify it to allow its use in real time with limited hardware resources. Experiments carried out on several multispectral images showed that the resulting unsupervised coder has a fully acceptable complexity, and a rate–distortion performance which is superior to that of the original supervised coder, and comparable to that of the best coders known in the literature.

- The standard separable two-dimensional (2-D) Wavelet Transform (WT) achieved a great success in image processing because it provides a sparse representation of smooth images. (Velisavljevic, Beferull-Lozano, Vetterli, & Dragotti, 2006) proposed a construction of critically sampled perfect reconstruction anisotropic transform with Directional Vanishing Moments (DVM) imposed in the corresponding basis functions, called directionlets. Later on, they showed that the computational complexity of their transform is comparable to the complexity of the standard 2-D WT and substantially lower than the complexity of other similar approaches. They also presented a zero tree-based image compression algorithm using directionlets that strongly outperforms the corresponding method based on the standard wavelets at low bit rates.

- One particular approach for image compression is the image cluster compression, surprisingly little previous work can be found about this approach. However, a number of special cases are available, though, most notably the approach of hyperspectral compression, which deals with the coding of a number (heavily correlated) images of the same dimensions. (Saghri, Tescher, & Reagan, 2005) developed a multispectral 3D image compression algorithm, which is then refined and adapted in a number of other papers, for example (Du & Chang, 2004). Lossless compression of correlated hyperspectral images is discussed in (Wang, Babacan, & Sayood, 2007). (Keshava, 2004) dealt with distance metrics between spectra. Another special case for cluster compression is the area of digital video compression, e.g., (Wen, et al., 2004) gave an overview over the current state-of-the-art. Lately, (Kramm, 2007) extended the Karhunen-Loeve compression algorithm to multiple images. The resulting algorithm is compared against single-image Karhunen Loeve as well as algorithms based on the DCT. Furthermore, various methods for obtaining compressible clusters from large image databases were evaluated.

- (Edirisinghe, Nayan, & Bez, 2004) proposed a novel, Discrete Wavelet Transform (DWT) domain implementation of the pioneering block-based disparity compensated predictive coding algorithm for stereo image compression. They performed predictive coding in the form of pioneering block search in the sub-band domain. The resulting transform domain predictive error image is subsequently converted to a so-called wavelet-block representation, before being quantized and entropy coded by a JPEG-like CODEC. They showed that their novel implementation is able to effectively transfer the inherent advantages of DWT-based image coding technology to efficient stereo image pair compression. At equivalent bit rates, the proposed algorithm achieves peak signal to noise ratio gains of up to 5.5 dB, for reconstructed predicted images, as compared to traditional and state of the art DCT and DWT-based predictive coding algorithms.

- Vector quantization (VQ) is an important technique in digital image compression. To improve its performance, (Li, Kim, & Al-Shamakhi, 2002) worked on speeding up the design process and achieve the highest compression ratio possible. To speed up the process, they used a fast Kohonen self-organizing neural network algorithm to achieve big saving in codebook construction time. To obtain better reconstructed images, they proposed an approach called the Transformed Vector Quantization (TVQ), combining the features of transform coding and VQ. They used several data sets to demonstrate the feasibility of the TVQ approach. A comparison of reconstructed image quality is made between the TVQ and VQ. Also, a comparison is made between a TVQ and a standard JPEG approach.

- (Mateu-Villarroya & Prades-Nebot, 2001) developed a lossless compression algorithm for images based on Ordered Binary-Decision Diagrams (OBDDs). The algorithm finds an OBDD which represents the image exactly and then codes the OBDD efficiently. The results obtained show a great improvement with respect to a previous work.

- (Hu & Chang, 2000) proposed a novel lossless image-compression scheme. A two-stage structure is embedded in this scheme. A linear predictor is used to decorrelate the raw image data in the first stage. Then in the second stage, an effective scheme based on the Huffman coding method is developed to encode the residual image. This newly proposed scheme could reduce the cost for the Huffman coding table while achieving high compression ratio. With this algorithm, a compression ratio higher than that of the lossless JPEG method for 512×512 images can be obtained. In other words, the newly proposed algorithm provides a good means for lossless image compression.

# Chapter Three

# The Adjustable-Quality HCDC Algorithm

It has been shown in Chapter 2 that the compression ratio (*C*) of the lossless Hamming Codes based Data Compression (HCDC) algorithm, and consequently the algorithm compression efficiency (Ec), depends on the distribution of the characters frequencies of the file (data) it is compressing (Al-Bahadili H. , 2008). Practically, *C* and Ec are increasing when the sum of the frequencies of a certain range of the most common characters increase, for example, for 7-bit Hamming codewords length, the frequencies of the first sixteen most common characters. This requirement is typically achieved in Standard English text, where the sum of the frequencies of the first sixteen most common characters usually forms more than 85% of the total text.

The characters frequencies of color photographs and other images with continuous color are almost flat or equally distributed, as a result of that the original HCDC algorithm cannot achieve significant compression ratio and sometimes inflated. However, it may be well-suited for simpler images such as graphics or logos with solid areas of color. In order to enhance the compression ratio of the HCDC algorithm for image compression, the algorithm needs to be modified, in particular, by increasing the frequencies of the most common characters to ensure satisfactory compression ratios, and also by decreasing the range of colors without affecting the compressed image color quality.

This chapter develops a modified version of the HCDC algorithm to be used for standstill image compression. The modified version of the HCDC algorithm is characterized as an image-based adjustable-quality lossless/lossy algorithm with high compression ratio; therefore, it is referred to as Adjustable-Quality HCDC and abbreviated as AQ-HCDC algorithm. It can be used efficiently and effectively for Multimedia Messaging Service (MMS) applications on mobile cellular devices to provide high performance, where it can provide high compression ratio while maintaining image quality, which means reduces bandwidth requirement and power consumption during MMS.

Section 3.1 presents a detail description of the compression process of the AQ-HCDC algorithm. The compressed image file header is described in Section 3.2. In Section 3.3, the decompression process of the algorithm is explained. An analytical analysis of the performance of the AQ-HCDC algorithm is discussed in Section 3.4.

## 3.1 The Compression Process of the AQ-HCDC Algorithm

The AQ-HCDC algorithm is an adjustable-quality bit-level data compression algorithm especially develops for standstill image compression, where the quality of the compressed image depends on the range of colors values in the original image. The algorithm performs lossless or lossy compression depending on the image data. It is similar to the HCDC algorithm, where it relies on Hamming codes representations to replace longer valid and non-valid binary Hamming codewords into shorter representations plus a pre-fix bit. This section describes in details how the lossless HCDC algorithm is modified to performing adjustable-quality image compression.

At the beginning, it is important to remember that the HCDC algorithm can use different block length ($n=2^p-1$ and $n=p+d$, where $p$ is the number of parity bits and $d$ is the number of data bits in Hamming codes) as explained in Chapter 2. However, as it has been discussed in (Al-Bahadili H. , 2008) and approved in (Al-Bahadili & Rababa'a, 2010) that the most suitable value for $n$ is 7, where in this case $p$=3 and $d$=4, and that what will be used in this work.

The AQ-HCDC algorithm consists of two main phases, these are:

(1) The Encoding Phase.

(2) The Compression Phase.

These two phases will be described in details below:

***Encoding Phase***

In this thesis, in order to re-encode the image data using 7-bit color depth, the AQ-HCDC algorithm performs the following pre-processing phase, which I will call the Encoding Phase. In this phase, the data of the uncompressed image is read one byte or character at a time; where each byte represents 8-bit color value ($V_i$) between 0-255 (from now onward I shall use the term color instead of byte or character to distinguish it from text compression), and the following steps are performed:

(1) Find the number of colors ($N_{co}$), according to their decimal values (8-bit color depth).

(2) Find the occurrence (counts) of each color ($O_i$).

(3) Calculate the colors frequencies ($F_i$) by dividing the counts of the color by the sum of counts of all colors; i.e., $F_i=O_i/N$. $i$ is the color index (i=1, 2, …, $N_{co}$), and $N$ is the sum of counts of all colors and it is expressed mathematically as:

$$N = \sum_{i=1}^{N_{co}} O_i \qquad (3.1)$$

|  |  |  |
|---|---|---|
| Where | $N$ | counts of all colors. |
|  | $O_i$ | counts of each color. |
|  | $N_{co}$ | number of color (8-bit color depth). |

(4) Sort the colors frequencies from the most common to the least common color.

If $N_{co}$ exceeds 128 colors ($N_{co}$>128), discard the Least-Significant-Bit (LSB) of the color value and shifts the remaining color bits one place to the right, which in turn, converts the 8-bit color to 7-bit color producing $V_i$ between 0-127 (in other words, this is equivalent to subtracting 128 from each original $V_i$). This is illustrated in Figure (3.1).

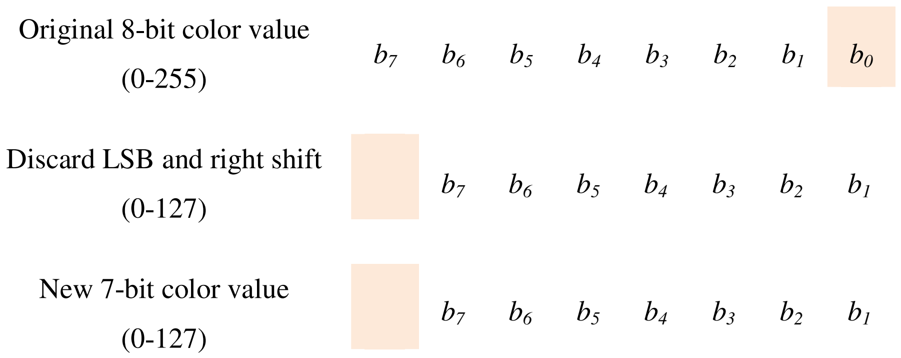

Original 8-bit color value
(0-255)
$b_7$ $b_6$ $b_5$ $b_4$ $b_3$ $b_2$ $b_1$ $b_0$
Discard LSB and right shift
(0-127)
$b_7$ $b_6$ $b_5$ $b_4$ $b_3$ $b_2$ $b_1$
New 7-bit color value
(0-127)
$b_7$ $b_6$ $b_5$ $b_4$ $b_3$ $b_2$ $b_1$

Figure (3.1). Illustration of the 8-bit to 7-bit encoding process.

If $N_{co}$ is equal to or less than 128 ($N_{co} \leq 128$), then I use the adaptive coding described in (Al-Bahadili & Rababa'a, 2010). In the adaptive coding, the colors values are sorted in descending order from the most common to the least common and stored in the compressed file header preceded by $N_{co}$. Then each color will be given a 7-bit value equivalent to its sorted sequence number, which is given the values from 0 to $N_{co}$-1. This means that the most common color value will be given a 0 decimal value or 0000000 in binary. The second most common color will assign a value of 1 in decimal or 0000001 in binary, and so on.

By the end of this pre-processing Encoding Phase, I end-up with a new number of colors ($N_{ce}$) that is equal to or less than $N_{co}$ ($N_{ce} \leq N_{co}$), where $N_{ce}$ is the number of colors after the Encoding Phase, practically, $N_{ce}=N_{co}$ when $N_{co} \leq 128$ otherwise $N_{ce}<N_{co}$.

When $N_{co} \leq 128$, the compression process is still lossless as I have not changed the colors values; otherwise, I enter the lossy region as some of the colors have their values changed. For color photographs and other images with continuous color, I always expect $N_{co}>128$. In this case, if the size of the compressed file header is neglected as compared to the size of the compressed image, the Encoding Phase provides a compression ratio of 1.125.

According to the Encoding Phase, there two main points that can be recognized:

(1) Encoding 8-bit color value to right-shifted 7-bit color value looks very acceptable in image compression as it achieves 12.5% reduction in image size while either reduces color value by 1 or keep it unchanged.

(2) It is not recommended using the AQ-HCDC algorithm to compress compressed images due to the lossy behavior of the algorithm. However, it can be used to compress 8-bit, 16-bit, 24-bit uncompressed images.

Based on the discussion above, Figure (3.2) outlines the procedure of the Encoding Phase of the AQ-HCDC algorithm.

***Procedure of the Encoding Phase of the AQ-HCDC Algorithm.***

```
Read image data one Byte (8-bit color depth) after another.
Find the number of colors ($N_{co}$).
Find the occurrence ($O_i$) or counts of each color.
Calculate the sum of counts of all colors ($N$) (Eqn. 3.1)
Calculate the colors frequencies ($F_i$), where $F_i=O_i/N$.
Sort the colors frequencies from the most common to the least common color.
If ($N_{co} > 128$),
Convert 8-bit color to 7-bit color producing $V_i$ between 0-127.
Calculate the new number of colors ($N_{ce}$)
Else
Use adaptive coding described above to represent the colors values.
$N_{ce}=N_{co}$
End If
```

Figure (3.2). Procedure of the Encoding Phase of the AQ-HCDC algorithm.

***Compression Phase***

This is the second and the main core of the AQ-HCDC algorithm. This phase consists of a number of steps, which can be summarized as follows:

(1) If $N_{ce}\leq 16$: The image data can be compressed using one of the following modes.

a. ***Mode 1***:

i. Set $N_{cc}=N_{ce}$, where $N_{cc}$ represents the number of colors that will be presented in the compressed image.

ii. Calculate $m$ as the number of bits required to represent the colors in the compressed image, and $m$ is calculated as:

$$m = \left\lceil \frac{\ln(N_{cc})}{\ln(2)} \right\rceil \quad (3.2)$$

| | | |
|---|---|---|
| Where | $M$ | number of bits required to represent the colors in the compressed image. |
| | $N_{cc}$ | $N_{cc}$ represents the number of colors that will be presented in the compressed image. |

So that $m$ can be 1 ($1<N_{cc}\leq 2$), 2 ($2<N_{cc}\leq 4$), 3 ($4<N_{cc}\leq 8$), or 4 ($8<N_{cc}\leq 16$) depending on the number of colors.

iii. Construct a header for the compressed image containing all information necessary for the decompression process, create a compressed image file, and store the header into the compressed image file. Description of the compressed image header will be given in the next section.

iv. Read the image data and replace each color with $m$-bit binary value equivalent to its sorted sequence number; starting from 0 to $N_{cc}$-1.

v. Convert the compressed binary sequence to bytes (8-bit characters).

vi. Append the compressed image data to the image header and save them into the compressed image file, and, in this case, the size of the compressed image file is given as:

$$S_c = H + \left\lceil \frac{m*N}{8} \right\rceil \tag{3.3}$$

| | | |
|---|---|---|
| Where | $S_c$ | size of the compressed image file. |
| | $H$ | size of the image file header. |
| | $N$ | Number of colors. |

Figure (3.3) outlines the procedure of Mode 1 of the Compression Phase of the AQ-HCDC algorithm.

***Procedure of Mode 1 of the Compression Phase of the AQ-HCDC Algorithm.***

```
Set $N_{cc}$=$N_{ce}$   // $N_{cc}$ represents No. of colors that will be presented in the compressed image.
Calculate $m$   // Number of bits required to represent each color, Eqn. (3.2).
Construct the compressed image header containing information that is required for the decompression process.
Read image data and replace each 8-bit color value with $m$-bit binary value equivalent to its sorted sequence number; starting from 0 to $N_{cc}$-1.
Convert the compressed binary sequence to bytes (8-bit characters).
Append the compressed image data to the image header.
Create a compressed image file.
Save the header and the compressed image data into the compressed image file.
```

Figure (3.3). Procedure of Mode 1 of the Compression Phase of the AQ-HCDC algorithm.

b. ***Mode 2***:

i. Start from the most common colors, add counts of colors that have the 1st, 2nd, or 3rd bit differs from that for the most common color, which I shall call the associated colors. This can be expressed as follows:

$$O_i = O_i + O_a = O_i + O_{1st} + O_{2nd} + O_{3rd} \tag{3.4}$$

Where $O_i$ is the initial counts for color *i;* $O_a$ is the total number of associative colors; and $O_{1st}$, $O_{2nd}$, $O_{3rd}$ are counts of color that have their 1st, 2nd, and 3rd bits differs from that for the initial color.

For example, assume the equivalent decimal value of the most common color is 35 (0100011), then the counts of associated colors 34 (0100010), 33 (0100001), and 39 (0100111) are added to the counts of the color 35 and the colors 34, 33, and 39 are replaced by 35. So that if the initial counts of 35 is 10, 34 is 8, 33 is 6, and 39 is 4, then the new counts for 35 is 28.

ii. Reduce $N_{ce}$ by the number of associated colors (*e*) found in the sorted colors, which varies between 0 and 3 as it is explained in Table (3.1).

Table (3.1) - Associated colors.

| *e* | Explanation |
|---|---|
| 0 | No associated color is found |
| 1 | Only one associated color is found (33 or 34 or 39) |
| 2 | Two associated colors are found (33 & 34 or 33 & 39 or 34 & 39) |
| 3 | Three associated colors are found (33 & 34 & 39) |

iii. Discard the associated colors and shift the colors up to fill the gap left-out by merging the associated colors.

iv. Set the new number of colors to $N_{ce}$, where $N_{ce}$=$N_{ce}$-$e$.

v. Repeat Steps (1.b.i) to (1.b.iv) until all possible associated colors are merged.

vi. Sort the new colors list in descending order from the most common to the least common.

vii. Perform Step (1.a).

Figure (3.4) outlines the procedure of Mode 2 of the Compression Phase of the AQ-HCDC algorithm. As it can be seen that Mode 1 (Step (1.a)) provides lossless compression to the input colors, while Mode 2 (Step (1.b)) provides either lossless or lossy depending whether there are associated colors on the colors list or not. However, it is also very important to realize that performing lossy compression (Step (1.b)) can be advantageous only if $N_{ce}$ after merging the associated colors can reduce the value of $m$; otherwise it is not beneficial to go through this Mode.

For example, for $N_{ce}$=11, $m$=4, and due to merging the associated colors, $N_{ce}$ is reduced to 7, which means $m$=3. As a result of that the compression ratio is increased by 4/3. However, if $N_{ce}$ is only reduced to 9, then applying Eqn. (3.2), $m$ remains unchanged ($m$=4). In this case, I loss quality without achieving any increases in the compression ratio and it will better to proceed as lossless compression to the input color list Mode 1. So that I must let the program determine the best compression mode. Furthermore, it can be well recognized that until this moment the compression algorithm is similar to fix-length compression benefiting from the concept of Hamming codes in realizing the associated colors.

***Procedure of Mode 2 of the Compression Phase of the AQ-HCDC Algorithm.***

```
Do ($i$=1; i==$N_{ce}$; $i$++)
Read Color $i$
Do ($j$=$i$+1; $j$==$N_{ce}$; $j$++)
Set $e = 0$
Read Color $j$
Add Counts of Color $j$ to Counts of Color $i$ if it has the 1st, 2nd, or 3rd bit differs.
$e=e+1$
Keep record of the match colors.
$N_{ce} = N_{ce} - e$
Sort the new color list.
Perform the procedure in Figure (3.3).
```

Figure (3.4). Procedure of Mode 2 of the Compression Phase of the AQ-HCDC algorithm.

(2) If $N_{ce}$>16, then

a. Perform Steps (1.b.i) to (1.b.vi) in Mode 2.

b. If $N_{ce}$≤16, then perform Steps (1.a.i) to (1.a.vi) in Mode 1, else continue.

c. Divide the list of colors into two groups as follows:

   i. Group 1 contains $G_1$ colors and $G_1$=$N_{ce}$-16.

   ii. Group 2 contains $G_2$ colors and $G_2$=16.

d. Construct a header for the compressed image containing all information necessary for the decompression process, create a compressed image file, and store the header into the compressed image file. Description of the compressed image header will be given in the next section.

e. Represent each color with $m$-bit binary value equivalent to its sorted sequence number; starting from 0 to $G_1$-1 for Group 1 and from 0 to 15 for Group 2. In this case, $m_1$ is the number of bits for Group 1 and $m_2$ is the number of bits for Group 2. $m_1$ and $m_2$ are calculated using Eqn. (3.2) with $G_1$ and $G_2$ replaces $N_{cc}$ for $m_1$ and $m_2$, respectively. $m_2$ is usually 4-bit, while $m_1$ could be equal to or less than 4-bit depending on the number of colors in Group 1 ($G_1$).

f. Replace each 8-bit color in the uncompressed image with its equivalent $m$-bit color (either $m_1$ or $m_2$), and then convert the compressed binary sequence to bytes (8-bit characters). Each color in Group 1 is represented by $m_1$-bit binary sequence preceded by 0, while colors in Group 2 are represented by $m_2$-bit binary sequence preceded by 1.

g. Append the compressed image data to the image header and store them into the compressed image file, and, in this case, the size of the compressed image file is given as:

$$S_c = H + m_1 \sum_{j=1}^{G_1} O_j + m_2 \sum_{j=1}^{G_2=16} O_j \qquad (3.4)$$

| | | |
|---|---|---|
| Where | $S_c$ | size of the compressed image file. |
| | $H$ | size of the image file header. |
| | $m_1$ | number of bits required to represent the colors in the group1. |
| | $G_1$ | Number of colors inside group1. |
| | $M_2$ | number of bits required to represent the colors in the group2. |
| | $G_2$ | Number of colors inside group2. |
| | $O_j$ | counts for color $j$. |

***Procedure for the Compression Phase when $N_{ce}$>16.***

```
Perform the procedure in Figure (3.4) except the last step.
If ($N_{ce}\leq 16$)
Perform the procedure in Figure (3.3)
Else
Divide the list of colors into two groups:
Group 1 contains $G_1$ colors and $G_1=N_{ce}-16$
Group 2 contains $G_2$ colors and $G_2=16$
Calculate $m_1$
Calculate $m_2$
Construct the compressed image header containing information for the decompression process.
Read image data and replace each 8-bit color value with either $m_1$ or $m_2$ -bit binary value equivalent to its sorted sequence number; starting from 0 to $G_1$-1 for Group 1 or from 0 to 15 for Group 2.
Convert the compressed binary sequence to bytes (8-bit characters).
Append the compressed image data to the image header.
Create a compressed image file.
Save the header and the compressed image data into the compressed image file.
```

Figure (3.5) outline the procedure for the Compression Phase when $N_{ce}$>16.

## 3.2 The Compressed File Header

The AQ-HCDC algorithm compressed file header contains all additional information that is required by the decompression algorithm. It consists of three main fields; these are:

(a) HCDC field

(b) Colors field

(c) Original file header field

These three fields will be followed by the image compressed data. In what follows a brief description is given for the fields of the compressed file header.

### (a) ***HCDC field***

The HCDC field of the AQ-HCDC algorithm is designed to keep the same structure of original HCDC field for consistency purposes. It is an 8-byte field encloses information related to the algorithm, such as: algorithm name (HCDC), algorithm version (*V*), number of symbols (colors) in the original image, coding format (*F*), number of compression loops (*k*). It can be seen that some of the data is redundant, however, they have insignificant effect due their very small (negligible size).

The original HCDC algorithm (Al-Bahadili H. , 2008) was designated as Version-0 (i.e., *V* is set to 0), the HCDC(*k*) scheme (Al-Bahadili & Rababa'a, 2010) is designated as Version-1 (i.e., *V* is set to 1), while the AQ-HCDC algorithm is designated as Version 2. The coding format (*F*) is set to 0 for ASCII coding, 1 for Huffman coding, 2 for adaptive coding, etc. Table (3.2) lists the components of this field and their description.

Table (3.2)

Components of the HCDC field of the AQ-HCDC compressed file header.

| Components | Length (Byte) | Description |
|---|---|---|
| HCDC | 4 | Name of the compression algorithm. |
| $V$ | 1 | Version of the HCDC algorithm. |
| $N_c$ | 1 | Number of symbols (colors) within the original text (image) file. |
| $F$ | 1 | Coding format (0 for ASCII coding, 1 for Huffman coding, 2 for adaptive coding, etc.) |
| $K$ | 1 | The number of compression loops. |

**(b)** ***Colors field***

The colors field is designed to include minimum information, it encloses information related to: number of colors in Group 1 ($G_1$), number of colors in Group 2 ($G_2$), and values ($V_i$) of colors sorted from the most common to the least common. Table (3.3) lists the components of the colors field, their wordlength, and brief description. The values of $m_1$ and $m_2$ can be calculated using Eqn. (3.2) for Group 1 and Group 2, and also $N_{cc}$ can be calculates as $N_{cc}= G_1+G_2$.

Table (3.3)

Components of the colors field of the AQ-HCDC compressed file header.

| Components | Length (Byte) | Description |
|---|---|---|
| $G_1$ | 1 | Number of colors in Group 1 ($0 \leq G_1 \leq N_{ce}-16$). |
| $G_2$ | 1 | Number of colors in Group 2 ($0 \leq G_2 \leq 16$). |
| $V_i$ | $N_{cc}$ | The colors values in the compressed image, where $i$=1 to $N_{cc}$, and $N_{cc}=G_1+G_2$. |

**(c)** ***<u>Original File Header Field</u>***

This field contains a copy of the header of the original image, e.g., the 54-Byte header of the uncompressed image. This field cannot be compressed as the image data, because the AQ-HCDC algorithm may perform lossy compression, which means it will be difficult to retrieve the original header of the image.

Figure (3.6) shows the structure of the AQ-HCDC compressed file header. It can be seen that the total length has a maximum size of 96 Bytes, and generally it is given by:

$$H = 64 + N_{cc} \tag{3.6}$$

| | | |
|---|---|---|
| Where | $H$ | size of the image file header. |
| | $N_{cc}$ | number of colors that will be presented in the compressed image. |

The 64 Byte in Eqn. (3.6) is the sum of 8-Byte for the HCDC field, 2-Byte for the fix-part of the color field, and 54-Byte the original BMP image header.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | C | D | C | $V$ | $N_c$ | $F$ | $K$ | $G_1$ | $G_2$ | $V_i$, $V_i$, $V_i$, …, $V_{Ncc}$ | *Original Image Header* | *Compressed Image Data* |

Figure (3.6). The structure of the AQ-HCDC compressed file header.

## 3.3 The Decompression Process of the AQ-HCDC Algorithm

The decompression process of the AQ-HCDC algorithm is very simple and straightforward and it is accomplished much faster than the compression process, therefore, the algorithm is classified as an asymmetric compression algorithm where the there is a difference in the processing time between the compression and the decompression processes.

The decompression algorithm of the AQ-HCDC algorithm can be divided into two main phases. In the first phase, the algorithm reads in the header data and prepares the list of the colors values. In particular, the decompression algorithm (decompressor) starts by reading the compressed image header. After approving the algorithm's name and version, it reads the values of $G_1$ and $G_2$ and computes $m_1$, $m_2$, and $N_{cc}$. Then it reads the sorted colors values from $V_i$ to $V_{Ncc}$.

In the second phase, which is the core of the decompressor, the algorithm reads in the image data based on the total number of compressed colors in the image header ($N_{cc}$). If $N_{cc}$≤16, then it reads in the image data as $m_1$-bit colors. For each $m_1$-bit, it calculates its equivalent decimal number ($D$) between 0 and $2^{m1}$-1, then it set the uncompressed color to color value in file header with index number $D$+1, i.e., the uncompressed color value is $V_i(D+1)$. Setting the index to $D$+1 because the colors are numbered from 1 to $N_{cc}$, while the decimal values starts from 0.

If $N_{cc}$>16, the decompressor starts by reading in 1-bit, if it is 0, it reads in $m_1$-bit, calculates $D$, and then set the color to the color value $V_{D+1}$ in Group 1; otherwise (1-bit value is 1), it reads in $m_2$-bit, calculate $D$, and then set the color to the color value $V_{D+1}$ in

Group 2. This process continues until readings in all image data. Figure (3.7) outlines the procedure for the AQ-HCDC algorithm decompressor.

## 3.4 Analytical Analysis of Performance of the AQ-HCDC Algorithm

This section presents the analytical analysis of the performance of the AQ-HCDC algorithm. From the description of the algorithm in previous sections, it can be recognized that the algorithm replaces an 8-bit original color to a shorter representation depending on the original image colors values. In worst case the algorithm replaces 8-bit colors by 5-bit colors; 1-bit as a prefix and 4-bit represents the sequence number of the color within its group. Therefore, the minimum compression ratio that can be achieved by the algorithm is 1.6, which comes from dividing 8 by 5, and assuming a negligible header size.

The compression ratio can be increased to 8 for images with 2 colors, where one color will be replaced by 0 while the other color by 1. Similarly, it is increased to 4, 2.67, and 2 for images with 4, 8, and 16 colors or approximated to have these numbers of colors inside the image. However, in all cases, due to the nature of approximating the colors, the AQ-HCDC algorithm can achieve better compressed image quality than other algorithms as I shall demonstrate in the next chapter.

***Procedure for the AQ-HCDC algorithm decompressor.***

```
Read in the image header.
Extract the values of $G_1$ and $G_2$.
Calculate $m_1$, $m_2$ (using Eqn. (3.2)) and $N_{cc}=G_1+G_2$.
Reads in the colors values ($V_i$ for $i=1$ to $N_{cc}$).
If ($Ncc \le 16$) Then
Do
Reads in $m_1$-bit
Calculate its equivalent decimal value ($D$)
Find the associate color from the colors list with index $D+1$ from Group 1.
```

```
Set this value to the uncompressed image data.
Loop until end of image data
Else
Do
Reads in 1-bit (B)
If (B=0) then
Reads in $m_1$-bit
Calculate its equivalent decimal value (D)
Find the associate color from the colors list with index D+1 from Group 1.
Set this value to the uncompressed image data.
Else
Reads in $m_2$-bit
Calculate its equivalent decimal value (D)
Find the associate color from the colors list with index D+1 from Group 2.
Set this value to the uncompressed image data.
End If
Loop until end of image data
End If
```

Figure (3.7). Procedure for the AQ-HCDC algorithm decompressor.

# Chapter Four

# Experimental Results and Discussions

The Adjustable-Quality Hamming Codes based Data Compression (AQ-HCDC) algorithm is implemented using VB.Net programming language. The resultant code allows a wide range of investigations and experiments to be performed on a wide range of image compression applications. However, in this thesis, I only present a limited number of image compression results to demonstrate the performance of the algorithm in providing a high performance adjustable quality image compression, especially useful for mobile and Internet applications.

In order to demonstrate the performance of the algorithm, I carried out two types of experiments; one experiment for large-size image compression and the second experiment for small-size image compression. The images used in these experiments are widely-used by many researchers as a test images due to their standard features. In particular, I have selected six large-size images, namely, AirPlane, Baboon, CornField, Flowers, Girl, and Monarch, which are shown in Figure (4.1). Small-size images are created by reducing the dimensions of the images. The dimensions and sizes of the large-size images are given in Tables (4.1) and shown in Figure (4.1), while those for small-size images are given in Table (4.2) and shown in Figure (4.2). The images are listed according to their sizes from the smaller to larger size.

Table (4.1)
Large-Size test images.

| # | Image | Dimensions (Pixel) | Size (Byte) |
|---|---|---|---|
| 1 | Flowers | 500x362 | 543,054 |
| 2 | Baboon | 500x480 | 720,054 |
| 3 | CornField | 512x480 | 737,334 |
| 4 | AirPlane | 512x512 | 786,486 |
| 5 | Monarch | 768x512 | 1,179,702 |
| 6 | Girl | 720x576 | 1,244,214 |

Table (4.2)
Small-Size test images.

| # | Image | Dimensions (Pixel) | Size (Byte) |
|---|---|---|---|
| 1 | AirPlane | 239x240 | 172,854 |
| 2 | Baboon | 250x240 | 180,534 |
| 3 | CornField | 256x240 | 184,374 |
| 4 | Monarch | 320x213 | 204,534 |
| 5 | Girl | 300x240 | 216,054 |
| 6 | Flowers | 320x231 | 221,814 |

Flowers.BMP

Baboon.BMP

CornField.BMP

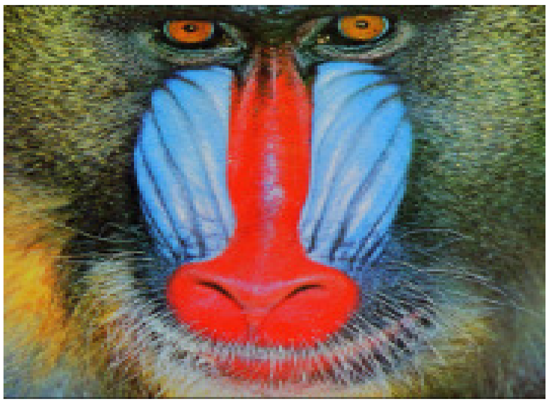
AirPlane.bmp

Monarch.BMP

Girl.BMP

Figure (4.1). The uncompressed large-size test images (BMP format).

AirPlane.bmp



Baboon.BMP

CornField.BMP

Monarch.BMP

Girl.BMP

Flowers.BMP

Figure (4.2). The uncompressed small-size test images (BMP format).

The performance of the AQ-HCDC algorithm is evaluated in terms of three performance measures, namely, Compression ratio (*C*), Mean Square Error (MSE), and Peak Signal to Noise Ratio (PSNR), which are defined in Chapter 1. At this stage, it is important to indicate that little efforts have been taken to optimize the runtime of the compression/decompression code, therefore will not present timing results. However, in general, through our investigations, I have found that the AQ-HCDC algorithm demonstrate an asymmetric timing behavior, where the compression processing time is higher than the time required for decompression.

In both experiments, the performance of the AQ-HCDC algorithm is compared against the performance of a number of standard lossless and lossy compressed image formats, and lossless compression tools. The image formats and tools that are considered in this thesis are:

- Uncompressed
    - Bitmap (BMP)
- Lossless compressed image formats:
    - Graphics Interchange Format (GIF)
    - Portable Network Graphics (PNG)
- Lossy compressed image formats:
    - Joint Photographic Experts Group (JPEG)
- Lossless tools
    - Windows Roshal ARchive (WinRAR)
    - Phil Katz's ZIP (ZIP)

A brief description of these image formats and tools were given in Chapter 1. It is also important to mention that the AQ-HCDC compressed file header is taken into considerations in both experiments.

## 4.1 Experiment #1: Compressing Large-Size Images

This experiment evaluates the performance measures (*C*, MSE, and PNSR) achieved by the AQ-HCDC algorithm for compression the six large-size images that are described in Table (4.1). Furthermore, it compares these performance measures for the AQ-HCDC algorithm against those for the standard lossless and lossy compressed image formats, and lossless compression tools that are described above. The results for the sizes of images before and after compression, *C*, MSE, and PNSR are listed in Tables (4.3), (4.4), (4.5), and (4.6), respectively, and also plotted in Figures (4.3) to (4.6).

Comparing the compression ratio between the AQ-HCDC algorithm and the compressed image formats and lossless compression tools, it can be seen in Tables (4.3) and (4.4), and Figures (4.3) and (4.4) that:

- The AQ-HCDC algorithm achieves a compression ratio of ≈1.6. This is because the compressed colors span over two groups, each of 16 colors; so that each uncompressed 8-bit color is expressed with only 5-bit (1-bit identifying the group number and 4-bit identifying the sequence of the color within the group). The colors span over two complete group because the images selected have a continuous color distribution, where after discarding the color's Least Significant Bit (LSB) and shift each color's bits one place to the right, all images are turn-out to have all possible color values (i.e., 0-127). It also important to recognize that the size of the compressed image header is 96 (Eqn. 3.6), and this value is negligible as compared to the images sizes.

- The compression ratio achieved by the AQ-HCDC algorithm is higher than (≈1.6) that achieved by the lossless PNG (≈1.2 except for Baboon image where it is 2.6) and less than that achieved by the lossy JPEG (from 4.2 to 9.1). This is at the cost of some reduction/improvement in the image quality, where the AQ-HCDC algorithm provides better image quality than that produced by the JPEG format and of course less image quality in comparison with PNG, which I will discuss next.

- The AQ-HCDC algorithm provides higher compression ratio than ZIP and competitive performance to WinRAR for all images except for Baboon, where both tools are higher due to the image structure and characters frequencies. Although, this doesn't look a fare comparison as these tools are lossless and multi-algorithms tools, while the AQ-HCDC algorithm is behaving likes lossy but it is a single algorithm. However, it is very useful to demonstrate the compression power of the AQ-HCDC algorithm.

Comparing the compressed image quality between the AQ-HCDC algorithm and the compressed image formats and lossless compression tools, it can be seen in Tables (4.5) and (4.6), and Figures (4.5) and (4.6) that:

- The AQ-HCDC algorithm provides better image quality that the lossy JPEG algorithm, which is also meeting the standards for acceptable image quality. It provides a PNSR of more than 30 dB, where the typical values for the PSNR in lossy image and video compression are between 30 and 50 dB, where higher is better (Barni, 2006); and the acceptable values for wireless transmission quality loss are considered to be about 20 dB to 25 dB (Thomos et al., 2006). However, these values are than that for the JPEG compression format.

Due to the wide range of colors values in the test images, the AQ-HCDC algorithm provides a compression ratio of 1.6, which represents the minimum it can achieve. At the same time, it can be easily realized that for such images with wide colors range the compression ratio has no effect on the compressed image quality, and in fact the compressed image quality improves (increases) as the compression ratio increase above 1.6. Because, for example, with reference to our discussion in Chapter 3, if the total number of compressed colors in the last group is less than 16 colors and they require less than 4-bit to represent them, then the compression ratio increases, and at the same time this means that I made less color approximation, which in turn reduces MSE and consequently increases PNSR.

Table (4.3) – Experiment #1 – Large-Size Images

Comparing the sizes of the uncompressed and compressed images for various compressed image formats and compression tools.

| Image | Un-compressed Image Size (BMP) (Byte) | Lossless Compressed Formats | | Lossy Compressed Format | Lossless Compression Tools | | AQ-HCDC |
|---|---|---|---|---|---|---|---|
| | | GIF | PNG | JPEG | ZIP | WinRAR | |
| Flowers | 543,054 | 101,431 | 498,897 | 91,286 | 442,875 | 338,807 | 339505 |
| Baboon | 720,054 | 161,823 | 281,327 | 167,834 | 215,198 | 226,663 | 450130 |
| Corn Field | 737,334 | 110,699 | 654,745 | 103,289 | 581,545 | 434,208 | 460930 |
| Air Plane | 786,486 | 96,638 | 639,736 | 89,900 | 561,678 | 416,430 | 491650 |
| Monarch | 1,179,702 | 190,991 | 946,312 | 130,164 | 812,905 | 529,232 | 737410 |
| Girl | 1,244,214 | 193,615 | 1,041,310 | 144,626 | 921,290 | 538,736 | 777730 |

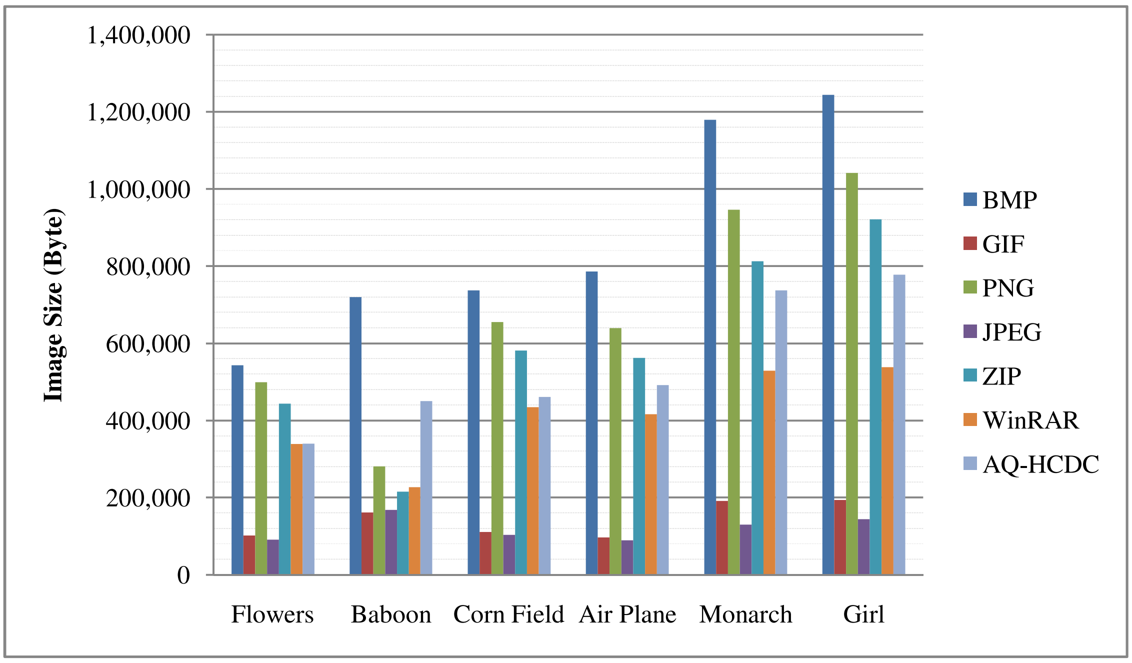


Figure (4.3). Experiment #1: Comparing the sizes of the uncompressed and compressed images for various compressed image formats and compression tools.

Table (4.4) – Experiment #1 – Large-Size Images

Comparing *C* for various compressed image formats and compression tools.

| Image | Un-compressed Image Size (BMP) (Byte) | Lossless Compressed Formats | | Lossy Compressed Format | Lossless Compression Tools | | AQ-HCDC |
|---|---|---|---|---|---|---|---|
| | | GIF | PNG | JPEG | ZIP | WinRAR | |
| Flowers | 543,054 | 5.354 | 1.089 | 5.949 | 1.226 | 1.603 | 1.600 |
| Baboon | 720,054 | 4.450 | 2.559 | 4.290 | 3.346 | 3.177 | 1.600 |
| Corn Field | 737,334 | 6.661 | 1.126 | 7.139 | 1.268 | 1.698 | 1.600 |
| Air Plane | 786,486 | 8.138 | 1.229 | 8.748 | 1.400 | 1.889 | 1.600 |
| Monarch | 1,179,702 | 6.177 | 1.247 | 9.063 | 1.451 | 2.229 | 1.600 |
| Girl | 1,244,214 | 6.426 | 1.195 | 8.603 | 1.351 | 2.310 | 1.600 |

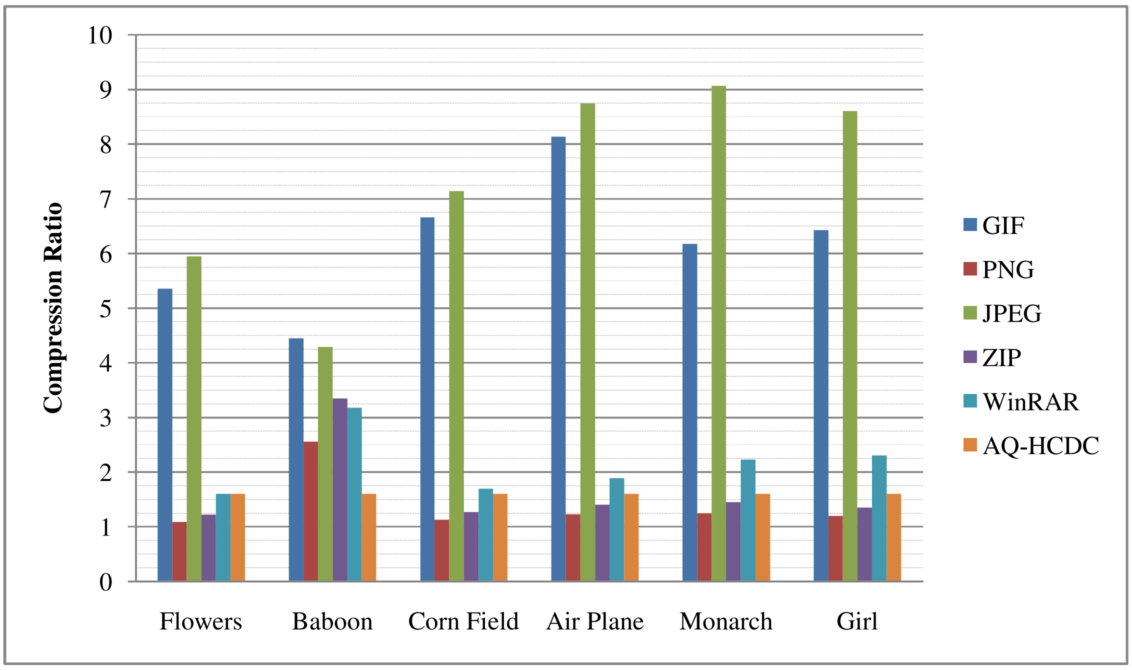


Figure (4.4). Experiment #1: Comparing *C* for various compressed image formats and compression tools.

Table (4.5) – Experiment #1 – Large-Size Images

Comparing MSE for various compressed image formats and compression tools.

| Image | Un-compressed Image Size (BMP) (Byte) | Lossless Compressed Formats | | Lossy Compressed Format | Lossless Compression Tools | | AQ-HCDC |
|---|---|---|---|---|---|---|---|
| | | GIF | PNG | JPEG | ZIP | WinRAR | |
| Flowers | 543,054 | 1131.28 | 0 | 102.23 | 0 | 0 | 30.67 |
| Baboon | 720,054 | 901.85 | 0 | 352.16 | 0 | 0 | 26.33 |
| Corn Field | 737,334 | 972.24 | 0 | 63.03 | 0 | 0 | 42.01 |
| Air Plane | 786,486 | 806.28 | 0 | 65.61 | 0 | 0 | 34.43 |
| Monarch | 1,179,702 | 1055.21 | 0 | 18.31 | 0 | 0 | 35.78 |
| Girl | 1,244,214 | 994.02 | 0 | 31.56 | 0 | 0 | 32.48 |

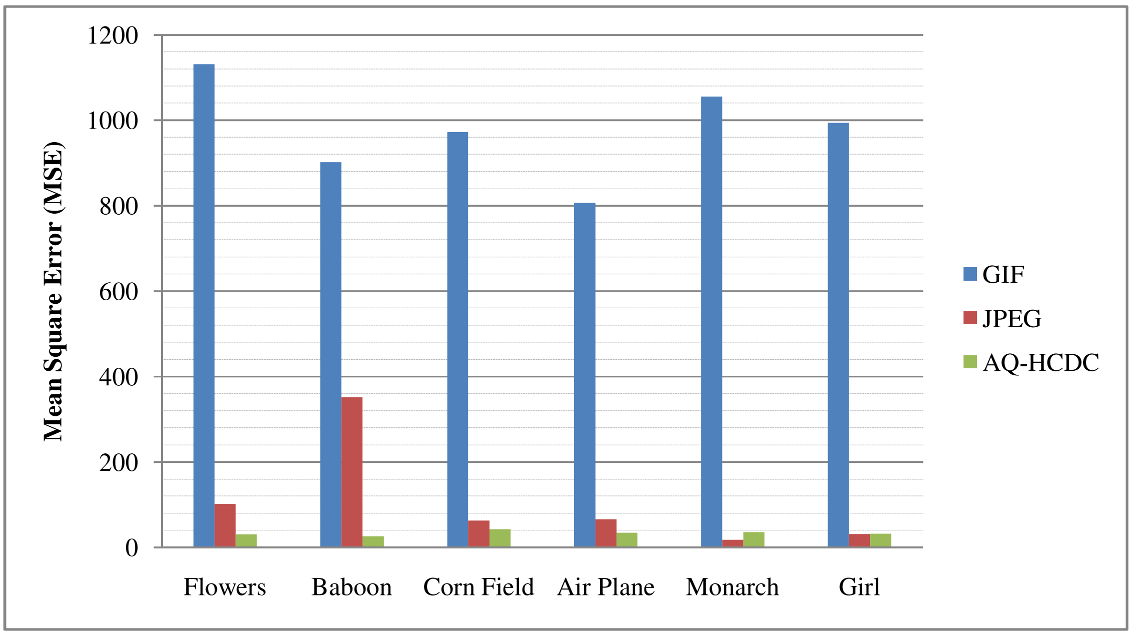


Figure (4.5). Experiment #1: Comparing MSE for various compressed image formats.

Table (4.6) – Experiment #1 – Large-Size Images

Comparing PNSR for various compressed image formats and compression tools.

| Image | Un-compressed Image Size (BMP) (Byte) | Lossless Compressed Formats | | Lossy Compressed Format | Lossless Compression Tools | | AQ-HCDC |
|---|---|---|---|---|---|---|---|
| | | GIF | PNG | JPEG | ZIP | WinRAR | |
| Flowers | 543,054 | 17.56 | ∞ | 28.03 | ∞ | ∞ | 33.26 |
| Baboon | 720,054 | 18.58 | ∞ | 22.66 | ∞ | ∞ | 33.93 |
| Corn Field | 737,334 | 18.25 | ∞ | 30.14 | ∞ | ∞ | 31.90 |
| Air Plane | 786,486 | 19.07 | ∞ | 29.96 | ∞ | ∞ | 32.76 |
| Monarch | 1,179,702 | 17.90 | ∞ | 35.50 | ∞ | ∞ | 32.60 |
| Girl | 1,244,214 | 18.16 | ∞ | 33.13 | ∞ | ∞ | 33.06 |

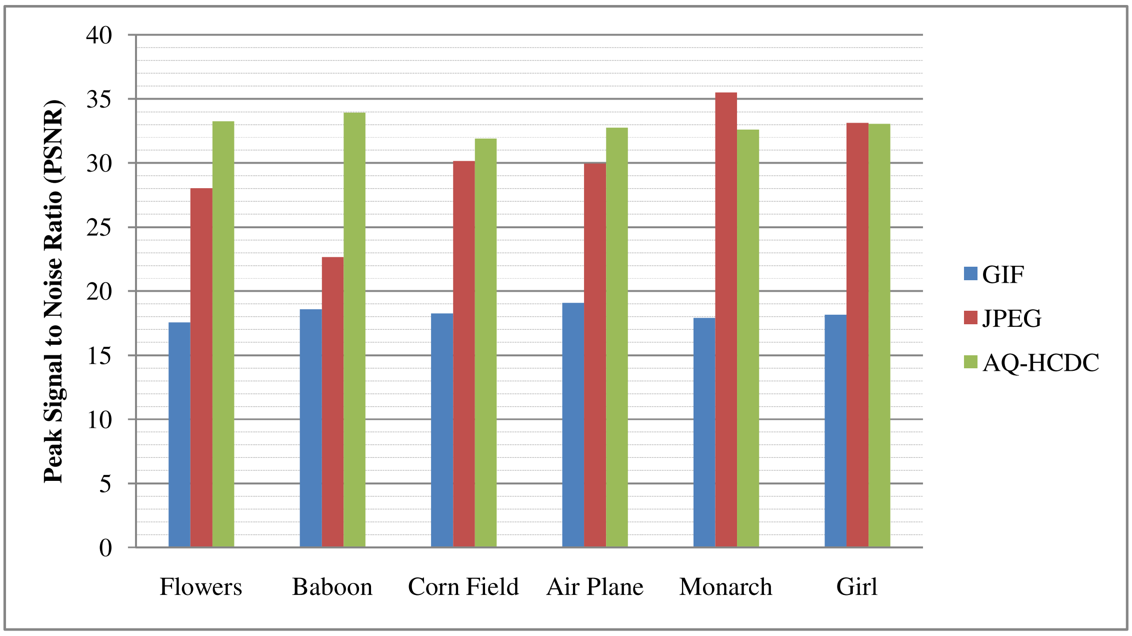


Figure (4.6). Experiment #1: Comparing PNSR for various compressed image formats.

## 4.2 Experiment #2: Compressing Small-Size Images

This experiment evaluates the performance measures (*C*, MSE, and PNSR) achieved by the AQ-HCDC algorithm for compression the six small-size images that are described in Table (4.2). Furthermore, it compares these performance measures for the AQ-HCDC algorithm against those for the standard lossless and lossy compressed image formats, and lossless compression tools that are described above. The results for the sizes of images before and after compression, *C*, MSE, and PNSR are listed in Tables (4.7), (4.8), (4.9), and (4.10), respectively, and also plotted in Figures (4.7) to (4.10).

Comparing the compression ratio between the AQ-HCDC algorithm and the compressed image formats and lossless compression tools, it can be seen in Tables (4.7) and (4.8), and Figures (4.7) and (4.8) that:

- The AQ-HCDC algorithm achieves as in the previous experiment a compression ratio of ≈1.6. This is because of the same reason described in previous section for large-size images, where the compressed colors span over two groups, each of 16 colors; so that each uncompressed 8-bit color is expressed with only 5-bit (1-bit identifying the group number and 4-bit identifying the sequence of the color within the group). The colors span over two complete group because the images selected have a continuous color distribution, where after discarding the color's LSB and shift each color's bits one place to the right, all images are turn-out to have all possible color values (i.e., 0-127). It also important to recognize that the size of the compressed image header is the same as well and it is equal to 96 Byte (Eqn. 3.6). However, in this case it shows very insignificant effect as the compression ratio is calculated as 1.599≈1.6.

- The compression ratio achieved by the AQ-HCDC algorithm is higher than (≈1.6) that achieved by the lossless PNG (≈1.1) and less than that achieved by the lossy JPEG (from 4.4 to 7.0). This is at the cost of some reduction/improvement in the image quality, where the AQ-HCDC algorithm provides better image quality than that produced by the JPEG format and of course less image quality in comparison with PNG as I shall discuss next.

- The AQ-HCDC algorithm provides higher compression ratio than ZIP and competitive performance to WinRAR.  Although, this doesn’t look a fare comparison as these tools are lossless and multi-algorithms tools, while the AQ-HCDC algorithm is behaving likes lossy but it is a single algorithm. But, it is very useful to demonstrate the power of the AQ-HCDC algorithm.

Comparing the compressed image quality between the AQ-HCDC algorithm and the compressed image formats and lossless compression tools, it can be seen in Tables (4.9) and (4.10), and Figures (4.9) and (4.10) that:

- The AQ-HCDC algorithm provides almost the same image quality as for the lossy JPEG algorithm, which is around 30 dB. However, for many images the AQ-HCDC algorithm provides little bit higher image quality than JPEG. Both algorithms meet the standards for acceptable compressed image quality, where the typical values for the PSNR in lossy image and video compression are between 30 and 50 dB, where higher is better (Barni, 2006).

Table (4.7) – Experiment #2 – Small-Size Images

Comparing the sizes of the uncompressed and compressed images for various compressed image formats and compression tools.

| Image | Un-compressed Image Size (BMP) (Byte) | Lossless Compressed Formats | | Lossy Compressed Format | Lossless Compression Tools | | AQ-HCDC |
|---|---|---|---|---|---|---|---|
| | | GIF | PNG | JPEG | ZIP | WinRAR | |
| AirPlane | 172,854 | 23,211 | 138,658 | 24,654 | 126,024 | 93,639 | 108130 |
| Baboon | 180,534 | 38,235 | 178,480 | 40,361 | 161,800 | 154,659 | 112930 |
| CornField | 184,374 | 30,894 | 164,897 | 31,582 | 147,950 | 121,050 | 115330 |
| Monarch | 204,534 | 35,915 | 192,306 | 30,524 | 172,507 | 115,270 | 127930 |
| Girl | 216,054 | 36,733 | 202,636 | 33,626 | 183.134 | 121,172 | 135130 |
| Flowers | 221,814 | 43,570 | 207,059 | 42,204 | 186,312 | 141,813 | 138730 |

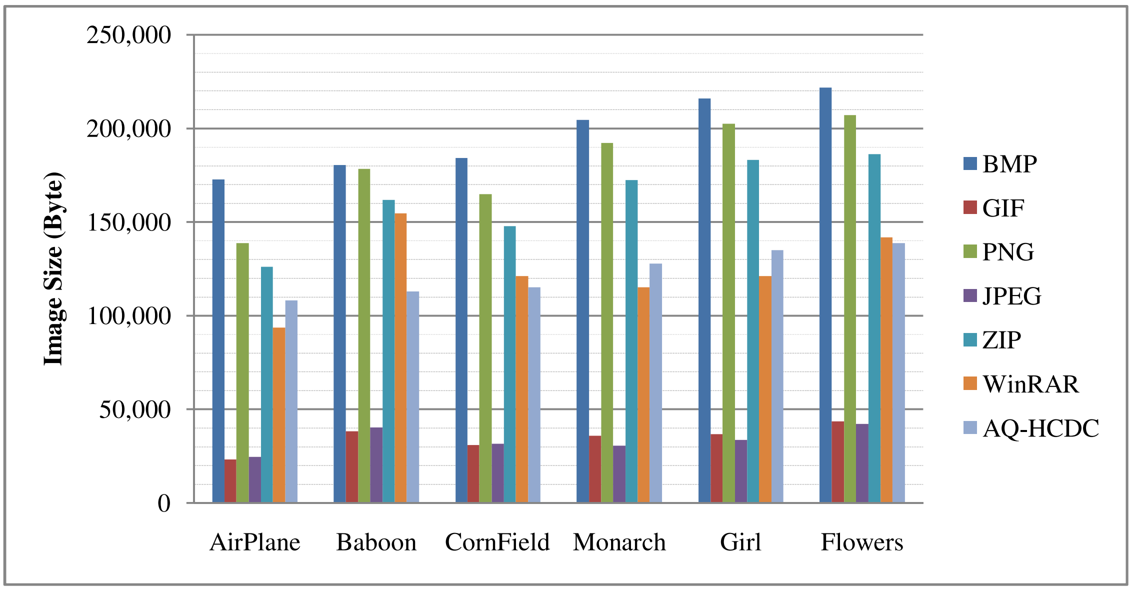


Figure (4.7). Experiment #2: Comparing the sizes of the uncompressed and compressed images for various compressed image formats and compression tools.

Table (4.8) – Experiment #2 – Small-Size Images

Comparing *C* for various compressed image formats and compression tools.

| Image | Un-compressed Image Size (BMP) (Byte) | Lossless Compressed Formats | | Lossy Compressed Format | Lossless Compression Tools | | AQ-HCDC |
|---|---|---|---|---|---|---|---|
| | | GIF | PNG | JPEG | ZIP | WinRAR | |
| AirPlane | 172,854 | 7.447 | 1.247 | 7.011 | 1.372 | 1.846 | 1.599 |
| Baboon | 180,534 | 4.722 | 1.012 | 4.473 | 1.116 | 1.167 | 1.599 |
| CornField | 184,374 | 5.968 | 1.118 | 5.838 | 1.246 | 1.523 | 1.599 |
| Monarch | 204,534 | 5.695 | 1.064 | 6.701 | 1.186 | 1.774 | 1.599 |
| Girl | 216,054 | 5.882 | 1.066 | 6.425 | 1.180 | 1.783 | 1.599 |
| Flowers | 221,814 | 5.091 | 1.071 | 5.256 | 1.191 | 1.564 | 1.599 |

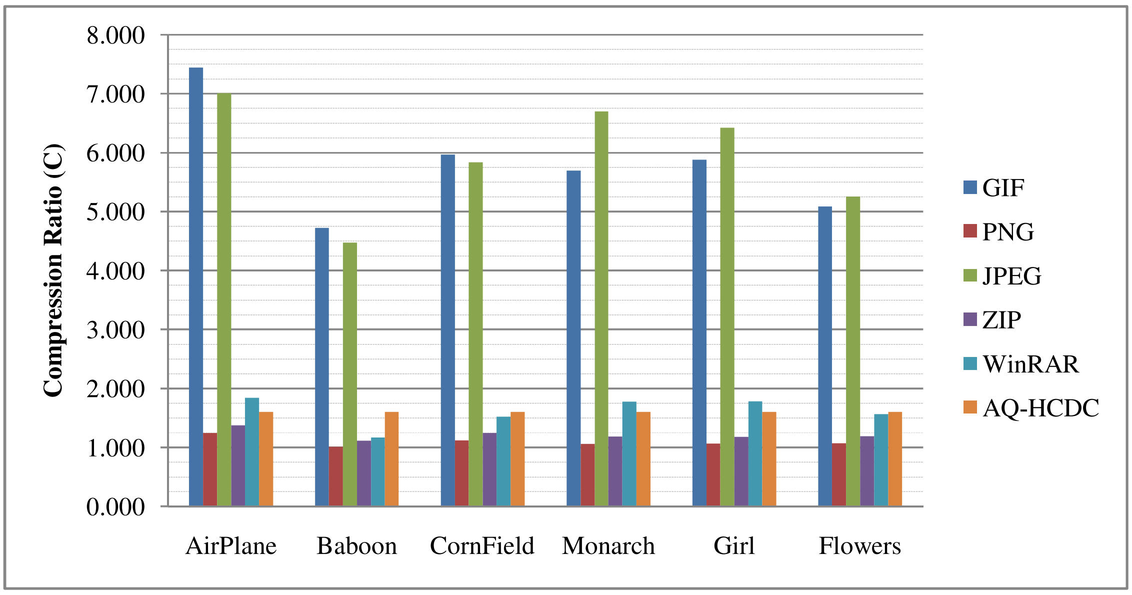


Figure (4.8). Experiment #2: Comparing *C* for various compressed image formats and compression tools.

Table (4.9) – Experiment #2 – Small-Size Images

Comparing MSE for various compressed image formats and compression tools.

| Image | Un-compressed Image Size (BMP) (Byte) | Lossless Compressed Formats | | Lossy Compressed Format | Lossless Compression Tools | | AQ-HCDC |
|---|---|---|---|---|---|---|---|
| | | GIF | PNG | JPEG | ZIP | WinRAR | |
| AirPlane | 172,854 | 782.46 | 0 | 59.17 | 0 | 0 | 34.12 |
| Baboon | 180,534 | 900.44 | 0 | 188.47 | 0 | 0 | 35.62 |
| CornField | 184,374 | 963.78 | 0 | 79.61 | 0 | 0 | 35.93 |
| Monarch | 204,534 | 1010.81 | 0 | 48.70 | 0 | 0 | 38.23 |
| Girl | 216,054 | 960.44 | 0 | 55.79 | 0 | 0 | 34.20 |
| Flowers | 221,814 | 1001.16 | 0 | 137.83 | 0 | 0 | 32.26 |

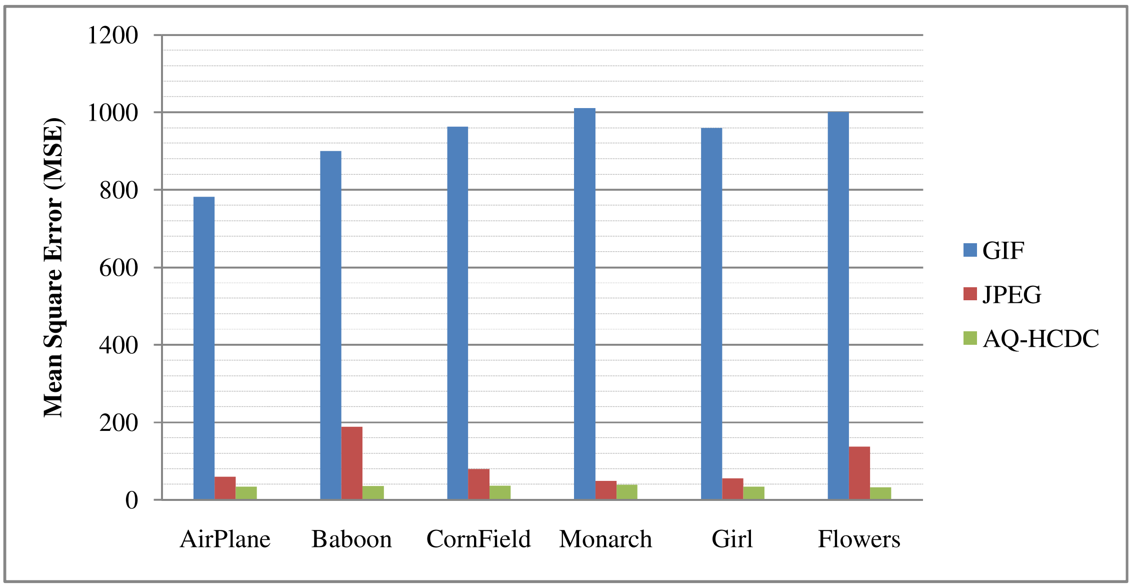


Figure (4.9). Experiment #2: Comparing MSE for various compressed image formats.

Table (4.10) – Experiment #2 – Small-Size Images

Comparing PNSR for various compressed image formats and compression tools.

| Image | Un-compressed Image Size (BMP) (Byte) | Lossless Compressed Formats | | Lossy Compressed Format | Lossless Compression Tools | | AQ-HCDC |
|---|---|---|---|---|---|---|---|
| | | GIF | PNG | JPEG | ZIP | WinRAR | |
| AirPlane | 172,854 | 19.20 | ∞ | 30.41 | ∞ | ∞ | 32.80 |
| Baboon | 180,534 | 18.59 | ∞ | 25.38 | ∞ | ∞ | 32.61 |
| CornField | 184,374 | 18.29 | ∞ | 29.12 | ∞ | ∞ | 32.58 |
| Monarch | 204,534 | 18.13 | ∞ | 26.74 | ∞ | ∞ | 33.04 |
| Girl | 216,054 | 18.30 | ∞ | 30.67 | ∞ | ∞ | 32.79 |
| Flowers | 221,814 | 18.09 | ∞ | 31.26 | ∞ | ∞ | 32.30 |

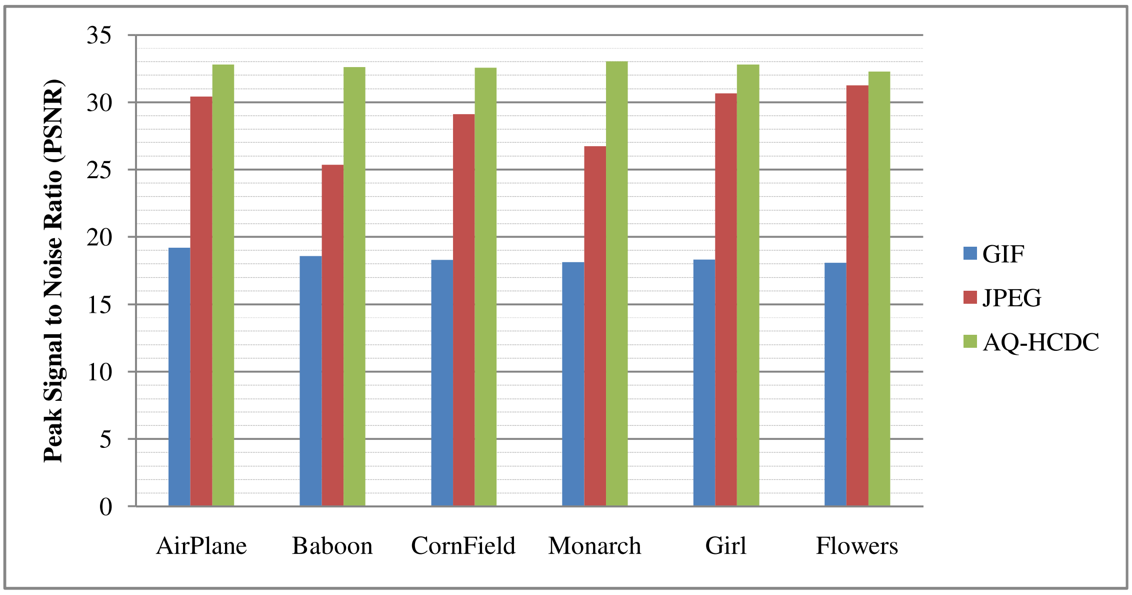


Figure (4.10). Experiment #2: Comparing PNSR for various compressed image formats.

## 4.3 Further Discussion

It can be clearly recognized from the results presented in the previous two sections that there are some common points on large-size and small-size images compression; these are:

(1) Despite the fact that GIF is classified as lossless algorithm, it always provides the highest compression ratio with very low image quality; and this is because GIF handle the image as an 8-bit color, while it is originally as 24-bit color.

(2) The PNG, as 24-bit lossless compression image format, always presents 0 error and consequently undefined PNSR (∞). The same for ZIP and WinRAR.

(3) The WinRAR always provides better performance in terms of compression ratio than the ZIP.

(4) The AQ-HCDC almost provides the same performance in terms of C, MSE, and PNSR for both large-size and small-size images compression, while GIF and JPEG provide fluctuated performance.

(5) The AQ-HCDC algorithm and JPEG always provides a PNSR of more than 30 dB, while GIF provides almost half of this value (∞17 dB) for both large-size and small-size image compression.

Our early investigations on using ZIP and WinRAR as post compressor for GIF, PNG, JPEG, and AQ-HCDC image formats indicate that the AQ-HCDC format can provide the highest compression ratio among them all. However, this needs further investigations and it is left for future research.

# Chapter Five

# Conclusions and Recommendations for Future Work

## 5.1 Conclusions

This Thesis develops a modified version of the lossless bit-level Hamming Codes based Data Compression (HCDC) algorithm to be used for standstill image compression. The modified version of the HCDC algorithm is characterized as an image-based adjustable-quality lossless/lossy algorithm; therefore, it is referred to as Adjustable-Quality HCDC and abbreviated as AQ-HCDC algorithm. It can be used efficiently and effectively for Multimedia Messaging Service (MMS) and Internet applications on mobile networks to provide high performance, where it can provide high compression ratio while maintaining image quality, which means reduces bandwidth requirement and power consumption during MMS and Internet applications.

In order to evaluate the performance of the AQ-HCDC algorithm, two types of experiments were carried-out; the first experiment evaluates the performance of the algorithm for large-size images compression, while the second for small-size images compression. The images used in these experiments are widely-used as a test images by many researchers due to their standard features. In particular, six large-size images, namely, AirPlane, Baboon, CornField, Flowers, Girl, and Monarch, were selected, and then an equivalent set of small-size images were created by reducing the dimensions of the these images.

The performance of the AQ-HCDC algorithm was evaluated in terms of three performance measures, namely, Compression ratio (*C*), Mean Square Error (MSE), and Peak Signal to Noise Ratio (PSNR). In all experiments, the performance of the AQ-HCDC algorithm is compared against the performance of a number of standard lossless compressed image formats (e.g., GIF and PNG), lossy image format (e.g., JPEG), and lossless compression tools (e.g., ZIP and WinRAR).

The main conclusions of this thesis are:

(1) The AQ-HCDC algorithm achieves a compression ratio of ≈1.6. This is because the compressed colors span over two groups, each of 16 colors; so that each uncompressed 8-bit color is compressed to only 5-bit (1-bit identifying the group number and 4-bit identifying the sequence of the color within the group). The colors span over two complete group because the images selected have a continuous color distribution, where after discarding the color's Least Significant Bit (LSB) and shift each color's bits one place to the right, all images are turn-out to have all possible color values (i.e., 0-127).

(2) The compression ratio achieved by the AQ-HCDC algorithm is higher than that achieved by the lossless PNG and less than that achieved by the lossless 8-bit GIF and the lossy JPEG. This is at the cost of some reduction/improvement in the image quality, where the AQ-HCDC algorithm provides better image quality than that produced by the JPEG format and of course less image quality in comparison with PNG.

(3) The AQ-HCDC algorithm provides higher compression ratio than ZIP and competitive performance to WinRAR for all images except for Baboon, where both tools are higher due to the image structure and characters frequencies. Although, this doesn't look a fare comparison, however, it is very useful to demonstrate the compression power of the AQ-HCDC algorithm.

(4) The AQ-HCDC algorithm provides better image quality that the lossy JPEG algorithm, which is also meeting the standards for acceptable image quality. It provides a PNSR of more than 30 dB.

(5) The AQ-HCDC algorithm achieves a compression ratio and image quality that are high enough to be competent with the compression ratio and image quality achieved by many well-known algorithms of statistical and adaptive nature.

## 5.2 Recommendations for Future Work

The main recommendations for future work are:

(1) Perform further investigations on the AQ-HCDC algorithm to cover a wider range of image sizes and of various colors frequencies.

(2) Evaluate the compression ratio of the AQ-HCDC algorithm followed by ZIP or WinRAR to compress BMP images, and compare it against the compression ratio for applying ZIP or WinRAR to other lossless (e.g., GIF and PNG) and lossy (e.g., JPEG) compression formats.

(3) Develop an optimized version of the code to compare its runtime with other compression algorithms and state-of-the-art software. In addition, to compare the compression and the decompression processing runtimes.

Modify the core of AQ-HCDC algorithm itself in some way to provide higher compression ratio, while maintaining the maximum possible quality for the compressed image.

## References

*ZIP*. (2011, April 14). Retrieved from Info-ZIP: http://www.info-zip.org/mans/zip.html

*BMP file format*. (2012, September 25). Retrieved from Wikipedia: http://en.wikipedia.org/wiki/BMP_file_format

*Graphics Interchange Format*. (2012, September 18). Retrieved from wikipedia: http://en.wikipedia.org/wiki/Graphics_Interchange_Format

*JPEG*. (2012, September 19). Retrieved from wikipedia: http://en.wikipedia.org/wiki/JPEG

*Portable Network Graphics*. (2013, January 23). Retrieved from DBpedia: http://live.dbpedia.org/page/Portable_Network_Graphics

Adiego, J., Navarro, G., & de la Fuente, P. (2007). Using structural contexts to compress semistructured text collections. *Information Processing & Management, 43*(3), 769–790.

Aguilera, P. (2006). *Comparison of different image compression formats.* Retrieved December 10, 2012, from CAE: http://homepages.cae.wisc.edu/~ece533/project/f06/aguilera_rpt.pdf

Al-Bahadili, H. (2008). A novel lossless data compression scheme based on the error correcting Hamming codes. *Computers & Mathematics with Applications, 56*(1), 143–150.

Al-Bahadili, H., & Hussain, S. M. (2008). An adaptive character wordlength algorithm for data compression. *Computers & Mathematics with Applications, 55*(6), 1250–1256.

Al-Bahadili, H., & Hussain, S. M. (2010). Bit-level Text Compression Scheme Based on the ACW Algorithm. *International Journal of Automation and Computing, 7*(1), 123-131.

Al-Bahadili, H., & Rababa'a, A. (2010). A Bit-Level Text Compression Scheme Based on the HCDC Algorithm. *International Journal of Computers and Applications, 32*(3), 123-131.

Al-Saab, S. (2011). A Novel Search Engine Model Based on Index-Query Bit-Level Compression. *The Research Bulletin of Jordan ACM, 2*(4), 119-126.

AlZboun, F. (2011). Hamming Correction Code Based Compression for Speech Linear Prediction Reflection Coefficients. *International Journal of Mobile & Adhoc Network, 1*(2), 228-233.

Ameer, S., & Basir, O. (2009). Image compression using plane fitting with inter-block prediction. *Image and Vision Computing, 27*(4), 385-390.

Amro, I., Zitar, R. A., & Bahadili, H. (2010). Speech compression exploiting linear prediction coefficients codebook and hamming correction code algorithm. *International Journal of Speech Technology, 14*(2), 65–76.

Bandyopadhyay, S. K., Paul, T. U., & Raychoudhury, A. (2011). Image Compression using Approximate Matching and Run Length. *International Journal of Advanced Computer Science and Applications, 2*(6), 117.

Barni, M. (2006, May). *Fractal Image Compression* (Vol. 978). CRC/Taylor & Francis. Retrieved April 5, 2011

Bentley, F., & Barrett, E. (2012). *Building Mobile Experiences.* The MIT Press.

Brittain, N. J., & El-Sakka, M. R. (2007). Grayscale true two-dimensional dictionary-based image compression. *Journal of Visual Communication and Image Representation, 18*(1), 35–44.

Cagnazzo, M., Cicala, L., Poggi, G., & Verdoliva, L. (2006). Low-complexity compression of multispectral images based on classified transform coding. *Signal Processing: Image Communication, 21*(10), 850-861.

Douak, F., Benzid, R., & Benoudjit, N. (2011). Color image compression algorithm based on the DCT transform combined to an adaptive block scanning. *AEU - International Journal of Electronics and Communications, 65*(1), 16-26.

Du, Q., & Chang, C.-I. (2004). Linear mixture analysis-based compression for hyperspectral image analysis. *IEEE Transactions on Geoscience and Remote Sensing, 42*(4), 875–891.

Dudek, G., Borys, P., & Grzywna, Z. J. (2007). Lossy dictionary-based image compression method. *Image and Vision Computing, 25*(6), 883-889.

Duffy, T. J. (2012). *Programming with Mobile Applications: Android(TM), iOS, and Windows Phone 7* (1st ed.). Course Technology.

Edirisinghe, E. A., Nayan, M. Y., & Bez, H. E. (2004). A wavelet implementation of the pioneering block-based disparity compensated predictive coding algorithm for stereo image pair compression. *Signal Processing: Image Communication, 19*(1), 37-46.

Forouzan, B. A. (2007). *Data Communications Networking* (4th ed.). McGraw-Hill.

Freschi, V., & Bogliolo, A. (2004). Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism. *Information Processing Letters, 90*(4), 167–173.

Gelli, G., & Poggi, G. (1999). Compression of multispectral images by spectral classification and transform coding. *IEEE Transactions on Image Process, 8*(4), 476-89.

Gilbert, J., & Abrahamson, D. M. (2006). Adaptive object code compression. *the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, (pp. 282–292). New York.

Gryder, R., & Hake, K. (1991). *SURVEY OF DATA COMPRESSION TECHNIQUES.* Virginia: OAK RIDGE NATIONAL LABORATORY.

Howard, P. G., & Vitter, J. S. (1994). Arithmetic coding for data compression. *Proceedings of the IEEE, 82*(6), 857–865.

Hu, Y. C., & Chang, C. C. (2000). A new lossless compression scheme based on Huffman coding scheme for image compression. *Signal Processing: Image Communication, 16*(4), 367-372.

Huffman, D. A. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE, 40*(9), 1098 –1101.

Huynh-Thu, Q., & Ghanbari, M. (2008). Scope of validity of PSNR in image/video quality assessment. *Electronics Letters, 44*(13), 800–801.

Karpinski, M., & Nekrich, Y. (2009). A Fast Algorithm for Adaptive Prefix Coding. *Algorithmica*, 29–41.

Keshava, N. (2004). Distance metrics and band selection in hyperspectral processing with applications to material identification and spectral libraries. *IEEE Transactions on Geoscience and Remote Sensing, 42*(7), 1552–1565.

Khancome, C. (2011). Text Compression Algorithm Using Bits for Character Representation. *international journal of advanced computer science, 1*(6), 215-219.

Kimura, N., & Latifi, S. (2005). A survey on data compression in wireless sensor networks. *International Conference on Information Technology: Coding and Computing*, (pp. 8–13).

Klein, S. T. (2000). Skeleton Trees for the Efficient Decoding of Huffman Encoded. *Information Retrieval, 3*(1), 7-23.

Knuth, D. E. (1985). Dynamic huffman coding. *Journal of Algorithms, 9*(2), 163–180.

Kramm, M. (2007). Compression of Image Clusters using Karhunen Loeve Transformations. *Human Vision and Electronic Imaging*, *7*, pp. 101-106. San Jose, CA.

Krinidis, M., Nikolaidis, N., & Pitas, I. (2007). The discrete modal transform and its application to lossy image compression. *Signal Processing: Image Communication, 22*(5), 480-504.

Krishnamoorthi, R., & Seetharaman, K. (2007). Image compression based on a family of stochastic models. *Signal Processing, 87*(3), 408-416.

Kui Liu, Y., & Žalik, B. (2005). An efficient chain code with Huffman coding. *Pattern Recognition*, 553–557.

Li, R. Y., Kim, J., & Al-Shamakhi, N. (2002). Image compression using transformed vector quantization. *Image and Vision Computing, 20*(1), 37-45.

Mahoney, M. (2000). Fast Text Compression with Neural Networks. *the 13th International Florida Artificial Intelligence Research Society Conference*, (pp. 230–234). Orlando.

Mateu-Villarroya, P., & Prades-Nebot, J. (2001). Lossless image compression using ordered binary-decision diagrams. *Electronics Letters, 37*(3), 162–163.

Nanavati, P. S., & Panigrahi, K. P. (2005). Wavelets: Applications to Image Compression-II. *Indian Academy of Sciences*.

Nelson, M. R. (1989). LZW data compression. *Dr. Dobb's J., 14*(10), 29–36.

Nofal, S. (2007). Bit-Level Text Compression. *the First International Conference on Digital Communications and Computer Applications*, (pp. 486 – 488). Irbid.

Pan, H., Jin, L. Z., Yuan, X. H., Xia, S. Y., & Xia, L. Z. (2010). Context-based embedded image compression using binary wavelet transform. *Image and Vision Computing, 28*(6), 991-1002.

Pandya, M. K. (2000). *Data Compression: Efficiency of Varied Compression Techniques.* UK: University of Brunel.

Patterson, A. D., & Hennessy, L. J. (2011). *Computer Organization and Design: The Hardware/Software Interface* ( Revised Fourth Edition ed.). Elsevier.

Rahman, Z., Jobson, D. J., & Woodell, G. A. (2011). Investigating the relationship between image enhancement and Image compression in the context of the multi-scale retinex. *Journal of Visual Communication and Image Representation, 22*(3), 237–250.

Rueda, L. G., & Oommen, B. J. (2006). A Fast and Efficient Nearly-Optimal Adaptive Fano Coding Scheme. *Information Science, 176*(12), 1656-1683.

Rueda, L. G., & Oommen, B. J. (n.d.). Enhanced static Fano coding., (pp. 2163 –2169).

Saghri, J., Tescher, A., & Reagan, J. (2005). Practical transform coding of multispectral imagery. *IEEE Signal Processing Magazine, 12*(1), 32–43.

Salomon, D. (2002). *A Guide to Data Compression Methods* (1st ed.). Springer.

Salomon, D. (2004). *Data Compression: The Complete Reference* (3rd ed.). Springer.

Salomon, D. (2007). *Data Compression: The Complete Reference* (4th ed.). Springer.

Sayood, K. (2012). *Introduction to data compression* (4th ed.). Morgan Kaufmann.

Shapira, D., & Daptardar, A. (2006). Adapting the Knuth–Morris–Pratt algorithm for pattern matching in Huffman encoded texts. *Information Processing & Management, 42*(2), 429-439.

Singh, S. K., & Kumar, S. (2011). Novel adaptive color space transform and application to image compression. *Signal Processing: Image Communication, 26*(10), 662–672.

Sirbu, A., & Cleju, I. (2011). On Some Characteristics of a Novel Lossless Data Compression Algorithm based on Polynomial Codes. *Memoirs of the Scientific Sections of the Romanian Academy*, 1-12. Retrieved from http://iit.iit.tuiasi.ro/Reviste/mem_sc_st_2011/14_MSS_Sirbu_Cleju.pdf

Stallings, W. (2010). *Data and Computer Communications* (9th ed.). Prentice Hall.

Talukder, K. H., & Harada, K. (2007). Haar Wavelet Based Approach for Image Compression and Quality Assessment of Compressed Image. *IAENG International Journal of Applied Mathematics, 36*(1), 1-8.

Tanenbaum, A. (2003). *Computer Networks.* Prentice Hall.

Telagarapu, P. (2011). Image Compression Using DCT and Wavelet Transformations. *International Journal of Signal Processing, Image Processing and Pattern Recognition, 4*(3), 61-73.

Thomos, N., Boulgouris, N. V., & Strintzis, M. G. (2006). Optimized Transmission of JPEG2000 Streams Over Wireless Channels. *IEEE Transactions on Image Processing.*

Thomos, N., Boulgouris, N. V., & Strintzis, M. G. (2006). Optimized Transmission of JPEG2000 Streams Over Wireless Channels. *IEEE Transactions on Image Processing, 15*(1), 54-67.

Timothy, C. B., John, G. C., & Ian, H. W. (1990). *Text Compression.* Prentice Hall.

Velisavljevic, V., Beferull-Lozano, B., & Vetterli, M. (2007). Efficient Image Compression Using Directionlets. *Proceedings of the International Conference on Information, Communications & Signal Processing (ICICS '07)*, (pp. 1-5). Singapore.

Velisavljevic, V., Beferull-Lozano, B., Vetterli, M., & Dragotti, P. L. (2006). Low-Rate Reduced Complexity Image Compression Using Directionlets. *the IEEE International Conference on Image Processing (ICIP '06)*, (pp. 1601–1604). Atlanta, USA.

Vitter, J. S. (1989). Algorithm 673: Dynamic Huffman coding. *ACM Transactions on Mathematical Software (TOMS), 15*(2), 158-167. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.68.7162

Wang, H., Babacan, D. S., & Sayood, K. (2007). Lossless hyperspectral image compression using context-based conditional average. *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, 45*(12), 4187–4192.

Wang, Z. H., Chang, C. C., Chen, K. N., & Li, M. C. (2010). A Modification of VQ Index Table for Data Embedding and Lossless Indices Recovery. *Journal of Computers, 20*(4), 42–52.

Wen, Z., Liu, Z., Cohen, M., Li, J., Zheng, K., & Huang, T. (2004). Low bit-rate video streaming for face-toface teleconference. *IEEE International Conference on Multimedia and Expo (ICME '04), 3*, 1631–1634.

Witten, I. H. (2004). Adaptive text mining: inferring structure from sequences. *Journal of Discrete Algorithms, 0*, 137–159.

Witten, I. H., & Neal, R. M. (1987). Arithmetic coding for data compression. *Communications of the ACM, 30*(6), 520–540.

Wong, K.-W., Lin, Q., & Chen, J. (2011). Error detection in arithmetic coding with artificial markers. *Computers & Mathematics with Applications*, 359–366.

Xie, Y., Wolf, W., & Lekatsas, H. (2003). Code compression using variable-to-fixed coding based on arithmetic coding. *the Data Compression*, (pp. 382 –391). New Jersey.

Ze-Nian Li, & Mark, S. D. (2004). *Fundamentals of multimedia* (1st ed.). Pearson Prentice Hall.

Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory, 23*(3), 337– 343.

Ziv, J., & Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory, 24*(5), 530–536.