



## **Enhancing the Compression Ratio of the HCDC-Based Text Compression Scheme**

**تقنية تحسين نسبة ضغط النصوص المستخدمة لأسلوب HCDC**

By

**Ahmed Imad Mohammed Ali**

Supervisors

**Dr. Hussein Al-Bahadili**

This thesis is submitted to the Department of Computer Information Systems, Faculty of Information Technology, Middle East University in partial fulfillment for the requirement of the degree of Master of Science in Computer information Systems.

Middle-East University  
Faculty of Information Technology  
Amman, Jordan

(January, 2013)

**Middle East University**  
**Examination Committee Decision**

This to certify that the thesis entitled "Enhancing the Compression Ratio of the HCDC-Based Text Compression Scheme" was successfully defended and approved on January 14<sup>th</sup> 2013

Examination Committee Members	Signature
-------------------------------	-----------

Dr. Hussein Al-Bahadili

Associate Professor, Department of Computer Science

(Petra University)



Dr. Ahmad Kayed

Associate Professor, Department of Computer Science

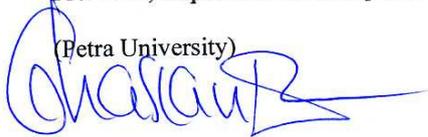
(Middle East University)



Dr. Ghassan F. Isaa

Professor, Department of Computer Science

(Petra University)



### أقرار التفويض

انا احمد عماد محمد علي افوض جامعة الشرق الاوسط بتزويد نسخ من رسالتي للمكتبات او  
الموسسات او الهيئات او الافراد عند طلبها.

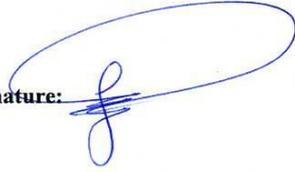
  
التوقيع:

التاريخ: ١٤ / ١ / ١٣٠٢

**Authorization Statement**

I'm Ahmed Imad Mohammed Ali, authorize the Middle East University to Supply a copy of my thesis to libraries, establishments or individuals upon their request.

**Signature:**

A handwritten signature in blue ink, consisting of a large, stylized loop at the top and a vertical stroke with a horizontal crossbar at the bottom.

**Date:** 14/1/2013

## Declaration

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

Signature:

Date:

Ahmed Imad Mohammed Ali

Department of Computer Information system

Faculty of Information Technology

Middle East University

## **Dedication**

I dedicate this work to my family for their, understanding and support, they were the light in my path, without them, nothing of this would have been possible.

Thank you for everything, I love you!

## **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 angles, how to approach it, and how to find the most suitable solution.

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

Finally, I should be thankful to my parents, my sisters and friends for their support and encouragement over the years, as well as to anyone who supported me during my master study.

## Abstract

Data compression is the process of reducing the size of data so that it requires less disk space for storage and less bandwidth to be transmitted over data communication channels. Reducing size of transmitted data compression also reduces the amount of errors during data transmission over error-prone (noisy) data communication channels by decreasing the size of information to be exchanged over such noisy channels. An additional benefit of data compression is in mobile and wireless communication devices, where it reduces transmission power consumption, because the transmission power consumption is directly proportional to the size of the transmitted data.

Al-Bahadili (2008) developed a new lossless, bit-level, and a symmetric data compression. The algorithm is based on the concept of Hamming Codes, and it is referred to as the Hamming Codes based Data Compression (HCDC) algorithm. Then the HCDC was used to develop a repeatedly applied adaptive text compression algorithm, which was referred to as the HCDC( $k$ ), where  $k$  refers to the number of repetition loops. The algorithms have taken the attention of many researchers around the world, due to their tremendous compression ratio and the potential they have. This encourages us to take step ahead to further improve the performance of the HCDC algorithm.

This thesis is concerned with the development of an enhanced version of the HCDC algorithm, namely, the E-HCDC algorithm. The E-HCDC utilizes a new encoding scheme called the  $m$ -encoding scheme, where  $m$  represents the maximum length for the pre-fix bits. The compression ratios achieved by the E-HCDC, over a number of text files of various sizes and natures from widely-used corpora, are evaluated; and the enhancement ratio is computed. The E-HCDC demonstrates an enhancement ratio of about 15% to 20% for the different text files under investigation. The compression ratios of the E-HCDC are also compared against the compression ratios of various well-known algorithms. It demonstrates a competitive performance, which waves the way for further research and development.

## الملخص

ضغط البيانات هو عملية تقليل حجم البيانات بحيث تتطلب أقل مساحة القرص لتخزين وأقل عرض النطاق الترددي لنقلها عبر قنوات الاتصال البيانات. الحد من حجم البيانات المرسله ضغط يقلل أيضا من كمية الأخطاء أثناء نقل البيانات عبر قنوات الاتصال المعرضة للخطأ (صاخبة). فائدة إضافية من ضغط البيانات في أجهزة الاتصالات النقالة واللاسلكية، حيث أنه يقلل من استهلاك طاقة نقل البيانات، وذلك لأن استهلاك الطاقة النقال يتناسب طرديا مع حجم البيانات المنقولة.

وضع البهادلي (2008) تقنية جديدة، وهي بت المستوى، وضغط البيانات المتماثلة. وتستند الخوارزمية على مفهوم رموز هامنك (Hamming Codes)، ويشار إلى الخوارزمية بأسم (Hamming Codes based Data Compression) ويرمز اليها (HCDC). ثم تم استخدام خوارزمية (HCDC) لتطوير تطبيق ضغط النصوص (HCDC(k)). اتخذت خوارزميات (HCDC) انتباه العديد من الباحثين في جميع أنحاء العالم، وذلك بسبب نسبة الضغط الهائلة وإمكاناتها العالية. هذا يشجعنا على اتخاذ خطوة إلى الأمام لزيادة نسبة الضغط وتشرح هذه الأطروحة نسخة محسنة (Enhanced) من الخوارزمية (HCDC) ويرمز لها (E-HCDC).

تستخدم خوارزمية E-HCDC نظام ترميز جديد يسمى ( $m$ -encoding)، حيث تمثل  $m$  الحد الأقصى لطول البتات قبل الضغط. يتم تقييم نسب ضغط الذي حققته E-HCDC على مدى عدد من الملفات النصية من مختلف الأحجام والطبيعة من المجاميع المستخدمة على نطاق واسع. و E-HCDC توضح نسبة زيادة من حوالي 15% إلى 20% للملفات نصية مختلفة. تتم مقارنة النسب أيضا من ضغط E-HCDC ضد نسب ضغط من الخوارزميات المعروفة المختلفة. فإنه يدل على الأداء التنافسي لهذه الخوارزمية فاتحة الطريق لمزيد من البحث والتطوير.

## Table of Contents

اقرار التفويض.....	III
<b>AUTHORIZATION STATEMENT .....</b>	<b>IV</b>
<b>DECLARATION.....</b>	<b>V</b>
<b>DEDICATION.....</b>	<b>VI</b>
<b>ACKNOWLEDGMENT .....</b>	<b>VII</b>
<b>ABSTRACT.....</b>	<b>VIII</b>
الملخص.....	IX
<b>TABLE OF CONTENTS .....</b>	<b>X</b>
<b>LIST OF FIGURES .....</b>	<b>XII</b>
<b>LIST OF TABLES .....</b>	<b>XIII</b>
<b>ABBREVIATIONS .....</b>	<b>XV</b>
<b>CHAPTER ONE .....</b>	<b>1</b>
1.1    CATEGORIZATION OF DATA COMPRESSION ALGORITHMS .....	3
1.1.1 <i>Data compression fidelity</i> .....	3
1.1.2 <i>Length of data compression symbols</i> .....	4
1.1.3 <i>Data compression symbol tables</i> .....	4
1.1.4 <i>Data compression cost</i> .....	6
1.2    DATA COMPRESSION MODELS .....	7
1.3    TEXT COMPRESSION .....	8
1.3.1 <i>Syllables and words based text compression</i> .....	8
1.3.2 <i>Bit-level text compression</i> .....	9
1.4    PERFORMANCE MEASURES.....	9
1.4.1 <i>Measuring the amount of compression</i> .....	9
1.4.2 <i>Processing time</i> .....	11
1.5    STATEMENT OF THE PROBLEM .....	12
1.6    OBJECTIVES OF THIS THESIS .....	13
1.7    ORGANIZATION OF THE THESIS .....	13
<b>CHAPTER TWO .....</b>	<b>15</b>
2.1    THE HCDC ALGORITHM .....	16
2.1.1 <i>Description of the HCDC algorithm</i> .....	16

2.1.2	<i>Derivation and analysis of HCDC algorithm compression ratio</i> .....	19
2.2	ENGLISH TEXT COMPRESSION.....	23
2.2.1	<i>Entropy of English text</i> .....	24
2.3	THE ADAPTIVE CHARACTER CODING FORMAT.....	26
2.4	REVIEW OF BIT-LEVEL TEXT COMPRESSION ALGORITHMS.....	29
2.5	REVIEW OF SYLLABLE/WORD-BASED TEXT COMPRESSION ALGORITHMS.....	33
<b>CHAPTER THREE.....</b>		<b>38</b>
3.1	LIMITATIONS OF THE HCDC ALGORITHM.....	39
3.2	THE <i>M</i> -ENCODING SCHEME.....	39
3.3	THE E-HCDC ALGORITHM.....	44
3.3.1	<i>The E-HCDC Compressor</i> .....	44
3.3.2	<i>The E-HCDC Decompressor</i> .....	47
3.4	THE E-HCDC HEADER.....	50
<b>CHAPTER FOUR.....</b>		<b>52</b>
4.1	EXPERIMENT #1: COMPARING THE COMPRESSION RATIO OF THE E-HCDC AND HCDC ALGORITHMS.....	53
4.2	EXPERIMENT #2: COMPARING THE COMPRESSION RATIO OF THE E-HCDC AND A NUMBER OF STATISTICAL ALGORITHMS.....	56
4.3	EXPERIMENT #3: COMPARING THE COMPRESSION RATIO OF THE E-HCDC AND A NUMBER OF ADAPTIVE ALGORITHMS.....	58
<b>CHAPTER FIVE.....</b>		<b>61</b>
5.1	CONCLUSIONS.....	61
5.2	RECOMMENDATIONS FOR FUTURE WORK.....	62
<b>REFERENCES.....</b>		<b>63</b>
<b>APPENDIX A.....</b>		<b>70</b>
A.1	CALGARY CORPUS.....	71
A.2	CANTERBURY CORPUS.....	73
A.3	ARTIFICIAL CORPUS.....	73
A.4	LARGE CORPUS.....	74
A.5	MISCELLANEOUS CORPUS.....	74
<b>APPENDIX B.....</b>		<b>76</b>

## List of Figures

<u>Figure</u>	<u>Title</u>	<u>Page</u>
2.1	Locations of data and parity bits in 7-bit codeword	18
2.2	The main steps of the HCDC compressor	19
2.3	The main steps of the HCDC decompressor	19
2.4	Variation of $C_{\min}$ and $C_{\max}$ with $p$	21
2.5	Variation of $r_l$ with $p$	22
2.6	Variations of $C$ with respect to $r$ for various values of $p$	23
3.1	The procedure of the E-HCDC compressor	46
3.2	The procedure of the E-HCDC decompressor	49
3.3	The structure of the E-HCDC compressed file header	51
4.1	Comparing $C_{HCDC}$ and $C_{E-HCDC}$ for a number of text files from the Calgary Corpus	55
4.2	Comparing $C_{HCDC}$ and $C_{E-HCDC}$ for a number of text files from the Canterbury, Artificial, and Large Corpora	55
4.3	Comparing $C_{E-HCDC}$ of the E-HCDC algorithm against the compression ratio of various statistical algorithms	58
4.4	Comparing $C_{E-HCDC}$ of the E-HCDC algorithm against the compression ratio of various adaptive and statistical algorithms	60

## List of Tables

<u>Table</u>	<u>Title</u>	<u>Page</u>
2.1	Variation of $C_{min}$ , $C_{max}$ , and $r_l$ with number of parity bits ( $p$ )	21
2.2	Variations of $C$ with respect to $r$ for various values of $p$	22
2.3	Valid 7-bit codewords	24
2.4	Information content of English text using different approaches	25
2.5	Information content of the English text using different standard software	26
2.6	Lossless compression ratios for text compression Calgary corpus	27
2.7	ASCII and adaptive codes of the 10 most common characters in paper1 file	28
2.8	Entropies of the binary sequences generated for a number of text files from the Calgary corpus using different coding formats	29
3.1	The pre-fix bits and the number of bits written to compressed data file	41
3.2	The pre-fix bits and the number of bits written to the compressed data file	43
3.3	Components of the HCDC field of E-HCDC compressed file header	51
4.1	Experiment #1 Comparing $C_{HCDC}$ and $C_{E-HCDC}$	54
4.2	Experiment #2 Comparing $C_{E-HCDC}$ of the E-HCDC algorithm against the compression ratios of various statistical algorithms	57
4.3	Comparing the compression ratio of the E-HCDC algorithm against various adaptive algorithms	59
A.1	Calgary Corpus	72

A.2	Canterbury Corpus	73
A.3	Artificial Corpus	74
A.4	Large Corpus	74
A.5	The Miscellaneous Corpus	75
B.1	Filename: alice29.txt	77
B.2	Filename: bib.txt	79
B.3	Filename: bible.txt	81
B.4	Filename: book1.txt	83
B.5	Filename: paper1	85

## Abbreviations

ACW:	Adaptive Character Word-length.
ASCII:	American Standard Code for Information Interchange.
BWT:	Burrows-Wheeler Transform.
CIQ:	Compressed Index-Query
E-HCDC:	Enhanced Hamming Code Data Compression
FLH:	Fixed-Length Hamming.
GZIP:	GNU zip
HCDC:	Hamming Code Data Compression.
HU:	Huffman Coding
ICT:	Information and Communication Technology
LDPC:	Low-Density Parity-Check
NVC:	Non-valid Codeword
RLE:	Run Length Encoding
VB:	Visual Basic
VC:	Valid Codeword
VoIP:	Voice over Internet Protocol

# Chapter One

## Introduction

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). Data compression also reduces the amount of errors during data transmission over error-prone data communication channels by decreasing the size of information to be exchanged over such channels (Joaquín Adiego, Navarro, & De la Fuente, 2007; Freschi & Bogliolo, 2004).

An additional benefit of data compression is in wireless communication devices where it may add a significant power saving. Power savings is possible by compressing data prior to transmission power consumption, where the power consumed is directly proportional to the size of the transmitted data. In fact, in wireless devices, transmission of a single bit may require over  $10^3$  times of more power than a single 32-bit computation (Sharma, Golubchik, Govindan, & Neely, 2009).

Data compression is usually obtained by substituting a shorter symbol for an original symbol in source data, containing the same information but with a smaller representation in length. The symbols may be characters, words, phrases, or any other units that may be stored in a dictionary of symbols and processed by a computing system (I.H. Witten, 2004).

Data compression 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. 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.

Data coding techniques have been widely used in developing data compression algorithms, since coding techniques may lend themselves well to the above concept. Data coding involves processing an input sequence (L. Rueda & Oommen, 2006).

An algorithm or coding function is called distinct if its mapping from source messages to codewords is one-to-one. Such a code is called uniquely decodable if every codeword is recognizable even when immersed in a stream of other codewords. A uniquely decodable code is known as a prefix code if it has the property that no codeword in the code is a prefix of any other codeword (Sayood, 2012).

Recently, a new lossless, bit-level, and a symmetric data compression was developed. The algorithm is based on the concept of Hamming Codes, and it is referred to as the Hamming codes based Data Compression (HCDC) algorithm (Al-Bahadili, 2008). In the HCDC algorithm, the source data is converted to binary sequence, which is then divided into  $n$ -bit blocks. Each block is considered as a Hamming codeword that consists of  $p$  parity bits and  $d$  data bits. In HCDC, as in Hamming codes, the codewords are classified into  $2^d$  valid codewords and  $2^n - 2^d$  non-valid codewords. During the compression process, each block is tested to find out if it is a valid codeword or a non-valid codeword. If the block is valid codeword, the  $d$  data bits preceded by 0 (i.e.,  $d+1$  bits) are appended to the compressed sequence, while for a non-valid codeword block, the  $n$ -bit block preceded by 1 (i.e.,  $n+1$  bits) is appended to the compressed sequence. These additional redundant 0 and 1 bits are required for the decompression process.

Then the HCDC algorithm was used to develop an adaptive bit-level text compression scheme, which was referred to as the HCDC( $k$ ) algorithm (Al-Bahadili & Rababa'a, 2010). It showed an excellent performance when it was compared with well-known data compression algorithm and state of the art-software. The HCDC( $k$ ) algorithm consists of six steps some of which are repetitively applied to achieve higher compression ratio. The repetition loops continue until inflation is detected and the accumulated compression ratio is the multiplication of the compression ratios of the individual loops and therefore it was called HCDC( $k$ ), where  $k$  refers to the number of repetition loops.

In order to enhance the compression power of the HCDC( $k$ ) scheme, an adaptive encoding format was proposed in which a character is encoded to binary according to its probability. This method of encoding reduces the binary sequence entropy so that it grants higher compression ratio.

## 1.1 Categorization of Data Compression Algorithms

A large number of data compression algorithms have been developed during the last decades; these algorithms are based of different concepts or principles to utilize the nature of the data such that reducing the size of the original data. Thus, it has been realized that it is necessary to categorize these algorithms in a way to help the researchers and developers understand these algorithms. Consequently, data compression algorithms are categorized by several characteristics, such as (Salomon, 2004; Sayood, 2012):

- Data compression fidelity
- Length of data compression symbols
- Data compression symbol table
- Data compression cost

In the following a brief definition is given for each of them.

### 1.1.1 Data compression fidelity

One of the most important characteristics is the data compression 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. As a result of those two fundamentally different styles of data compression can be recognized, depending on the fidelity of the restored data, these are:

#### (1) Lossless data compression

It involves a transformation of the representation of the original data set so as to make it 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 original and decompressed files must be identical. Lossless compression is used in many applications, for example, 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 (Brittain & El-Sakka, 2007; L. Rueda & Oommen, 2006).

## **(2) Lossy data compression**

It involves a transformation of the representation of the original data set whereas it is impossible to reproduce exactly the original data set, but performing a decompression transformation reproduces an approximate representation. 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; I.H. Witten, 2004).

### **1.1.2 Length of data compression symbols**

Data compression algorithms are characterized by the length of the symbols in algorithm process, regardless of whether the algorithm uses variable length symbols in the original data or in the compressed data, or both. For example, Run-Length Encoding (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.

### **1.1.3 Data compression symbol tables**

Another distinguishing feature of the data compression algorithms is the source of the symbol table. According to this feature, data compression algorithms can be classified into:

#### **(1) 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, while the same dictionary is used for other types of information the efficiency of the algorithm suffers and provides a lower compression ratio.

## **(2) 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.

## **(3) 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. Due to 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 (Gilbert & Abrahamson, 2006; Klein, 2000; Xie, Wolf, & Lekatsas, 2003).

#### 1.1.4 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 requirements 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, as follows:

- (1) Symmetric data compression algorithms. In a symmetric data compression algorithm, the processing times are almost the same for both compression and decompression processes.
- (2) Asymmetric data compression algorithms. In an asymmetric data compression algorithm the compression processing time is more than the decompression processing time.

## 1.2 Data Compression Models

Different data compression algorithms have been recommended and used throughout the years. These data compression algorithms can be classified into four major models depending on the methodology it uses in replacing the original character sets and how it initialize the replacement process and at what level; these are (Pandya, 2000; 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 data compression 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 data compression 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 (Rue 06, Rue 04), static/dynamic Huffman coding (Huffman, 1952; Knuth, 1985; Vitter, 1989), and arithmetic coding (Howard & Vitter, 1994; Ian H. Witten, Neal, & Cleary, 1987).

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

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 such algorithms include: the Adaptive Character Wordlength (ACW(n)) algorithm (Al-Bahadili & Hussain, 2008). The Adaptive Character Wordlength (ACW(n,s)) scheme (Al-Bahadili & Hussain, 2008), the logic-function simplification algorithm (Nofal, 2007), the neural network based algorithm (Mahoney, 2000).

### **1.3 Text Compression**

There are a number of data compression techniques that have been developed 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 efficiently compress a particular type of files. In this work, we are concerned with text files compression.

Text compression can often derive its benefit from the two views of the textual content: the content can be seen as a stream of syllables/words or a sequence of bits. Therefore, text compression algorithms can be categorized into two types: syllables/words based text compression and bit-level text compression. In this section, we provide a brief introduction to each of them.

#### **1.3.1 Syllables and words based text compression**

The word-based methods are older, so many implementations of classical methods exist, for instance Huffman coding (Hashemian, 2003; Huffman, 1952; I.H. Witten, Bell, Emberson, Inglis, & Moffat, 1994), Burrows-Wheeler transformation (Isal & Moffat, 2001), PPM (J. Adiego & De la Fuente, 2006) or Arithmetic coding (Moffat & Isal, 2005). The syllable-based methods are rather young with initial implementations of Huffman coding and LZW (Lánský & Zemlicka, 2005).

Porting of classical character-based methods to syllable or word based is not easy. The transformation heavily influences almost all inner data structures, because they must be able to work with undefined number of syllables or words instead of the original alphabet of 256 characters. Moreover, the large input alphabet also requests the encoder to export elements of the alphabet to the decoder. This issue is often

solved by exporting the alphabet as a part of the encoded document (Lansky & Zemlicka, 2006)

The confrontation and comparison of the word and syllable based methods depends on a language of the input document. Accordingly, the languages with a simple morphology, e.g. English, are better compressed by the word-based algorithms. On the other hand, the languages with a complex morphology are often compressed better by the syllable-based methods.

### **1.3.2 Bit-level text compression**

Text files compression can also be performed at bit-level, as each character has its specific binary representation. However, bit-level data compression algorithms are even younger than the syllable/word based data compression algorithms, therefore, there are only few algorithms have been developed to exploit this concept. However, this type of compression methodology is the core concept of this thesis; the bit-level data compression model will be discussed in details in Chapter 2 with many algorithms, and comprehensive literature review.

## **1.4 Performance Measures**

In order to be able to compare the efficiency of the different compression techniques reliably, and without extreme cases to cloud or bias the technique unfairly, certain issues need to be considered. The most important ones need to be taken into account in evaluating the performance of various algorithms including the following: measuring the amount of compression, and processing time (algorithm complexity).

These issues need to be carefully considered in the context for which the compression algorithm is used. Practically, things like finite memory, error control, type of data, and compression style (adaptive/dynamic, semi-adaptive or static) are all factors that should be considered in comparing the different data compression algorithms(Rico-Juan, Calera-Rubio, & Carrasco, 2005).

### **1.4.1 Measuring the amount of compression**

Several parameters are used to measure the amount of compression that can be achieved by a particular data compression algorithm, such as:

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

The amount of compression is measured by a factor known as compression ratio ( $C$ ), which is defined as the ratio between the size of the data before compression and the size of the data after compression. It is expressed as:

$$C = \frac{S_o}{S_c} \quad (1.1)$$

Where  $S_o$  and  $S_c$  are the sizes of the original and the compressed data, respectively.

### (2) Reduction ratio ( $R$ )

The reduction ratio represents the ratio of difference between the size of the original data ( $S_o$ ) and the size of the compressed data ( $S_c$ ) related to the size of the original data, which is referred to as  $R$ . It is usually given in percent and it is mathematically expressed as:

$$R = \frac{S_o - S_c}{S_o} \times 100 \quad (1.2)$$

### (3) Coding rate ( $C_r$ )

The coding rate expresses the same concept at the compression ratio, but it relates the ratio to a more tangible quantity. For example, for a text file, the coding rate may be expressed in “bits/character” (bpc), where in uncompressed text file a coding rate of 7 or 8 bpc is used. In addition, the coding rate of an audio stream may be expressed in “bits/analogue”, and for still image compression, the coding rate is expressed by “bits/pixel”. The coding rate is expressed as:

$$C_r = \frac{q \cdot S_c}{S_o} \quad (1.3)$$

Where  $q$  is the number of bit represents each symbol in the uncompressed file. The relationship between the coding rate ( $C_r$ ) and the compression ratio ( $C$ ), for example, for text file originally using 7 bpc, can be given by:

$$C_r = \frac{7}{r} \quad (1.4)$$

It is clear from Eqn. (1.4) that a lower coding rate indicates a higher compression ratio.

### 1.4.2 Processing time

The processing time (which is an indication of the algorithm complexity) is defined, as the time required compressing or decompressing the data. These compression and decompression times have to be evaluated separately. As it has been discussed in section 1.3, data compression algorithms are classified according to the processing time into either symmetric or asymmetric algorithms. For a symmetric algorithm, both the compression and the decompression processing time are almost the same, while for an asymmetric algorithm, usually, the compression time is much more than the decompression time.

In this context, data storage applications mainly concern with the amount of compression that can be achieved and the decompression processing time that is required to retrieve the data back (asymmetric algorithms).

Data transmission applications focus predominately on reducing the amount of data to be transmitted over communication channels, and both compression and decompression processing times are the same at the respective junctions (symmetric algorithms) (Kui Liu & Žalik, 2005).

For a fair comparison between the different available algorithms, it is important to consider both the amount of compression and the processing time. Therefore, it would be useful to be able to parameterize the algorithm so that the compression ratio and processing time could be optimized for a particular application.

There are extreme cases where data compression works very well or other conditions where it is inefficient, the type of data that the original data file contains and the upper limits of the processing time have an appreciable effect on the efficiency of the technique selected. Therefore, it is important to select the most appropriate technique for a particular data profile in terms of both data compression and processing time (Dai, 2008)

However, in this thesis, we are mainly concerned with measuring and comparing the compression ratio of the developed algorithm, and other parameters are either can be derived by the reader or left for future work.

## 1.5 Statement of the Problem

The HCDC algorithm and its repetitive version, namely, the HCDC( $k$ ) algorithm have demonstrated an excellent performance and got the attention by many researchers around the world (Al-Saab, 2011; Amro, Zitar, & Bahadili, 2011; Karpinski & Nekrich, 2009; SÎRBU & CLEJU, 2011; W. Y. Wang, 2009; Z. H. Wang, Chang, Chen, & Li, 2009; Wong, Lin, & Chen, 2011) due to the tremendous performance of the algorithm and the potential it has. This encourages us to take step ahead to further improve the performance of the HCDC algorithm.

The HCDC algorithm utilizes a simple encoding scheme for the pre-fix bit, where the character set is divided into two groups only, the first one contains the 16 most common characters, which are considered as valid Hamming codewords and preceded by 1-bit of 0 logic, and the second group contains all remaining characters, which are considered as non-valid Hamming codewords and preceded by 1-bit of 1 logic. For the valid codewords, the pre-fix bit is followed by 4-bit representing the data bits in Hamming codeword, and for the non-valid codewords, the prefix bit is followed by the 7-bit representing the codeword. This appends 5-bit for the valid 16 most common codewords, and 8-bit for the remaining non-valid codewords, which limits the compression power of the algorithm.

Therefore, we believe it is necessary to improve the encoding scheme so we can enlarge the number of valid code words or limit the more redundant bits for the least common character only, while less redundant characters for the middle common characters. This is achieved through the development of a new encoding scheme for the pre-fix bits, namely, the  $m$ -encoding scheme, where  $m$  represents the maximum pre-fix bits sequence. In this new  $m$ -encoding scheme, after finding the characters frequencies and sorting out the characters from the most common to the least common, the characters are split into groups each of 16 characters (in general  $2^d$ , where  $d$  is the number of data in the Hamming codeword).

Then, different pre-fix bits are used to precede the compressed character binary representation during the compression process (both the pre-fix bits and the compressed character binary representation are appended to the compressed binary sequence). These pre-fix bits starts with 1-bit (usually 0) for the first group and increase by one bit for following groups until the number of pre-fix bits reaches  $m$  bits ( usually all 1's).

## 1.6 Objectives of this Thesis

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

- (1) Develop a new encoding scheme for the pre-fix bits combination of various lengths for the valid and non-valid Hamming codewords, which we refer to it as the  $m$ -encoding scheme, where  $m$  is the maximum length for the pre-fix bits.
- (2) Develop an enhanced version of the HCDC algorithm, namely, the E-HCDC algorithm that utilizes the  $m$ -encoding scheme.
- (3) Evaluate the compression ratio of the E-HCDC algorithm for compressing text files from well-known to widely-used corpora.
- (4) Compare the compression ratio of the E-HCDC algorithm against its preceding algorithm (i.e., HCDC algorithm) and estimate the enhancement factor.
- (5) Compare the compression ratio of the E-HCDC algorithm against other well-known and widely-used algorithms.
- (6) Discuss the results obtained, draw conclusions and point-out some recommendations for future work.

## 1.7 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 reveals 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 to lossless (text) compression algorithms. It presents first a review on the most recent work that is related to bit-level data compression algorithms, and second reviews some work related to syllable/word –based data compression algorithms.

Chapter 3 provides a detail description of the E-HCDC algorithm and its core part, the  $m$ -encoding scheme. Chapter 3 also provides a detailed derivation of a formula for computing the optimum  $m$  value. Chapter 4 presents a number of experiments that are performed to evaluate the compression ratio of the E-HCDC algorithm over a number of text file from Standard Corpora, namely, Calgary, Canterbury, Artificial, and Large Corpora.

Chapter 4 also presents a comparison between the compression ratios achieved by the E-HCDC algorithm against the compression ratios of a number of well-known algorithms. In Chapter 5 and based on the results obtained conclusions are drawn and a number of recommendations for future work are pointed-out.

Finally, the thesis also includes two appendices. Appendix A provides the Standard data compression corpora (e.g., Calgary, Canterbury, Artificial, Large, and Miscellaneous Corpora), which are widely used in comparing the performance of the different data compression algorithms. Appendix B lists some statistical values for a number of text files from the various corpora.

## Chapter Two

### Literature Review and Previous Work

A large number of data compression algorithms 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; Solomon, 2004; Solomon, 2002; Moffat & Turpin, 2002). In this work, we are concerned with text compression, which relates to lossless compression so exact form of the original data can be retrieved back.

It has been realized that text compression algorithms can be broadly classified into two main classes according to the representation form of the data at which the compression process is performed; these are:

- (1) Bit-level text compression algorithms
- (2) Syllable or word based text compression algorithms

This thesis is mainly concerned with the development and performance evaluation of an enhanced version of the lossless bit-level text compression algorithm, namely, the Hamming Codes based Data Compression (HCDC) algorithm (Al-Bahadili & Rabab'a, 2007; Al-Bahadili, 2008).

In Section 2.1, we provide a detail description of the HCDC algorithm and the analytical analysis of the performance of the algorithm. Section 2.2 discusses the issues and challenges against English text compression. A new recently proposed coding technique, namely, the Adaptive Character Coding is described in Section 2.3. Then, Section 2.4 provides a review on the most recent development and state-of-the-art in bit-level text compression algorithms, while Section 2.5 reviews some syllable or word -based text compression algorithms.

## 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, 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, we 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 (Kimura & Latifi, 2005; Tanenbaum, 2003).

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:

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

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:

$$n = 2^p - 1 \quad (2.2)$$

The data and the parity bits are located at particular locations in the codeword. The parity bits are located at positions  $2^0, 2^1, 2^2, \dots, 2^{p-1}$  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, 2008)(Al-Bahadili, 2008). In this algorithm, the data symbols (characters) of a source file are converted to binary sequence by concatenating 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:

$$B = \left\lceil \frac{S_o}{n} \right\rceil \quad (2.3)$$

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

$$g = B \cdot n - S_o \quad (2.4)$$

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

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

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.

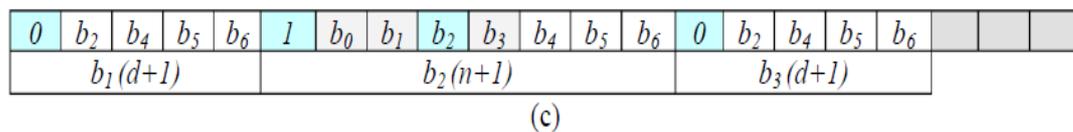
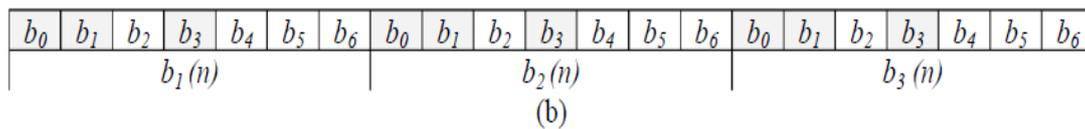
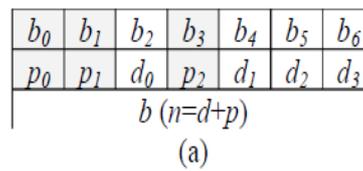


Figure 2.1. (a) Locations of data and parity bits in 7-bit codeword, (b) an uncompressed binary sequence of 21-bit length divided into 3 blocks of 7-bit length, where  $B_1$  and  $B_3$  are valid blocks, and  $B_2$  is a non-valid block, and (c) the compressed binary sequence (18-bit length).

```

Initialization
Select  $p$ 
Calculate  $n = 2^p - 1$ 
Calculate  $d = n - p$ 
Calculate  $B = \text{ceiling}(S_o/n)$ 
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 ( $h = 00$ ) 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:

$$B = v + w \quad (2.6)$$

Where  $v$  and  $w$  are the number of valid and non-valid blocks, respectively. 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:

$$S_v = v (d + 1) \quad (2.7)$$

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:

$$S_w = w (n + 1) \quad (2.8)$$

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

$$S_c = S_v + S_w = v (d + 1) + w (n + 1) \quad (2.9)$$

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

$$S_c = Bn + B - v \cdot p \quad (2.10)$$

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

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

Where  $r=v/B$ , and it represents the fraction of valid blocks. Substitute Eqn. (2.2) into Eqn. (2.11) gives:

$$C = \frac{2^p - 1}{2^p - r p} \quad (2.12)$$

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  $r_l$ , and it can be found out that  $r_l=1/p$ .

Table (2.1) lists the values of  $C_{max}$ ,  $C_{min}$ , and  $r_l$  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  $r_l$  with  $p$ .

$p$	$C_{min}$	$C_{max}$	$r_l$
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

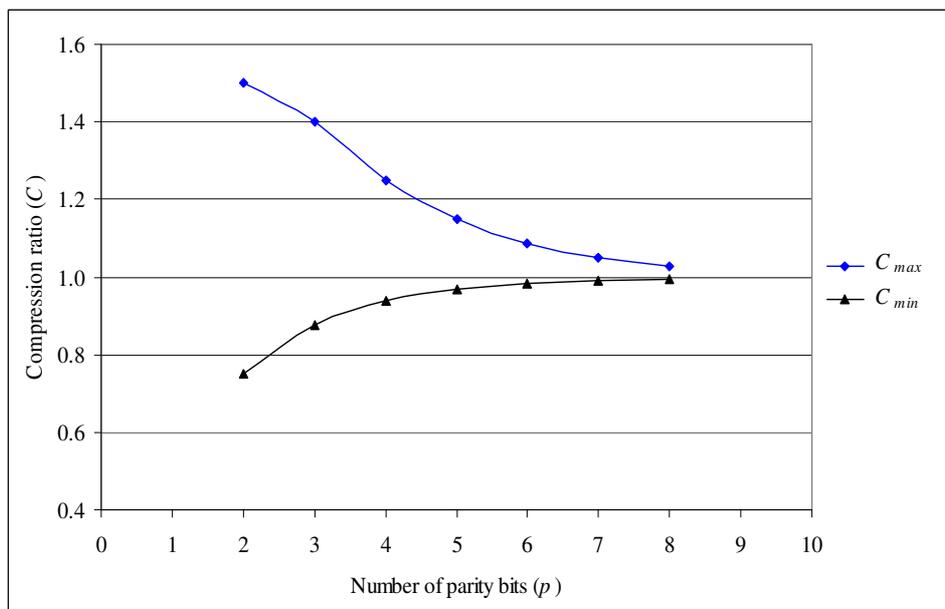


Figure 2.4. Variation of  $C_{min}$  and  $C_{max}$  with  $p$ .

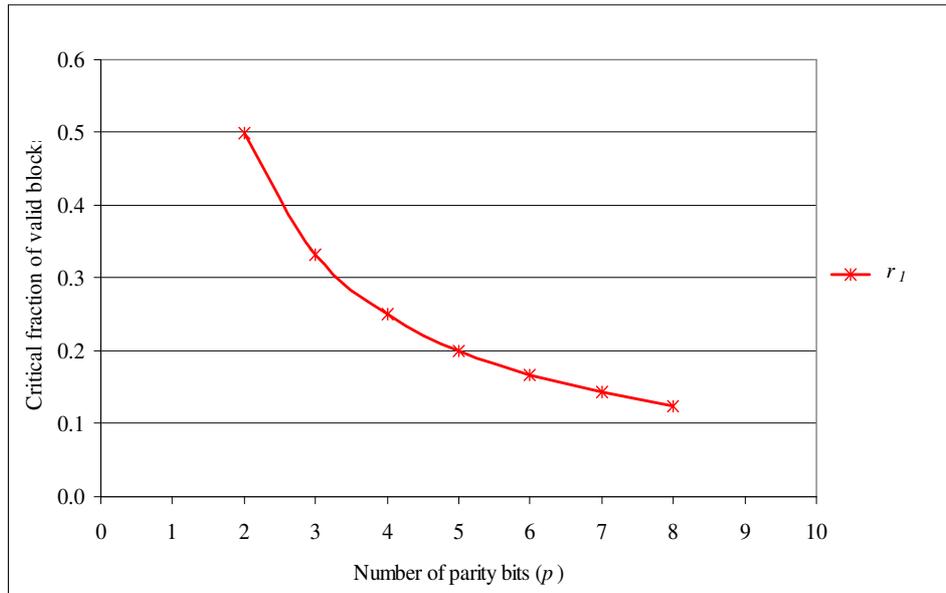


Figure (2.5). Variation of  $r_l$  with  $p$ .

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  $r > r_l$ .

$r$	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

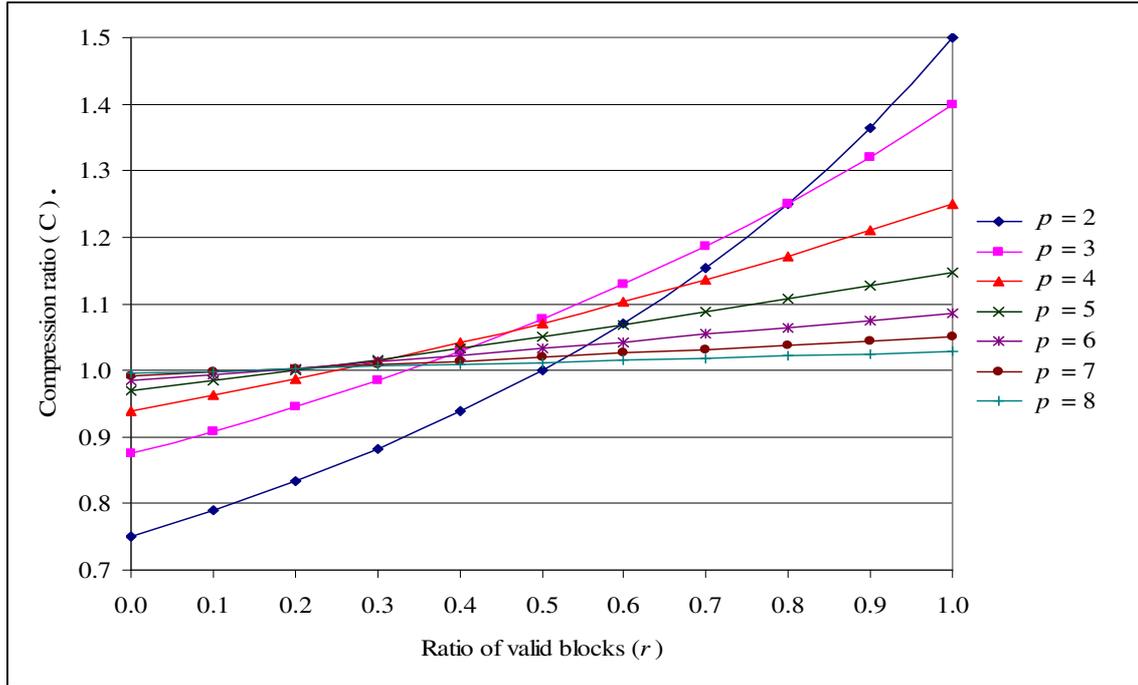


Figure (2.6). Variations of  $C$  with respect to  $r$  for various values of  $p$ .

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

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

Where  $k$  is the number of repetitions,  $S_i$  and  $S_{i-1}$  sizes of the binary file after and before the  $i^{th}$  compression loop, and  $C_i$  is the compression ratio of the  $i^{th}$  compression loop. For  $i=1$ ,  $S_0$  represents the size of the original file.

## 2.2 English Text Compression

English text characters are usually converted to binary using its equivalent 7-bit ASCII codes, which means that each character can be considered as a (7,4) codeword. It has been mentioned earlier that not all 7-bit codewords are valid codewords, in fact only 16 codewords ( $2^4$ ) are valid, and the remaining 112 codewords ( $2^7 - 2^4$ ) are non-valid. The ASCII codes of these 16 valid codewords and the characters they represent are listed in Table (2.3). According to statistics in Standard English text, these valid codewords can be categorized into three groups:

- (1) Wide-use (their decimal values are equivalent to the ASCII code of the characters a and f).
- (2) Rare-use (their decimal values are equivalent to the ASCII code of the characters K, L, R, U, x, \*, -, 3, and 4).
- (3) Not-used (their decimal values are equivalent to the ASCII code of the unprintable characters of 0, 7, 25, 30, and 127 ASCII code).

Thus, encoding characters to binary using their equivalent ASCII codes and testing the validity of these characters either yields a very low compression ratio or most properly yields inflation. This is because a small proportion of the characters within the text may have valid codewords.

Table (2.3) Table 3 Valid 7-bit codewords.		
ASCII Code	Character	Category
0	Control Character	Not used
7	Control Character	Not used
25	Control Character	Not used
30	Control Character	Not used
42	*	Rare-use
45	-	Rare-use
51	3	Rare-use
52	4	Rare-use
75	K	Rare-use
76	L	Rare-use
82	R	Rare-use
85	U	Rare-use
97	a	Frequent –use
102	f	Frequent-use
120	x	Rare-use
127	Control Character	Not-used

### 2.2.1 Entropy of English text

The entropy is a statistical parameter that can be used to measure how much information is produced on the average for each letter of a text in the language. If the language is translated into bits (0 and 1) in the most efficient way, the entropy  $E$ , is the average number of bits required per letter of the original language. The redundancy, on the other hand, measures the amount of constraint imposed on text in the language due to its statistical structure (Shannon, 1951).

For a set of possible messages  $M$ , the entropy is defined as:

$$E(M) = \sum_{m \in M} p(m) \cdot i(m) \quad (2.14)$$

Where  $p(m)$  is the probability of message  $m$ .  $i(m)$  is the notion of the self-information of a message and it is given by:

$$i(m) = \log_2 \left( \frac{1}{p(m)} \right) \quad (2.15)$$

This self-information represents the number of bits of information contained in it and, roughly speaking, the number of bits that should be used to represent that message. Larger entropies represent more information, and perhaps counter-intuitively, the more random a set of messages (the more even the probabilities) the more information they contain on average. The amount of information that is contained by a text could be used as a bound to the maximum amount of compression that can be achieved.

As it has been mentioned in Chapter 1, one way to measure the information content is in terms of the average number of bits per character (bpc), i.e., coding rate ( $C_r$ ). Table (2.4) shows a few approaches that can be used to measure the amount of information contained by an English text in bpc. If all characters are assumed to have equal probabilities, a separate code is used for each character, and there are 96 printable characters (the number on a standard keyboard) then a 7-bit character word length (7 bpc) is required. The entropy, assuming even probabilities ( $p=1/96$ ), is 6.6 bpc.

Table 2.4 information content of the English text using different approaches		
#	Approach	Bpc
1	Standard text (7 bpc)	6.6
2	Entropy	4.5
3	Huffman code (Average)	4.7
4	Entropy (Group of 8 characters)	2.4
5	Asymptotically approaches	1.3

If a probability distribution (based on a corpus of English text) is given for the characters the entropy is reduced to about 4.5 bps. If a separate code is used for each character (for which the Huffman code is optimal), then the number is slightly larger 4.7 bpc.

It is clear that so far no advantage, of relationships among adjacent or nearby characters, is considered. If a text is broken into blocks of 8 characters, and the entropy of those blocks (based on measuring their frequency in an English corpus) is measured, then entropy of about 19 bits is obtained. Thus, since 8 characters are coded at a time, the entropy is 2.4 bpc. If groups of larger and larger blocks are processed, entropy would approach 1.3 (or lower) can be reached. It is impossible to actually measure this because there are too many possible strings to run statistics on, and no corpus is large enough.

This value 1.3 bpc is an estimate of the information content of the English text. Assuming it is approximately correct, this bounds how much can be expected if an English text is losslessly compressed. Table (2.5) shows the compression rate of various data compression algorithms implemented by standard software. All these software, however, are general purpose and not designed specifically for the English text.

#	Software	Bpc
1	Compress	3.7
2	GZIP	2.7
3	BOA	2.0

The Bayesian Optimization Algorithm (BOA) is the current state-of-the-art for general-purpose compressors. To reach 1.3 bpc, the compressor would surely have to know about English grammar, standard idioms, etc. A complete set of compression ratios for the Calgary corpus for a variety of data compression algorithms is shown in Table (2.6).

### 2.3 The Adaptive Character Coding Format

There are different coding formats that can be used in converting a data file into bits. They usually have enormous effects on the entropy of the generated binary sequence, and subsequently affect the compression ratio and the coding rate that can be achieved by a particular data compression algorithm over the compressed data file. A conventional coding format is the ASCII code, in which each character within the source file is coded using an 8-bit codeword. However, a text character is usually coded using 7-bit codeword.

Table (2.6)			
Lossless compression ratios for text compression Calgary corpus.			
#	Scheme	bps	Researcher
1	LZ77	3.94	Ziv and Lempel, 1977
2	LZMW	3.32	Miller and Wegman, 1984
3	LZH	3.30	Brent, 1987
4	MTF	3.24	Moffat, 1987
5	LZB	3.18	Bell, 1987
6	GZIP	2.71	-
7	PPMC	2.48	Moffat, 1988
8	SAKDC	2.47	Williams
9	PPM	2.34	Cleary, Teahan, and Witten, 1994
10	BW	2.29	Burrows and Wheeler, 1995
11	BOA	1.99	Sutton, 1997
12	RK	1.89	Taylor, 1999

Thus, the length of the binary sequence in bits that is generated from encoding a text file into binary sequence is given by:

$$S_o = 7 \cdot T_c \quad (2.16)$$

Where  $S_o$  is the length of the binary sequence in bits, and  $T_c$  is the total number of character within the text file. Another coding format is the Huffman coding, which is described in detail in (Al-Bahadili & Hussain, 2008) Using Huffman coding, the length of the binary sequence may be expressed as:

$$S_o = T_c \sum_{i=1}^{N_c} f_i w_i \quad (2.17)$$

Where  $N_c$  is the types of character within the source file.

$f_i$  frequency or the probability of occurrence of the  $i^{th}$  character.

$w_i$  number of bits representing the  $i^{th}$  character.

$T_c$  total number of characters within the source file (size of file in Bytes).

$S_o$  is the length of the binary sequence generated in bits.

Another coding format was introduced and investigated in (Al-Bahadili & Rababa'a, 2010), namely, the adaptive coding format. In adaptive coding, first, the character frequencies are calculated and sorted out in descending order from the most common character to the least, similar to Huffman coding. Second, the most common character is given a 0 sequence number, while the least common character is given  $N_c-1$  sequence number. Then, each character is coded to binary according to its sequence number. For example, the equivalent binary codes for the most (first), second, and the third characters are 0000000, 0000001, and 0000010, respectively.

This form of coding ensures a low entropy binary sequence, therefore, it can provide a higher compression ratio and so lower bpc is required to represent characters within the compressed file. In order not to get the data mixed up during the decompression phase, the number of the sorted characters and the characters themselves should be included in the compressed file header. This of course will add an overhead of not more than 129 bytes. It is clear that this overhead is small as compared to the size of the data file.

Table (2.7) presents the binary codeword of the 10 most common characters in the paper1 text file from the Calgary corpus using ASCII and adaptive coding formats. It is clear that the entropy using adaptive coding will be lower than using ASCII coding as the number of 0s will overwhelm the number of 1s in the binary sequence.

#	Character	Frequency	ASCII coding		Adaptive Coding	
			Decimal	Binary	Decimal	Binary
1	Space	7301	32	0100000	0	0000000
2	e	4689	101	1100101	1	0000001
3	t	3048	116	1110100	2	0000010
4	i	2879	105	1101001	3	0000011
5	o	2568	111	1101111	4	0000100
6	n	2503	110	1101110	5	0000101
7	a	2441	97	1100001	6	0000110
8	s	2374	115	1110011	7	0000111
9	r	2058	114	1110010	8	0001000
10	l	1593	108	1101010	9	0001001

The entropies of the binary sequences generated for a number of text files from the Calgary corpus, using different coding formats (e.g., ASCII coding, Huffman coding, and adaptive coding) are compared in Table (2.8). The results obtained reveal that the adaptive coding has the minimum entropy as compared to the others, therefore we expect to achieve higher compression ratio by using this coding format.

#	File Name	Size (Byte)	$N_c$	Entropy		
				ASCII	Huffman	Adaptive
1	bib	111261	81	0.999717	0.996537	0.859069
2	book1	768771	82	0.999438	0.995545	0.793412
3	book2	610856	96	0.999078	0.996209	0.818262
4	paper1	53161	95	0.999482	0.996193	0.834678
5	paper2	82199	91	0.998521	0.995549	0.801658
6	paper3	46526	84	0.997612	0.996422	0.810520
7	paper4	13286	80	0.999019	0.995512	0.809544
8	paper5	11954	91	0.999971	0.996128	0.824279
9	paper6	38105	93	1.000000	0.996432	0.835081

## 2.4 Review of Bit-Level Text Compression Algorithms

(Al-Bahadili, 2008) developed a lossless bit-level data compression algorithm based on the error correcting Hamming codes, and consequently it was referred to as the HCDC algorithm, which was early described in details in Section 2.1. The theoretical analysis of the algorithm indicates that the algorithm reserved a tremendous potential and as a result of that it has been used by many researchers in many applications.

(Al-Bahadili & Rababa'a, 2010) investigated the performance of the HCDC algorithm for text compression application. They developed a scheme that utilizes the algorithm, which consists of six steps, some of which are applied repetitively to enhance the compression ratio of the algorithm, which is called the HCDC( $k$ ) scheme, where  $k$  refers to the number of repetition loops. The repetition loop continues until inflation is detected. In this scheme, the overall (accumulated) compression ratio is the multiplication of the compression ratios of the individual loops. The results obtained for the HCDC( $k$ ) scheme demonstrated that the scheme has a higher compression ratio than most well-known text compression algorithms, and also exhibits a competitive performance with respect to many widely-used state-of-the-art software.

(Al-Bahadili & Al-Saab, 2011; Al-Saab, 2011) used the HCDC algorithm in developing a novel Web search engine model, namely, the compressed index-query (CIQ) model. The model incorporates two compression layers both based on the HCDC algorithm and implemented at the back-end processor (server) side of the Web search engine, one layer resides after the Web search engine indexer acting as a second compression layer to generate a double compressed index, and the second layer be located after the query parser for query compression to enable compressed index-query search. The test results demonstrated that the new CIQ model achieves 100% agreement with current uncompressed Web search engine models, a compression ratio of more than 1.3 with compression efficiency of more than 95% (i.e., a reduction in storage requirement of more than 24%), and a speed up factor of more than 1.3 providing a reduction in processing time of more than 24%.

(Amro et al., 2011) used the HCDC algorithm for speech compression in Voice over Internet Protocol (VoIP) applications, in particular, they implemented the HCDC( $k$ ) algorithm with  $k=4$  to achieve a significant compression ratio in the coefficient codebook without any scarification in the original Code-Excited Linear Prediction (CELP) signal quality since compression is lossless.

(Al-Bahadili & Hussain, 2008) developed a bit-level data compression algorithm in which the binary sequence is divided into blocks of  $n$ -bit length, which gives each block possible decimal values between 0 to  $2^n-1$ . The decimal values of all blocks are calculated and if the number of different decimal values ( $d$ ) is  $\leq 256$ , then the binary sequence can be compressed using  $n$ -bit character wordlength. Otherwise, the process continues until finding a value of  $n$  that gives  $d \leq 256$ . Thus, a compression ratio of approximately  $n/8$  can be achieved. They referred to this algorithm as the Adaptive Character Wordlength (ACW) algorithm, since the compression ratio of the algorithm is a function of  $n$ , and abbreviated as ACW( $n$ ) algorithm.

Implementation of the ACW( $n$ ) algorithm highlights a number of issues that may degrade its performance, and need to be carefully resolved, such as: (i) If  $d$  is greater than 256, then the binary sequence cannot be compressed using the associated  $n$ -bit character wordlength, (ii) the probability of being able to compress a binary sequence using  $n$ -bit character wordlength is inversely proportional to  $n$ , and (iii) finding the optimum value of  $n$  that provides maximum compression ratio is a time consuming

process, especially for large binary sequences. In addition, for text compression, converting text to binary using the equivalent ASCII code of the characters gives a high entropy binary sequence, thus only a small compression ratio or sometimes no compression can be achieved.

Later on, (Al-Bahadili & Rababa'a, 2010) developed an efficient implementation scheme to enhance the performance of the  $ACW(n)$  algorithm, and overcome all drawbacks that are mentioned above. In this scheme the binary sequence is divided into a number of subsequences ( $s$ ), each of them satisfies the condition that  $d \leq 256$ , and they referred to it as the  $ACW(n,s)$  scheme. The scheme achieved compression ratios of more than 2 on most text files from most widely used corpora.

(Nofal, 2007) proposed a bit-level files compression algorithm. In this algorithm, the binary sequence is divided into a set of groups of bits, which are considered as minterms representing Boolean functions. Applying algebraic simplifications on these functions reduce in turn the number of minterms, and hence, the number of bits of the file is reduced as well. To make decompression possible one should solve the problem of dropped Boolean variables in the simplified functions. He investigated one possible solution and their evaluation shows that future work should find out other solutions to render this technique useful, as the maximum possible compression ratio they achieved was not more than 10%.

(Jaradat, Irshid, & Nassar, 2006) proposed a file splitting technique for the reduction of the  $n^{\text{th}}$ -order entropy of text files. The technique is based on mapping the original text file into a non-ASCII binary file using a new codeword assignment method and then the resulting binary file is split into several sub files each contains one or more bits from each codeword of the mapped binary file. The statistical properties of the sub files are studied and it was found that they reflect the statistical properties of the original text file, which was not the case when the ASCII code is used as a mapper. The  $n^{\text{th}}$ -order entropy of these sub files was determined and it was found that the sum of their entropies was less than that of the original text file for the same values of extensions. These interesting statistical properties of the resulting sub files can be used to achieve better compression ratios when conventional compression techniques were applied to these sub files individually and on a bit-wise basis rather than on character-wise basis.

(Barr & Asanović, 2006) presented a study of the energy savings possible by lossless compressing data prior to transmission. Because wireless transmission of a single bit may require over 1000 times more energy than a single 32-bit computation. It can therefore be beneficial to perform additional computation to reduce the number of bits transmitted. If the energy required to compress data is less than the energy required to send it, there is a net energy savings and an increase in battery life for portable computers. They demonstrated that with several typical compression algorithms, there was actually a net energy increase when compression was applied before transmission. Reasons for this increase were explained and suggestions were made to avoid it. One such energy-aware suggestion was asymmetric compression, the use of one compression algorithm on the transmit side and a different algorithm for the receive path. By choosing the lowest-energy compressor and decompressor on the test platform, overall energy to send and receive data can be reduced by 11% compared with a well-chosen symmetric pair, or up to 57% over the default symmetric scheme.

(G. Caire & S. Verdu, 2004) presented a new approach to universal noiseless compression based on error correcting codes. The scheme was based on the concatenation of the Burrows-Wheeler block sorting transform (BWT) with the syndrome former of a Low-Density Parity-Check (LDPC) code. Their scheme has linear encoding and decoding times and uses a new closed-loop iterative decoding algorithm that works in conjunction with belief-propagation decoding. Unlike the leading data compression methods, their method is resilient against errors, and lends itself to joint source-channel encoding/decoding; furthermore their method offers very competitive data compression performance.

(Sharieh, 2004) introduced a Fixed-Length Hamming (FLH) algorithm as enhancement to Huffman Coding (HU) to compress text and multimedia files. He investigated and tested these algorithms on different text and multimedia files. His results indicated that the FLH following HU and HU following FLH enhance the compression ratio.

(Irshid, 2001) proposed a very simple and efficient binary run-length compression technique. The technique is based on mapping the non-binary information source into an equivalent binary source using a new fixed-length code instead of the ASCII code. The codes are chosen such that the probability of one of the two binary symbols; say

0, at the output of the mapper is made as small as possible. Moreover, the "all 1's" code is excluded from the code assignments table to ensure the presence of at least one 0 in each of the output codewords. Compression is achieved by encoding the number of 1's between two consecutive 0's using either a fixed-length code or a variable-length code. When applying this simple encoding technique to English text files, they achieve a compression of 5.44 bpc and 4.6 bpc for the fixed-length code and the variable length (Huffman) code, respectively.

(Mahoney, 2000) introduced a fast text compression with neural network model that produces better compression than popular Lempel-Ziv compressors (zip, gzip, compress), and is competitive in time, space, and compression ratio with PPM and Burrows-Wheeler algorithms. The compressor, a bit-level predictive arithmetic encoder using a 2-layer,  $4 \times 10^6$  by 1 network, is fast (about  $10^4$  characters/second) because only 4-5 connections are simultaneously active and because it uses a variable learning rate optimized for one-pass training. He showed that it is practical to use neural networks for text compression in any application that requires high speed.

## **2.5 Review of Syllable/Word-Based Text Compression Algorithms**

(Joaquín Adiego et al., 2007) described a compression model for semi-structured documents, called Structural Contexts Model (SCM), which takes advantage of the context information usually implicit in the structure of the text. The idea is to use a separate model to compress the text that lies inside each different structure type (different XML tag). The intuition behind SCM was that the distribution of all the texts that belong to a given structure type should be similar, and different from that of other structure types. They mainly focused on semi-static models, and test their idea using a word-based Huffman method. This was a standard for compressing large natural language text databases, because random access, partial decompression, and direct search of the compressed collection were possible. This variant is dubbed as SCMHuff, and it is retained those features and improved Huffman's compression ratios.

(Galambos, 2007) discussed the selection of a suitable compression method, which would utilize the semantics, and structure of HTML documents. Their guess was that such a method has the best chance to achieve an optimal level of compression. Three

branches of compression algorithms were discussed: textual, special XML, and a mix of the previous two. Last branch was represented by a XBW algorithm, which combines textual method with a method for XML compression.

(Conley & Klein, 2006) introduced the notion of multilingual-text compression. The basis of multilingual-text compression is first the ability to match the corresponding parts of related texts by identifying semantic correspondences across the various sub-texts, a task generally referred to as text alignment. Some methods for detailed alignment use an existing multilingual glossary, but all of them generate their own probabilistic glossary, which corresponds to the processed text. The idea is to save storage space by replacing words and phrases with pointers to their translations, determined by any alignment algorithm. Unaligned words are compressed on their own using HuffWord encoding. The suggested method was tested on an English-French corpus of the European Union. They obtained a compression ratio of 22%, which is similar to the performances of Bzip and HuffWord and better than Gzip.

(Rein, Gühmann, & Fitzek, 2006) proposed a lossless compression method for short data series (larger than 50 Bytes). The method uses arithmetic coding and context modeling with a low-complexity data model. A data model that takes 32 KBytes of RAM already cuts down the data size in half. The compression method just takes a few pages of source code, is scalable in memory size, and may be useful in sensor or cellular networks to spare bandwidth. S. Rein et. al demonstrated that their method allows for battery savings when applied to mobile phones. Further work on very short text files compression can be found in (Rein, 2006; Lansky & Zemlicka, 2006; Lansky & Zemlicka, 2005).

(Robert & Nadarajan, 2006) developed a few algorithms for random access text compression in which there is a direct access to the compressed data, so that it is possible to start decompression from any place in the compressed file. If any byte changed during transmission, the remaining data can be retrieved safely. Their work based on the Byte Pair Encoding (BPE) Scheme. The BPE algorithm based on the fact that ASCII character set uses only codes from 0 through 127. That frees up codes from 128 through 255 for use as pair codes. Pair code is a byte, used to replace the most frequently appearing pair of bytes in the text file. Five algorithms are developed based on this BPE scheme. These algorithms find the unused bytes at each level and

tries to use those bytes for replacing the most frequently used bytes. These algorithms compresses typical text files approximately half of their original size, but of course, the actual amount of compression depends on the data being compressed. In these algorithms, most of the time is spent on searching for the most frequently occurring pairs. However, decompression is very fast of all these algorithms.

(Brisaboa, Fariña, Navarro, & Paramá, 2005) addressed the problem of adaptive compression of natural language text, focusing on the case where low bandwidth is available and the receiver has little processing power, as in mobile applications. Their technique achieves compression ratios around 32% and requires very little effort from the receiver. This tradeoff, not previously achieved with alternative techniques, is obtained by breaking the usual symmetry between sender and receiver dominant in statistical adaptive compression. Moreover, they showed that their technique could be adapted to avoid decompression at all cases where the receiver only wants to detect the presence of some keywords in the document. This is useful in scenarios such as selective dissemination of information, news clipping, alert systems, text categorization, and clustering. The asymmetry they introduced, enable the receiver to search the compressed text much faster than the plain text. This was previously achieved only in semi-static compression scenarios. They improved the existing results on word-based adaptive compression, focusing on reducing the effort of the receiver in order to either uncompressed or search for the compressed text.

(Moffat & Isal, 2005) enhanced the performance of the block-sorting algorithm, which is an innovative compression mechanism introduced by Burrows and Wheeler. It involves three steps: permuting the input one block at a time using the Burrows–Wheeler transform (BWT), applying a move-to-front (MTF) transform to each of the permuted blocks, and then entropy coding the output with a Huffman or arithmetic coder. Block-sorting implementations assumed that the input message is a sequence of characters. They extended the block-sorting mechanism to word-based models. They also considered other transformations, and were able to show improved compression results compared to MTF and uniform arithmetic coding. For large files of text, the combination of word-based modeling, BWT, and MTF-like transformations allowed excellent compression effectiveness to be attained within reasonable resource costs.

(I.H. Witten, 2004) showed that the text mining is about inferring structure from sequences representing natural language text, and may be defined as the process of analyzing text to extract information that is useful for particular purposes. Although handcrafted heuristics are a common practical approach for extracting information from text, a general, approach requires adaptive techniques. He studied the way in which the adaptive techniques to use a text compression approach in text mining. He developed several examples: extraction of hierarchical phrase structures from text, identification of key phrases in documents, locating proper names and quantities of interest in a piece of text, text categorization, word segmentation, acronym extraction, and structure recognition. He concluded that compression forms a sound unifying principle that allows many text-mining problems to be tackled adaptively.

Presented a new Huffman coding and decoding technique in which there is no need to construct a full size Huffman table in this technique; instead, the symbols were encoded directly from the table of code-lengths. For decoding purposes a new Condensed Huffman Table (CHT) was also introduced. It was shown that by employing this technique both encoding and decoding operations became significantly faster, and the memory consumption became much smaller compared to the normal Huffman coding/decoding.

(Chu, 2002) presented a new universal lossless data compression algorithm derived from the popular and widely used LZ77 family. He referred to it as LZAC. The objective of LZAC was to improve the compression ratios of the LZ77 family while retaining the family's key characteristics: simple, universal, fast in decoding, and economical in memory consumption. LZAC presented two new ideas: composite fixed-variable-length coding and offset difference coding. A composite fixed-variable-length coding combined fixed-length coding and variable-length coding into a single coding scheme.

(L. G. Rueda & Oommen, 2001) present an enhanced version of the static Fano method, namely Fano+. They formally analyzed Fano+ by presenting some properties of Fano trees, and the theory of list rearrangements. The enhanced algorithm achieved compression ratios arbitrarily close to those of Huffman's algorithm. Empirical results on files of the Canterbury corpus corroborate the almost-optimal efficiency of the enhanced algorithm and its canonical nature.

(Plantinga, 2006) proposed a heuristic for text compression via diagram replacement and a fast entropy coding method. The resulting compression algorithm is an asymmetric algorithm, in the sense that compression requires much time and memory, but decompression is fast and requires little memory. The algorithm is also classified as a semi-adaptive, since it allows a random access into the compressed file without decompressing the whole file. Compression ratios achieved are competitive with gzip for a standard corpus of texts, and better for large files. The algorithm is well suited to applications for which expensive compression of large files is acceptable but decompression must be inexpensive and high compression ratios or random access into the compressed file is required.

## Chapter Three

### The Enhanced HCDC Algorithm

The Hamming Codes Data Compression (HCDC) algorithm (Al-Bahadili, 2008), which was discussed in details in Chapter 2, has been used successfully in many lossless bit-level data compression applications, such as: English text compression (Al-Bahadili & Rababa'a, 2010), high performance Web search engine development (Al-Bahadili & Al-Saab, 2010), and speech compression for VoIP applications (Amro et al., 2011).

However, theoretical analysis of the performance of the algorithm also demonstrated that the algorithm has a maximum compression ratio ( $C_{max}$ ), which is a function of the number of parity bits ( $p$ ) used in the Hamming codewords. For example,  $C_{max} \approx 1.4$  when  $p=3$ , and  $p=3$  is the most suitable value for text compression (Al-Bahadili & Rababa'a, 2010), while  $p=4$  for speech compression (Amro et al., 2011).

The main objective of the work is to enhance the compression ratio of the HCDC algorithm. In particular, in order to enhance the performance (compression ratio) of the HCDC algorithm, a new pre-fix encoding scheme is developed. The new pre-fix encoding scheme is referred to as the  $m$ -encoding scheme. In this encoding scheme, different pre-fix bit combinations are used depending on the types of characters presented in the text file and frequencies of these characters. This is in replacement of the "0" or "1" bit pre-fix encoding scheme of the HCDC algorithm. The enhanced version of the HCDC algorithm is referred to as the Enhanced HCDC (E-HCDC) algorithm, which is expected to achieve a higher compression than the original HCDC algorithm.

Section 3.1 discusses the limitations of the HCDC algorithm, in particular the pre-fix encoding scheme and how it limits the maximum compression ratio. Section 3.2 presents a description of the new pre-fix encoding scheme, namely, the  $m$ -encoding scheme. The E-HCDC algorithm and the structure of the compressed file header are described in details in Section 3.3. The analytical analysis of the compression ratio of the E-HCDC algorithm is presented in Section 3.4.

### 3.1 Limitations of the HCDC Algorithm

In the implementation of the HCDC algorithm for English text compression, first, the text file characters' frequencies ( $f_i$ ) are calculated and sorted out from the most common to the least common characters. Then, the first 16 most common characters (which are usually 7-bit text characters) are replaced with valid Hamming codewords, while all other characters are considered as a non-valid codewords. The characters are treated as 7-bit Hamming codewords, in this case,  $n=7$ ,  $d=4$ ,  $p=3$ , where  $n$  is the length of the Hamming codeword, and  $d$  and  $p$  are the number of data and parity bits in the Hamming codeword.

This means that the first 16 most common characters can be replaced with 5-bit instead of the original uncompressed 7-bit (i.e.,  $d+1$  bits, where one bit is added usually "0" as prefix bit). While all remaining characters are replaced with 8-bit instead of the original uncompressed 7-bit (i.e.,  $n+1$  bits, where one bit is added usually "1" as prefix bit). These "0" and "1" bits are added to be used during the decompression process to distinguish the valid from the non-valid codewords.

### 3.2 The $m$ -encoding Scheme

It can be seen from the above discussion that we can achieve compression only when the sum of valid codewords for all first 16 most common characters is greater than the sum of the non-valid codewords for all remaining characters. Furthermore, if the number of different characters in the text file is less than or equal to 16 characters, then all characters can be replaced with 5-bit instead of 7-bit and a maximum compression ratio can be achieved which in this case 1.4 if the size of the compressed file header is neglected.

Furthermore, in practical English text compression applications, the average number of text characters within any text file is more than 60 characters, especially for medium and large size text files (Sayood, 2012; Salomon, 2004). This means it will be very difficult to achieve compression for files with flat (equal) characters frequencies or sometimes reduces the compression ratio, as for most characters we added extra bit (8-bit).

Therefore, in order to enhance the compression ratio and also to solve the problem with relatively flat characters frequencies, we develop a new pre-fix encoding scheme, which we call the m-encoding scheme. In this scheme, after finding the characters frequencies and sorting them from the most common to the least common character, we divide them into groups of 16 characters each starting from the most common character, which means the last group may contain 16 or less than 16 characters. Since, the maximum number of text character is 96 characters as we discussed in Chapter 2, the maximum number of groups is 8 numbered from  $G_1$  to  $G_8$ . The characters for each group are numbered from 0 to 15 and assigned a binary code from 0000 to 1111.

Then, for each group, we will assign different combination of pre-fix bits. For example, the pre-fix of  $G_1$  is "0" as it was in the original HCDC algorithm. The pre-fix bit for the other groups are 10, 110, 1110, and so on until the last group which will have all 1's. Thus, we will have 7 1's for  $G_8$ . For each group we will add 4-bit representing the sequence number of the character within the group. The pre-fix bits and the numbers of bits for each group are summarized in Table (3.1).

It can be seen from the above table that we are reducing the number of bits written to the compressed data file to 6-bit and 7-bit for  $G_2$  and  $G_3$  in comparison to 8-bit for the HCDC algorithm. While it remained unchanged with 8-bit for  $G_4$ , and start increasing by 1-bit for each of the remaining groups until it reaches 11 for  $G_8$ .

It is clear from the above discussion that we need more bits to represent the characters in the last four groups ( $G_5$  to  $G_8$ ), but, fortunately, the characters frequencies within these groups are small in comparison to that in the top four groups ( $G_1$  to  $G_4$ ). Furthermore, most of the time we have the characters span over not more than 6 groups, which means the maximum number of bits required to represent the compressed characters is 9 bits, because the number of pre-fix bits for the last group is equal the number of bits for previous one but will pre-fix bits set to 1's. This form of pre-fix encoding guarantees a higher compression ratio than for the HCDC algorithm even for files with flat characters frequencies distribution. However, in this work, to improve the performance further, we introduce a modification to the above continuous encoding.

Table 3.1 The pre-fix bits and the number of bits written to the compressed data file.							
Group ( $G$ )	Character ( $E$ )	Pre-fix bits	No. of Pre-fix bits	Character Code	Compressed Bits	No. of Compressed Bits	
						E-HCDC	HCDC
1	1	0	1	0000	00000	5	5
	:	0		:	:		
	:	0		:	:		
	16	0		1111	01111		
2	17	10	2	0000	100000	6	8
	:	10		:	:		
	:	10		:	:		
	32	10		1111	101111		
3	33	110	3	0000	1100000	7	8
	:	110		:	:		
	:	110		:	:		
	48	110		1111	1101111		
4	49	1110	4	0000	11100000	8	8
	:	1110		:	:		
	:	1110		:	:		
	64	1110		1111	11101111		
5	65	11110	5	0000	111100000	9	8
	:	11110		:	:		
	:	11110		:	:		
	80	11110		1111	111101111		
6	81	111110	6	0000	1111100000	10	8
	:	111110		:	:		
	:	111110		:	:		
	96	111110		1111	1111101111		
7	97	1111110	7	0000	11111100000	11	8
	:	1111110		:	:		
	:	1111110		:	:		
	112	1111110		1111	11111101111		
8	113	1111111	7	0000	11111110000	11	8
	:	1111111		:	:		
	:	1111111		:	:		
	128	1111111		1111	11111111111		

In the modified encoding, we limit the number of pre-fix bits to a limited number of bits ( $m$ ). Therefore, we call this pre-fix encoding as  $m$ -encoding. Before, we proceed with the description of this new encoding; let us define some parameters. The first one is the number of groups ( $G$ ), which is calculated as:

$$G = \left\lceil \frac{N_c}{2^d} \right\rceil \quad (3.1)$$

Where  $G$  is the number groups,  $N_c$  is the number of character, and  $d$  is the number of data bits in Hamming codeword.

Second, the number of characters within the last group  $L_g$ , which can be calculated by using the following equation:

$$L_g = N_c - 2^d \times (G-1) \quad (3.2)$$

Third, the number of bit ( $b_t$ ) required to represent the characters within the last group ( $t$ ), which can be calculated as:

$$b_t = \left\lceil \frac{\ln(L_t)}{\ln(2)} \right\rceil \quad (3.3)$$

When  $m$  is selected to be 4 bits, then the pre-fix combinations are 0 for  $G_1$ , 10 for  $G_2$ , 110 for  $G_3$ , 1110 for  $G_4$ , and 1111 for the remaining group(s). So that the number of bits for the last groups is calculated using Eqn. (3.3), with  $L_t$  represents the number of characters in the last group, or it can be calculated by:

$$L_t = N_c - m \times 2^d \quad (3.4)$$

Table (3.2) shows the pre-fix bits combinations, the characters codes, and the compressed bits written to the compressed file for a file contains 96 characters. It is clear from the above discussion that the characters in  $G_5$  and  $G_6$  are merged together forming a group of 32 characters requiring 5-bit to represent each character from character 1 (00000) to character 31 (11111).

The pre-fix bits are 4-bits (1111), and then each character will be written as 9-bit character to the compressed file. However, if  $N_c \leq 80$  and  $N_c > 64$ , then  $1 < L_t \leq 16$ , and in this case  $b_t$  could be  $\leq 4$ , which means the number of compressed bits written to the compressed file is  $\leq 8$ .

Table 3.2 The pre-fix bits and the number of bits written to the compressed data file.							
Group ( $G$ )	Character ( $E$ )	Pre-fix bits	No. of Pre-fix bits	Character Code	Compressed Bits	No. of Compressed Bits	
						E-HCDC	HCDC
1	1	0	1	0000	00000	5	5
	:	0		:	:		
	:	0		:	:		
	16	0		1111	01111		
2	17	10	2	0000	100000	6	8
	:	10		:	:		
	:	10		:	:		
	32	10		1111	101111		
3	33	110	3	0000	1100000	7	8
	:	110		:	:		
	:	110		:	:		
	48	110		1111	1101111		
4	49	1110	4	0000	11100000	8	8
	:	1110		:	:		
	:	1110		:	:		
	64	1110		1111	11101111		
5	:	65	4	00000	111100000	9	8
		1111		:	:		
		1111		:	:		
		1111		:	:		
		1111		:	:		
		1111		:	:		
6	:	96	1111	11111	111111111		

One important feature of the  $m$ -encoding scheme is that for  $m=1$ , the scheme behaves exactly the same as in the original HCDC algorithm, where in this case the first 16 most common characters ( $G_1$ ) is preceded by 0, while for all other characters (groups) are preceded by 1. However, there is only one difference here, which is in the original HCDC, all 7-bit character is appended to the compressed file, while here, only  $b_i$  bits are appended, and  $b_i$  is calculated depending on the sequence number of the character excluding the first 16 most common characters. In this case, we may need less than 7-bit to represent characters in the compressed file, which provides further saving and increasing the compression ratio.

Finally, in this section, one other important feature of the  $m$ -encoding scheme is that users do not need to specify  $m$ , and they can leave the code to calculate the optimum value of  $m$  depending on the characters frequencies or characters counts. Using the  $m$ -encoding scheme, the size of the compressed binary sequence can be calculated as:

$$Q_b = (d + 1) \sum_{i=1}^{2^d} T_i + \sum_{j=2}^m (d + j) \sum_{i=1+(j-1) \times 2^d}^{j \times 2^d} T_i + (b_t + m) \sum_{i=1+m \times 2^d}^{N_c} T_i \quad (3.5)$$

Where

- $Q_b$  length of the compressed binary sequence in bits.
- $d$  number of data bits in the Hamming codeword.
- $T_i$  counts for character  $i$  ( $i=1$  to  $N_c$ ).
- $m$  maximum length for the pre-fix bits.
- $b_t$  number of bits representing the characters of the last group.
- $N_c$  total number of characters within the text file.

So that the code can calculate the values of  $Q_b$  for all possible values of  $m$ , for example, for 7-bit Hamming codeword, the maximum possible value for  $m$  is 7. Then, the code can select the value of  $m$  that gives the minimum  $Q_b$  to ensure maximum compression ratio. However, until this moment we have not discussed the overhead bit added for the compressed file header, which will be discussed in the next section.

### 3.3 The E-HCDC Algorithm

This Section presents the detail description of the E-HCDC algorithm for text compression. In order to improve the compression ratio of the HCDC algorithm, we utilize the concept of the pre-fix  $m$ -encoding scheme described in the previous section instead of the original 0/1 prefix-encoding. The algorithm is developed into two algorithms, one is the compression algorithm (E-HCDC compressor), and the other one is the decompression algorithm (E-HCDC decompressor), which are described below.

#### 3.3.1 The E-HCDC Compressor

The E-HCDC compressor consists of the following steps:

- (1) Read-in the input uncompressed text file

- (2) Find the characters within the uncompressed file ( $U_i$ ); and calculate the number of characters within the uncompressed text file ( $N_c$ ), characters counts ( $T_i$ ), and characters frequencies ( $f_i$ ), where  $i=1$  to  $N_c$ .
- (3) Sort out the characters from the most common to the least common.
- (4) Determine the optimum value for  $m$  as described in the previous section and Eqn. (3.5).
- (5) Initialize the compressed binary sequence to Null.
- (6) Start the compression process by reading in one character at a time until processing all characters; find the character sequence number, the group it's belonging to, and its sequence within the group. According to the value of  $m$  and the group it's belonging to, append the pre-fix bits to the compressed binary sequence, find the character's equivalent binary representation based on the associated character wordlength ( $d$  or  $b_i$  depending on  $m$  and  $G$ ), and then append the character binary representation to the compressed binary sequence.
- (7) In order to be able to convert the resultant compressed binary sequence ( $Q_b$ ) to character sets (8-bit character length), and because  $Q_b$  may not be a multiple of 8, then we append padded bits (say  $a$  0's) at the end of the compressed binary sequence, and the length of the padded bits ( $a$ ) need to be stored at the compressed file header, so these bits can be discarded during the decompression phase. The number of padded bits ( $a$ ) can be calculated as:

$$a = 8 - Q_b \text{ Mod } 8 \quad (3.6)$$

- (8) Initialize the compressed text sequence to Null.
- (9) Construct the compressed file header and append it to the compressed text sequence, which should contain all information necessary during the decompression process. The compressed file header will be described in the next section.

- (10) Convert the compressed binary sequence to characters by reading-in 8-bit at a time until converting all compressed binary sequence. For each 8-bit, find its equivalent ASCII code, and then append the equivalent character to the compressed text sequence.
- (11) Write the compressed text sequence to a file (compressed file).

In this case the size of the compressed file ( $S_c$ ) is given by:

$$S_c = H + \left\lceil \frac{Q_b}{8} \right\rceil \quad (3.6)$$

Where  $H$  is the length of the compressed file header. The compression ratio can be easily calculated as  $C=N_c/S_c$ . Figure (3.1) outlines the procedure for the E-HCDC compressor.

```

// The procedure of the E-HCDC compressor.
//
Read-in the input uncompressed text file
Find the uncompressed characters ( $U_i$ ) within the file; and calculate the number of characters within the
text file ( $N_c$ ), characters counts ( $T_i$ ), and characters frequencies ( $f_i$ ), where  $i=1$  to  $N_c$ .
Sort the characters from the most common to the least common.
Determine the optimum value for  $m$ .
Initialize the compressed binary sequence ( $Q_b$ ) to Null ( $Q_b=Null$ ).
Start the compression process
    Read-in one character at a time until processing all characters
        Find the character sequence number ( $s$ )
        Find the group to which the character is belonging to ( $g$ )
        Find the sequence number of the character within the group ( $q$ )
        Append the pre-fix bits to the compressed binary sequence according to  $m$  and  $g$ 
        Find the character's equivalent binary representation based on the associated character
        wordlength ( $d$  or  $b_i$ , depending on  $m$  and  $g$ )
        Append the character binary representation to the compressed binary sequence
    Append padding bits to ensure ( $Q_b$ ) is a multiple of 8-bit.
    Convert  $Q_b$  to characters of 8-bit character wordlength
    Initialize the compressed text sequence to Null.
    Construct the compressed file header and append it to the compressed text sequence
    Convert the compressed binary sequence to characters
        Read-in 8-bit at a time until converting all compressed binary sequence
            Find its equivalent ASCII code
            Find equivalent character
            Append the character to the compressed text sequence
    Write the compressed text sequence to a file (compressed file).

```

Figure (3.1). The procedure of the E-HCDC compressor.

### 3.3.2 The E-HCDC Decompressor

For each compression algorithm, there should be an associated decompression algorithm to ensure a pre-specified data original data retrieval, either exact as in lossless data compression (e.g., text compression) or approximate as in lossy data compression (e.g., multimedia compression). In this work, we concern with text compression, thus we are in need of data decompressor to work in association with the E-HCDC compressor presented above, which will be called the E-HCDC decompressor.

The E-HCDC decompressor consists of the following steps:

- (1) Read-in the input compressed file
- (2) Extract the file header, and find-out the number of characters within the original uncompressed text file ( $N_c$ ), the uncompressed characters themselves ( $U_i$ ) (where  $i=1$  to  $N_c$ ), the maximum length of the pre-fix bits ( $m$ ), and the number of padded bits ( $a$ ).
- (3) Determine the number of groups ( $G$ ) as  $G=m+1$ , the number of characters in the last group ( $L_t$ ), and the number of bits to represent the characters of the last group ( $b_t$ ).
- (4) Initialize the compressed binary sequence ( $Q_b$ ) to Null.
- (5) Initialize the decompressed text sequence ( $D_t$ ) to Null.
- (6) Extract the compressed text sequence and read-in it one character at a time, convert each character to a binary representation according to its ASCII code, and append the resultant binary representation to the compressed binary sequence ( $Q_b$ ), until all characters are read-in.
- (7) Discard the appended bits ( $a$ ).
- (8) Start the decompression process by reading in 1-bit, if this bit is 0, then the character belongs to  $G_1$  (i.e.,  $g=1$ ), read-in the next 4-bit, find-out the equivalent decimal value ( $e$ ) between 0 to 15, which represents the sequence number of the character ( $q$ ) within  $G_1$  minus 1 ( $q=e+1$ ), determine the global

sequence number ( $s$ ) as:

$$s = q + 2^d (g - 1) = 1 + e + 2^d (g - 1) \quad (3.7)$$

Then, retrieve the character with sequence number  $s$  ( $U_s$ ) from the uncompressed characters lists and append it  $D_t$ .

If the read-in 1-bit is 1 (count the number of 1's ( $z$ )), then reads-in another bit until a 0 is read-in or read-in a maximum of  $m$  1's. In this case, the group number ( $g$ ) is equal to  $z+1$  as long as  $z+1 < m$ , otherwise it is equal to  $m$ , as illustrated in the following equation:

$$g = \begin{cases} z + 1, & z + 1 < m \\ m + 1, & \text{otherwise} \end{cases} \quad (3.8)$$

If  $g$  is not the last group ( $g < G$ ), then read-in 4-bit, otherwise ( $g$  is the last group  $g = G$ ) read-in  $b_t$ -bit, and perform the same steps above and append another character to  $D_t$ . This process continues until reading all compressed binary sequence.

- (9) Write the decompressed text sequence ( $D_t$ ) to a file (uncompressed file).

Figure (3.2) outlines the procedure for the E-HCDC decompressor.

It can be clearly seen from the above compressor and decompressor procedures of the E-HCDC algorithm that the E-HCDC algorithm is:

- (1) Bit-level. It processes the data at a binary level.
- (2) Lossless. An exact form of the source file is retrieved.
- (3) Adaptive. The character-binary coding depends on the characters frequencies.
- (4) Asymmetric. The compression processing time is higher than the decompression processing time.

```

// The procedure of the E-HCDC decompressor.
//
Read-in the input compressed file.
Extract the file header and find-out the following:
    The number of characters within the original uncompressed text file ( $N_c$ )
    The uncompressed characters themselves ( $U_i$ ) (where  $i=1$  to  $N_c$ .)
    The maximum length of the pre-fix bits ( $m$ )
    The number of padded bits ( $a$ )
Determine the following:
    The number of groups ( $G$ ) as  $G=m+1$ 
    The number of characters in the last group ( $L_l$ )
    The number of bits to represent the characters of the last group ( $b_l$ )
Initialize the compressed binary sequence ( $Q_b$ ) to Null.
Initialize the decompressed text sequence ( $D_t$ ) to Null
Extract the compressed text sequence and read-in it one character at a time until all characters are read-
in
    Convert each character to a binary representation according to its ASCII code
    Append the resultant binary representation to the compressed binary sequence ( $Q_b$ )
Discard the appended bits ( $a$ )
Start the decompression process
Do
    Read in 1-bit
    If this bit is 0
        The character belongs to  $G_1$  (i.e.,  $g=1$ )
        Read-in the next 4-bit
        Determine the equivalent decimal value ( $e$ ) between 0 to 15
        Determine the sequence number of the character ( $q$ ) within  $G_1$  ( $q=e+1$ )
        Determine the global sequence number ( $s$ )
        Retrieve the character with sequence number  $s$  ( $U_s$ ) from the uncompressed characters lists
        Append  $U_s$  to  $D_t$ 
    Else
        Count the number of 1's ( $z$ ) ( $z=z+1$ ) (initially  $z=0$ )
        Reads-in another bit until a 0 is read-in or read-in a maximum of  $m$  1's
        Set the group number ( $g$ ) to  $z+1$  as long as  $z+1 < m$ , otherwise  $g=m$ 
        If  $g$  is not the last group ( $g < G$ )
            Read-in 4-bit
        Else ( $g$  is the last group  $g=G$ )
            Read-in  $b_l$ -bit
        End If
        Determine the equivalent decimal value ( $e$ ) between 0 to 15 or 0 to  $2^{b_l}-1$ 
        Determine the sequence number of the character ( $q$ ) within its group
        Determine the global sequence number ( $s$ )
        Retrieve the character with sequence number  $s$  ( $U_s$ ) from the uncompressed characters lists
        Append  $U_s$  to  $D_t$ 
    End If
Until reading-in all compressed binary sequence
Write the decompressed text sequence ( $D_t$ ) to a file (uncompressed file).

```

Figure (3.2). The procedure of the E-HCDC decompressor.

### 3.4 The E-HCDC Header

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

- (a) HCDC field
- (b) Text characters field

These two fields will be followed by the text compressed data. In what follows brief description is given for these two fields.

#### (a) HCDC field

The HCDC field of the E-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 (character) in the original text file ( $N_c$ ), 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, 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 E-HCDC algorithm is designated as Version 3. A version under development called AQ-HCDC for adaptive-quality image compression allocated as Version 2. The coding format ( $F$ ) is set to 0 for ASCII coding, 1 for Huffman coding, 2 for adaptive coding, etc. Table (3.3) lists the components of this field and their description.

#### (b) Text Characters field

The text characters field is designed to include minimum information, it encloses information related to:

- (1) The maximum length for the pre-fix bits ( $m$ ), which is accommodated at the left 4-bit of the 9<sup>th</sup> character.

- (2) The number of padded bits ( $a$ ), which is accommodated at the right 4-bit of the 9<sup>th</sup> character.
- (3) The number of text characters ( $N_c$ ), which is accommodated at the 10<sup>th</sup> character.
- (4) The actual text characters ( $U_i$ ) sorted out from the most common to the least common, which are accommodated at the 11<sup>th</sup> character up to the  $(11+N_c)$ <sup>th</sup> characters.

Figure (3.3) shows the structure of the E-HCDC compressed file header. It can be seen that the total length has a maximum size of 138 Bytes, and generally it is given by:

$$H = 10 + N_c \quad (3.9)$$

The 10 Bytes in Eqn. (3.9) is the sum of 8-Byte for the HCDC field, 1-Byte for  $m$  and  $a$ , and 1-Byte for  $N_c$ , which is stored twice for consistency purposes with previous versions.

Components	Length (Byte)	Description
HCDC	4	Name of the compression algorithm.
$V$	1	Version of the HCDC algorithm.
$N_c$	1	Number of symbols (text characters) within the original text 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.

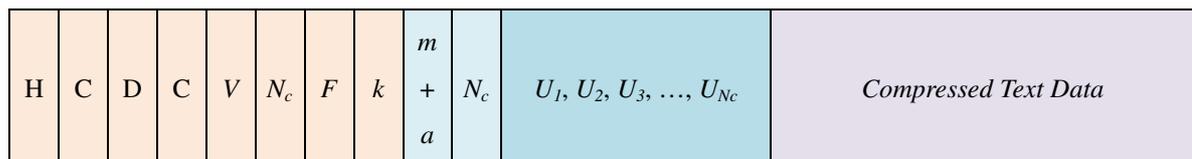


Figure (3.3). The structure of the E-HCDC compressed file header.

## Chapter Four

### Experimental Results and Discussions

The Enhanced Hamming Codes based Data Compression (E-HCDC) algorithm is implemented using VB.net programming language. The resultant code allows a wide range of investigations and experiments to be performed. However, it is only used to compress a number of text files from standard corpora, namely, Calgary Corpus, Canterbury Corpus, Artificial Corpus, and Large Corpus (Arnold & Bell, 1997; Bell & Witten, 1990; Ian H. Witten et al., 1987), which are described in Appendix A.

At this stage, it is important to indicate that little efforts have been made to optimize the runtime of the compression/decompression prototype code, therefore, in this work, we only compare and show the results for the compression ratio of the algorithm. However, this scheme is classified as an asymmetric bit-level data compression algorithm as the compression processing time is higher than the time required for decompression.

In this thesis, results of three experiments are presented to evaluate and compare the compression ratio of the E-HCDC algorithm ( $C_{E-HCDC}$ ). The first experiment evaluates  $C_{E-HCDC}$  and compares it against the compression ratio of the original HCDC algorithm ( $C_{HCDC}$ ) (Al-Bahadili, 2010), and also estimate the enhancement ratio ( $E_r$ ) achieved by the enhanced version. The other two experiments compare  $C_{E-HCDC}$  against a number of well-known statistical and adaptive algorithms.

The three experiments are summarized below and will be described in the next sections.

- (1) Experiment #1: Comparing the Compression Ratio of the E-HCDC and HCDC Algorithms.
- (2) Experiment #2: Comparing the Compression Ratio of the E-HCDC and a Number of Statistical Algorithms.
- (3) Experiment #3: Comparing the Compression Ratio of the E-HCDC and a Number of Adaptive Algorithms

## 4.1 Experiment #1: Comparing the Compression Ratio of the E-HCDC and HCDC Algorithms

This experiment evaluates  $C_{E-HCDC}$  of the E-HCDC algorithm, and compares it against  $C_{HCDC}$  of the HCDC algorithm for a number of text files from standard corpora, namely, Calgary Corpus, Canterbury Corpus, Artificial Corpus, and Large Corpus, which are briefly described in Appendix A and further details can be found in (Arnold & Bell, 1997; Bell & Witten, 1990; Ian H. Witten et al., 1987).

The results for  $C_{HCDC}$  and  $C_{E-HCDC}$  are listed in Table (4.1) and also plotted in Figure (4.1) for text files from the Calgary Corpus, and in Figure (4.2) for text files from the Canterbury, Artificial, and Large corpora. Table (4.1) lists the following parameters:

- (1) Name and size of the text file.
- (2) Number of characters within the text files ( $N_c$ ).
- (3) The compression ratios  $C_{HCDC}$  and  $C_{E-HCDC}$ .
- (4) The number of groups ( $G$ ) calculated using Eqn. (3.1).
- (5) The maximum number of padded bits ( $m$ ) for the  $m$ -encoding scheme.
- (6) The compression enhancement ratio ( $E_r$ ), which is calculated by:

$$E_r = \frac{C_{E-HCDC} - C_{HCDC}}{C_{HCDC}} \times 100 \quad 4.1$$

Where  $C_{E-HCDC}$  is the compression ratio of the E-HCDC algorithm,  $C_{HCDC}$  is the compression ratio of the HCDC algorithm, and  $E_r$  is the enhancement ratio.

It can be clearly seen from the above results that the E-HCDC algorithm enhances the performance of the HCDC algorithm by **15% to 20%**, which is achieved due to the implementation of  $m$ -encoding.

It is also important to realize that  $C_{HCDC}$  and  $C_{E-HCDC}$  can be calculated analytically as described in Chapter 2 for the HCDC algorithm and in Chapter 3 for the E-HCDC algorithm; after calculating the characters frequencies of the text files. This can be considered as a validation to the accuracy of the coding process.

Table ( 4.1 ) – Experiment #1							
Comparing $C_{HCDC}$ and $C_{E-HCDC}$ .							
#	File Name	File Size (Byte)	$N_c$	$C_{HCDC}$	$C_{E-HCDC}$	G (m) $m=G-1$	$E_r$ (%)
Calgary corpus							
1	bib	111261	81	1.177	1.455	6 (5)	18
2	book1	768771	82	1.269	1.537	6 (5)	19
3	book2	610856	96	1.247	1.509	6 (5)	18
4	paper1	53161	95	1.211	1.484	6 (5)	18
5	paper2	82199	91	1.265	1.531	6 (5)	19
6	paper3	46526	84	1.261	1.525	6 (5)	19
7	paper4	13286	80	1.257	1.52	5 (4)	19
8	paper5	11954	91	1.218	1.486	6 (5)	18
9	paper6	38105	93	1.203	1.477	6 (5)	18
Canterbury corpus							
10	alice29.txt	152089	74	1.258	1.529	5 (4)	19
11	asyoulik.txt	125179	68	1.230	1.508	5 (4)	18
12	lcet10.txt	426754	84	1.252	1.520	6 (5)	19
13	plrabn12.txt	481861	81	1.269	1.537	6 (5)	19
Artificial corpus							
14	alphabet.txt	100000	26	1.138	1.486	2 (1)	18
15	random.txt	100000	64	0.969	1.236	4 (3)	15
Large corpus							
16	bible.txt	4047390	63	1.294	1.551	4 (3)	19
17	world192.txt	2473400	94	1.206	1.471	6 (5)	18

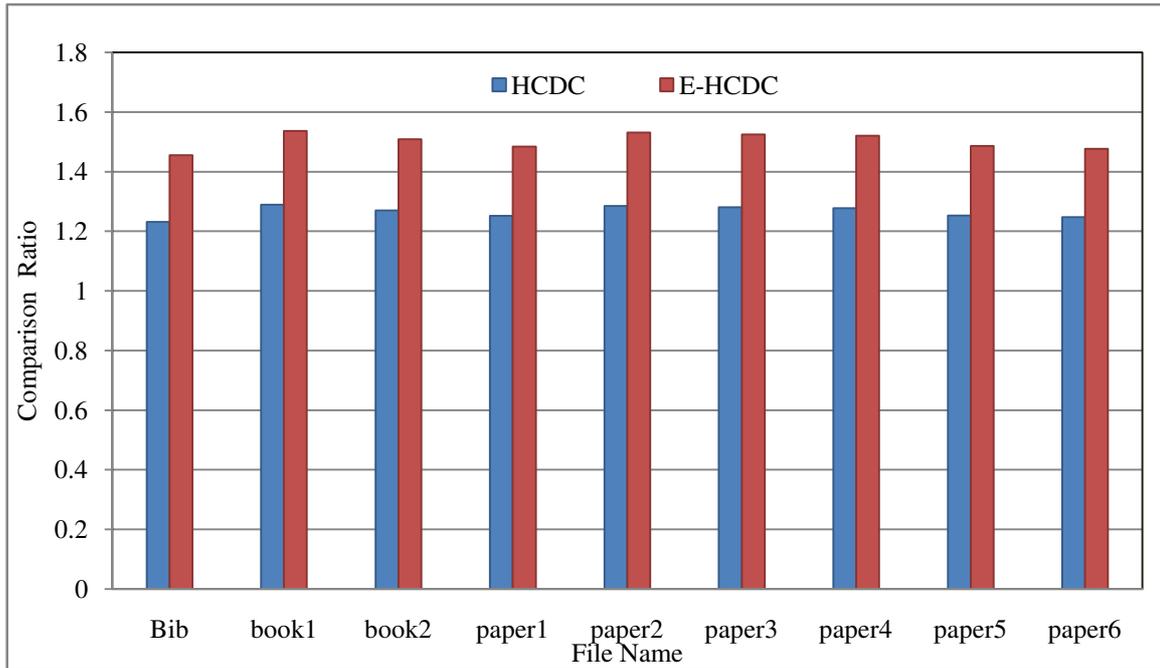


Figure (4.1). Comparing  $C_{HCDC}$  and  $C_{E-HCDC}$  for a number of text files from the Calgary Corpus.

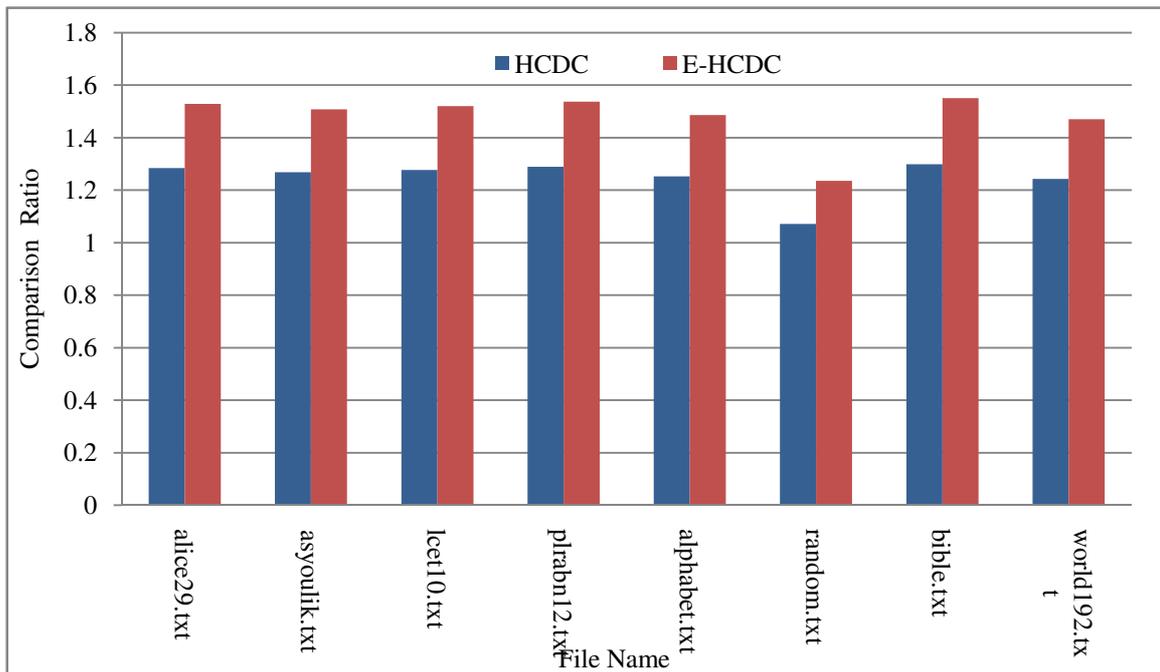


Figure (4.2). Comparing  $C_{HCDC}$  and  $C_{E-HCDC}$  for a number of text files from the Canterbury, Artificial, and Large Corpora.

Statistics of five files are given in tables in Appendix B, precisely; we present the statistics for five text files of different sizes and from different corpora, namely, *alice29.txt*, *bib*, *bible*, *book1*, and *paper1*. The tables list the following:

- (1) Size of the text (uncompressed) file in Bytes ( $S_B$ )
- (2) Size of the text (uncompressed) file in bits ( $S_o$ )
- (3) Size of the compressed file in bits ( $S_c$ )
- (4) Number of characters within the text file ( $N_c$ )
- (5) The maximum number of pre-fix bits ( $m$ ) of the  $m$ -encoding scheme
- (6) List of characters ( $c_i$ ) ( $i=1$  to  $N_c$ )
- (7) Character occurrence (counts) ( $T_i$ ) ( $i=1$  to  $N_c$ )
- (8) Character frequency ( $f_i$ ) ( $i=1$  to  $N_c$ )
- (9) Character ASCII code in binary representation ( $A_i$ ) ( $i=1$  to  $N_c$ )
- (10) Character encoding in the E-HCDC algorithm ( $E_i$ )

The results in Appendix B give us insight into the behavior and the encoding process of the E-HCDC algorithm, and also the tables can be used to calculate the compression ratio analytically.

## 4.2 Experiment #2: Comparing the Compression Ratio of the E-HCDC and a Number of Statistical Algorithms

In this experiment, the  $C_{E-HCDC}$  of the E-HCDC algorithm is compared against a number of statistical data compression algorithms, such as:

- (1) The Huffman coding (HU)
- (2) The fixed-length Hamming (FLH)
- (3) The Huffman coding following the fixed-length Hamming (HF)

- (4) The ACW( $n,s$ ) scheme using adaptive coding (ACW-A)
- (5) The ACW( $n,s$ ) scheme using Huffman coding (ACW-H)
- (6) The HCDC algorithm
- (7) The HCDC( $k$ ) algorithm

The compression ratios are calculated for two text files from the Calgary Corpus, namely, book1 and paper1, where book1 represents a large file example with size 768771 Byte, while paper1 represents a small file with size 53161 Byte. The statistics of these text files are presented in Appendix B. The results are listed in Table (4.2) and also plotted in Figure (4.3).

Table (4.2) – Experiment #2 Comparing $C_{E-HCDC}$ of the E-HCDC algorithm against the compression ratios of various statistical algorithms.		
Algorithm	Book1	Paper1
HU	1.72	1.60
FLH	1.14	1.14
HF	1.71	1.57
ACW-A	1.67 (14)	1.54 (11)
ACW-H	2.67 (11)	2.43 (11)
HCDC	1.27	1.21
HCDC( $k$ )	2.23 (6)	1.66 (4)
E-HCDC	1.54	1.48
HU	Huffman coding	Sharieh, 2004
FLH	Fixed-Length Hamming	
HF	HU following FLH	
ACW-A	Adaptive Character Wordlength with adaptive coding	Al-Bahadili & Hussain, 2010
ACW-H	Adaptive Character Wordlength with Huffman coding	
HCDC	Hamming Codes based Data Compression	Al-Bahadili & Rababa'a, 2010
HCDC( $k$ )	Adaptive HCDC	

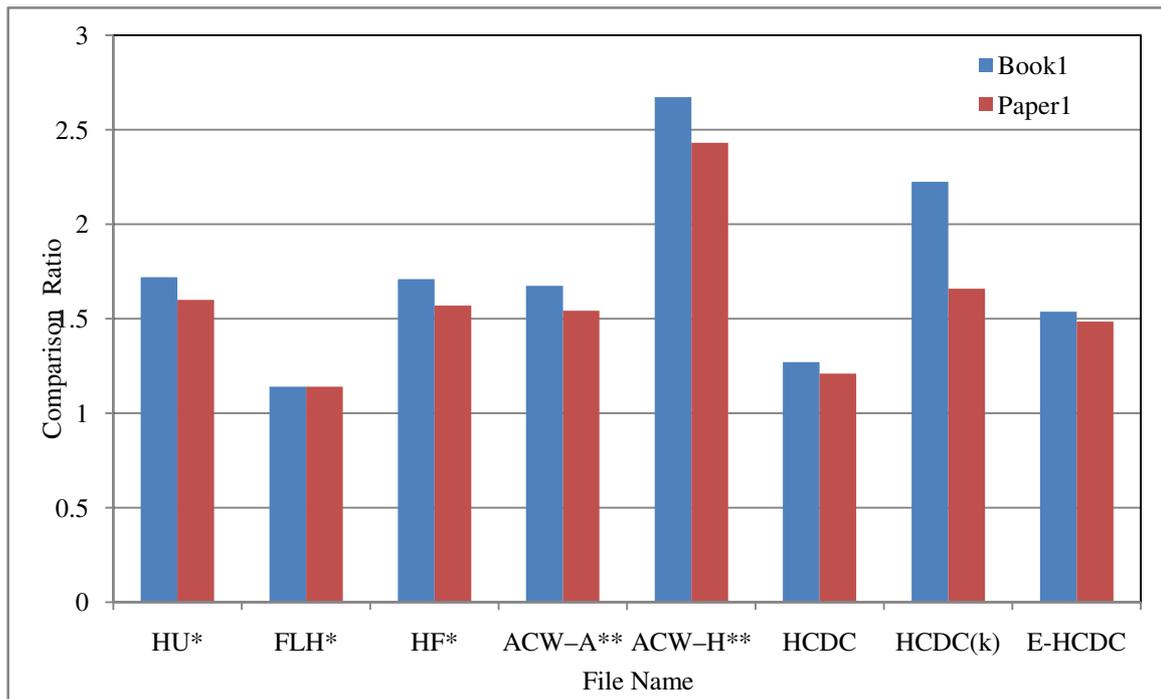


Figure (4.3). Comparing  $C_{E-HCDC}$  of the E-HCDC algorithm against the compression ratio of various statistical algorithms.

The results show that the E-HCDC algorithm achieves a reasonable compression ratio with respect to other algorithms, and it is expected to be significantly improved if the algorithm is repetitively applied, i.e. developing an E-HCDC( $k$ ) algorithm similar to the HCDC( $k$ ) algorithm. The values between brackets for the ACW algorithm represent the compression character wordlength, while the values between brackets for the HCDC represent the number of repetitive loops.

### 4.3 Experiment #3: Comparing the Compression Ratio of the E-HCDC and a Number of Adaptive Algorithms

In this experiment, we compare  $C_{E-HCDC}$  of the E-HCDC algorithm against the compression ratios for a number of adaptive algorithms, namely:

- (1) The Unix compact utility that is based on adaptive Huffman (AH)
- (2) The greedy adaptive Fano coding (AF)

Furthermore, for the sake of comparison, we include results for the HCDC and  $HCDC(k)$  algorithms. The compression ratios are calculated for seven text files from the Calgary and Canterbury Corpora, namely, bib, book1, book2, and paper1 from the Calgary corpus, and alice29, asyoulik, lect10, and plravn12 from the Canterbury Corpus. The statistics of some of these files are presented in Appendix B. The results are listed in Table (4.3) and also plotted in Figure (4.4).

Once again, the  $C_{E-HCDC}$  of the E-HCDC looks very competitive to the compression ratios of the adaptive algorithms, and it is expected to overwhelmed them if the algorithm is repeatedly applied, i.e., developing a new version of the  $HCDC(k)$  that utilizes the m-encoding scheme, which should be very interesting and promising research.

Table (4.3)						
Comparing the compression ratio of the E-HCDC algorithm against various adaptive algorithms.						
Corpus	File Name	AH	AF	HCDC	$HCDC(k)$	E-HCDC
Calgary Corpus	bib	1.526	1.524	1.177	1.428 (4)	1.529
	book1	1.753	1.750	1.269	2.225 (6)	1.508
	book2	1.658	1.653	1.247	1.971 (5)	1.455
	paper1	1.587	1.588	1.211	1.658 (4)	1.537
Canterbury Corpus	alice29	1.753	1.746	1.258	2.097 (5)	1.509
	asyoulik	1.648	1.645	1.230	1.825 (5)	1.520
	lect10	1.718	1.717	1.252	2.009 (5)	1.484
	plravn12	1.769	1.766	1.269	2.235 (6)	1.537
AH	Adaptive Huffman use in Unix compact utility				(Rueda & Oommen, 2006)	
AF	Adaptive Fano uses Greedy adaptive Fano coding					
HCDC	Hamming Codes based Data Compression				Al-Bahadili & Rababa'a, 2010	
$HCDC(k)$	Adaptive HCDC					

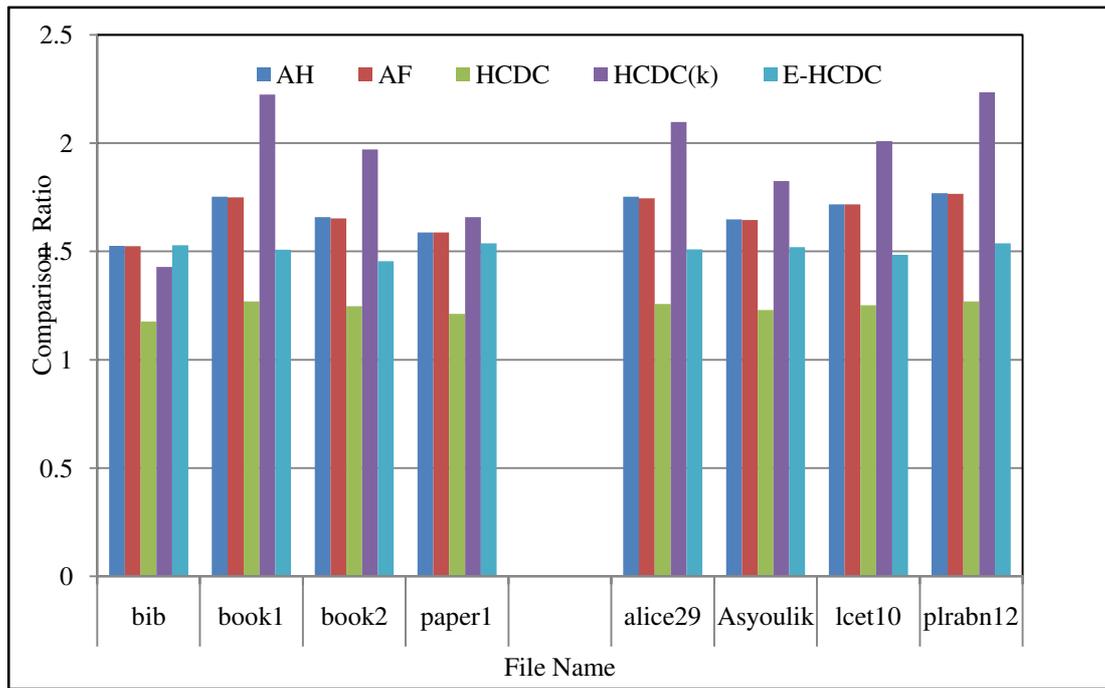


Figure (4.4). Comparing  $C_{E-HCDC}$  of the E-HCDC algorithm against the compression ratio of various adaptive and statistical algorithms.

## Chapter Five

### Conclusions and recommendations for future work

#### 5.1 Conclusions

This thesis presents a description of an enhanced version of the Hamming Codes based Data Compression (HCDC) algorithm, which is called the Enhanced HCDC and abbreviated as E-HCDC. The compression power of the algorithm and the actual enhancement achieved by the E-HCDC algorithm over the original HCDC are evaluated through a number of experiments of compressing a number of text files from Standards Corpora, such as the Calgary, Canterbury, Artificial, and Large Corpora. The compression ratios of the new algorithm are also compared against its preceding algorithm (i.e., the HCDC algorithm) and many other statistical and adaptive algorithms for compressing the same text files.

The main conclusions of this thesis are:

- (1) The E- HCDC algorithm provides a compression ratio of 15% to 20% higher than the compression ratio of the original HCDC algorithm for a range of text files from various standard corpora. This is achieved through the utilization of the newly developed  $m$ -encoding scheme for the pre-fix bits.
- (2) The analytical procedure for estimating the compression ratio provides a mean to automatically compute the optimum length for the pre-fix bits, which minimize the user intervention with the code.
- (3) The E-HCDC algorithm achieves a compression ratio that is high enough to be competent with the compression ratios achieved by many well-known algorithms of statistical and adaptive nature.
- (4) The E-HCDC algorithm achieves acknowledgeable compression ratios encouraging for extending this work to the development of an enhanced version of the HCDC( $k$ ) algorithm, which will be expected to be the highest possible compression ratio for a lossless bil-level data compression algorithm.

- (5) The new algorithm as its preceding algorithm is characterized as:
  - a. Lossless
  - b. bit-level
  - c. An asymmetric
- (6) Finally, the E-HCDC algorithm can be used as a post processing technique to increase the compression ratio of statistical lossless data compression algorithms, such as Shanon-Fano coding, Huffman coding, arithmetic coding, a combination of these algorithms, or any modified form of them.

## 5.2 Recommendations for Future Work

The main recommendations for future work are:

- (1) Utilize the m-encoding to develop a repetitive version of the E-HCDC algorithm, namely, a new E-HCDC( $k$ ) algorithm.
- (2) Evaluate the performance of the E-HCDC algorithm in compressing other types of files, such as standstill images, video files, etc.
- (3) Use the E-HCDC algorithm as a post-compression stage to other data compression algorithms, in particular, Shanon-Fano coding, Huffman coding, arithmetic coding, a combination of these algorithms, or any modified form of them.
- (4) 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.
- (5) Modify the core of E-HCDC algorithm itself in some way to provide higher compression ratio.

## References

- Adiego, J., & De la Fuente, P. (2006). On the use of words as source alphabet symbols in PPM. In *Data Compression Conference, 2006. DCC 2006. Proceedings* (p. 1 pp. –435). Presented at the Data Compression Conference, 2006. DCC 2006. Proceedings. doi:10.1109/DCC.2006.60
- Adiego, Joaquín, Navarro, G., & De la Fuente, P. (2007). Using structural contexts to compress semistructured text collections. *Information Processing & Management*, 43(3), 769–790. doi:10.1016/j.ipm.2006.07.001
- 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. doi:10.1016/j.camwa.2007.11.043
- Al-Bahadili, H., & Al-Saab, S. (2011). A Novel Compressed Index-Query Web Search Engine Model Utilizing the HCDC Algorithm. *The Research Bulletin of Jordan ACM*, 2(4), 119 – 128.
- Al-Bahadili, H., & Hussain, S. M. (2008). An adaptive character wordlength algorithm for data compression. *Computers & Mathematics with Applications*, 55(6), 1250–1256. doi:10.1016/j.camwa.2007.05.014
- Al-Bahadili, H., & Rababa'a, A. (2010). A Bit-Level Text Compression Based on the HCDC Algorithm. *International Journal of Computers and Applications*, 32(3). doi:10.2316/Journal.202.2010.3.202-2914
- Al-Saab, S. (2011). *A Novel Search Engine Model Based on Index-Query Bit-Level Compression* (PhD Thesis). University of Banking & Financial Sciences, Faculty of Information Technology and Systems, Amman, Jordan.
- Amro, I., Zitar, R. A., & Bahadili, H. (2011). Speech compression exploiting linear prediction coefficients codebook and hamming correction code algorithm. *International Journal of Speech Technology*, 14(2), 65–76. doi:10.1007/s10772-011-9091-7

- Arnold, R., & Bell, T. (1997). A corpus for the evaluation of lossless compression algorithms. In *Data Compression Conference, 1997. DCC '97. Proceedings* (pp. 201–210). Presented at the Data Compression Conference, 1997. DCC '97. Proceedings. doi:10.1109/DCC.1997.582019
- Barr, K. C., & Asanović, K. (2006). Energy-aware lossless data compression. *ACM Trans. Comput. Syst.*, 24(3), 250–291. doi:10.1145/1151690.1151692
- Bell, T. C., & Witten, I. H. (1990). Test Compression. In *Test Compression*.
- Brisaboa, N. R., Fariña, A., Navarro, G., & Paramá, J. R. (2005). Efficiently decodable and searchable natural language adaptive compression. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 234–241). New York, NY, USA: ACM. doi:10.1145/1076034.1076076
- 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. doi:10.1016/j.jvcir.2006.09.001
- Chu, A. (2002). LZAC lossless data compression. In *Data Compression Conference, 2002. Proceedings. DCC 2002* (p. 449). Presented at the Data Compression Conference, 2002. Proceedings. DCC 2002. doi:10.1109/DCC.2002.999992
- Conley, E. S., & Klein, S. T. (2006). Compression of multilingual aligned texts. In *Data Compression Conference, 2006. DCC 2006. Proceedings* (p. 1 pp. – 442). Presented at the Data Compression Conference, 2006. DCC 2006. Proceedings. doi:10.1109/DCC.2006.15
- Dai, V. (2008). *Data Compression for Maskless Lithography Systems: Architecture, Algorithms and Implementation*. ProQuest.
- 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. doi:10.1016/j.ipl.2004.02.011

- G. Caire, S. S., & S. Verdu. (2004). *Advances In Network Information Theory: Dimacs Workshop Network Information Theory, March 17-19, 2003, Piscataway, New Jersey*. American Mathematical Soc.
- Gilbert, J., & Abrahamson, D. M. (2006). Adaptive object code compression. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (pp. 282–292). New York, NY, USA: ACM. doi:10.1145/1176760.1176795
- Hashemian, R. (2003). Direct Huffman coding and decoding using the table of code-lengths. In *International Conference on Information Technology: Coding and Computing [Computers and Communications], 2003. Proceedings. ITCC 2003* (pp. 237 – 241). Presented at the International Conference on Information Technology: Coding and Computing [Computers and Communications], 2003. Proceedings. ITCC 2003. doi:10.1109/ITCC.2003.1197533
- Howard, P. G., & Vitter, J. S. (1994). Arithmetic coding for data compression. *Proceedings of the IEEE*, 82(6), 857–865. doi:10.1109/5.286189
- Huffman, D. A. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9), 1098 –1101. doi:10.1109/JRPROC.1952.273898
- Irshid, M. I. (2001). A Simple Binary Run-Length Compression Technique for Non-Binary Sources Based on Source Mapping. *Active and Passive Electronic Components*, 24(4), 211–221. doi:10.1155/2001/23505
- Isal, R. Y. K., & Moffat, A. (2001). Parsing strategies for BWT compression. In *Data Compression Conference, 2001. Proceedings. DCC 2001*. (pp. 429 –438). Presented at the Data Compression Conference, 2001. Proceedings. DCC 2001. doi:10.1109/DCC.2001.917174
- Jaradat, A.-R., Irshid, M., & Nassar, T. (2006). A File Splitting Technique for Reducing the entropy of text files. *Int. Journal of Information Technology*, 3(2).

- Karpinski, M., & Nekrich, Y. (2009). A Fast Algorithm for Adaptive Prefix Coding. *Algorithmica*, 55(1), 29–41. doi:10.1007/s00453-007-9140-4
- Kimura, N., & Latifi, S. (2005). A survey on data compression in wireless sensor networks. In *International Conference on Information Technology: Coding and Computing, 2005. ITCC 2005* (Vol. 2, pp. 8 – 13 Vol. 2). Presented at the International Conference on Information Technology: Coding and Computing, 2005. ITCC 2005. doi:10.1109/ITCC.2005.43
- Klein, S. T. (2000). Skeleton Trees for the Efficient Decoding of Huffman Encoded Texts. *Information Retrieval*, 3(1), 7–23. doi:10.1023/A:1009910017828
- Knuth, D. E. (1985). Dynamic huffman coding. *Journal of Algorithms*, 6(2), 163–180. doi:10.1016/0196-6774(85)90036-7
- Kui Liu, Y., & Žalik, B. (2005). An efficient chain code with Huffman coding. *Pattern Recognition*, 38(4), 553–557. doi:10.1016/j.patcog.2004.08.017
- Lansky, J., & Zemlicka, M. (2006). Compression of small text files using syllables. In *Data Compression Conference, 2006. DCC 2006. Proceedings* (p. 1 pp. – 458). Presented at the Data Compression Conference, 2006. DCC 2006. Proceedings. doi:10.1109/DCC.2006.16
- Lánský, J., & Zemlicka, M. (2005). Text Compression: Syllables. In *DATESO* (Vol. 129, pp. 32–45). Retrieved from <http://www.cs.vsb.cz/dateso/sbornik/dateso05.pdf#page=40>
- Mahoney, M. (2000). Fast Text Compression with Neural Networks. *Proceedings of the 13th International Florida Artificial Intelligence Research Society Conference*, 230 – 234.
- Moffat, A., & Isal, R. Y. K. (2005). Word-based text compression using the Burrows–Wheeler transform. *Information Processing & Management*, 41(5), 1175–1192. doi:10.1016/j.ipm.2004.08.009
- Nelson, M. R. (1989). LZW data compression. *Dr. Dobb's J.*, 14(10), 29–36.

- Nofal, S. (2007). Bit-Level Text Compression. *Proceedings of the First International Conference on Digital Communications and Computer Applications*, 486 – 488.
- Pandya, M. K. (2000). *Data Compression: Efficiency of Varied Compression Techniques*. UK: University of Brunel.
- Plantinga, H. (2006). An Asymmetric, Semi-adaptive Text Compression Algorithm. *IEEE Data Compression*.
- Rein, S., Gühmann, C., & Fitzek, F. (2006). Compression of Short Text on Embedded Systems. *Journal of Computers*, 1(6). doi:10.4304/jcp.1.6.1-10
- Rico-Juan, J. R., Calera-Rubio, J., & Carrasco, R. C. (2005). Smoothing and compression with stochastic -testable tree languages. *Pattern Recognition*, 38(9), 1420–1430. doi:10.1016/j.patcog.2004.03.024
- Robert, L., & Nadarajan, R. (2006). New Algorithms For Random Access Text Compression. In *Third International Conference on Information Technology: New Generations, 2006. ITNG 2006* (pp. 104 –111). Presented at the Third International Conference on Information Technology: New Generations, 2006. ITNG 2006. doi:10.1109/ITNG.2006.98
- Rueda, L. G., & Oommen, B. J. (2001). Enhanced static Fano coding. In *2001 IEEE International Conference on Systems, Man, and Cybernetics* (Vol. 4, pp. 2163 –2169 vol.4). Presented at the 2001 IEEE International Conference on Systems, Man, and Cybernetics. doi:10.1109/ICSMC.2001.972876
- Rueda, L., & Oommen, B. J. (2006). *A Fast and Efficient Nearly-Optimal Adaptive Fano Coding Scheme*.
- Salomon, D. (2004). *Data Compression: The Complete Reference*. Springer.
- Sayood, K. (2012). *Introduction to data compression. Fourth Edition, Morgan Kaufmann*.
- Shannon, C. E. (1951). Prediction and Entropy of Printed English”, *The Bell System Technical Journal*.

- Sharieh, A. (2004). An Enhancement of Huffman Coding for the Compression of Multimedia File. *Transactions of Engineering Computing and Technology*, 3(1), 303 – 305.
- Sharma, A. B., Golubchik, L., Govindan, R., & Neely, M. J. (2009). Dynamic data compression in multi-hop wireless networks. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems* (pp. 145–156). New York, NY, USA: ACM. doi:10.1145/1555349.1555367
- Sirbu, A., & Cleju, I. (2011). On Some Characteristics of a Novel Lossless Data Compression Algorithm based on Polynomial Codes. Retrieved from [http://iit.iit.tuiasi.ro/Reviste/mem\\_sc\\_st\\_2011/14\\_MSS\\_Sirbu\\_Cleju.pdf](http://iit.iit.tuiasi.ro/Reviste/mem_sc_st_2011/14_MSS_Sirbu_Cleju.pdf)
- Tanenbaum, A. (2003). *Computer Networks*. Prentice Hall.
- Vitter, J. S. (1989). Dynamic Huffman Coding. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.68.7162>
- Wang, W. Y. (2009). A lossless compression method for JPEG based on shuffle algorithm. In *International Conference on Image Analysis and Signal Processing, 2009. IASP 2009* (pp. 131 –132). Presented at the International Conference on Image Analysis and Signal Processing, 2009. IASP 2009. doi:10.1109/IASP.2009.5054622
- Wang, Z. H., Chang, C. C., Chen, K. N., & Li, M. C. (2009). A Modification of VQ Index Table for Data Embedding and Lossless Indices Recovery. *電腦學刊*, 20(4), 42–52.
- Wijngaarden, A. J. V. (2004). *Advances In Network Information Theory: Dimacs Workshop Network Information Theory, March 17-19, 2003, Piscataway, New Jersey*. American Mathematical Soc.
- Witten, I. H. (2004). Adaptive text mining: inferring structure from sequences. *Journal of Discrete Algorithms*, 2(2), 137–159. doi:10.1016/S1570-8667(03)00084-4

- Witten, I. H., Bell, T. C., Emberson, H., Inglis, S., & Moffat, A. (1994). Textual image compression: two-stage lossy/lossless encoding of textual images. *Proceedings of the IEEE*, 82(6), 878–888. doi:10.1109/5.286192
- Witten, I. H., Neal, R. M., & Cleary, J. G. (1987). Arithmetic coding for data compression. *Commun. ACM*, 30(6), 520–540. doi:10.1145/214762.214771
- Wong, K.-W., Lin, Q., & Chen, J. (2011). Error detection in arithmetic coding with artificial markers. *Computers & Mathematics with Applications*, 62(1), 359–366. doi:10.1016/j.camwa.2011.05.017
- Xie, Y., Wolf, W., & Lekatsas, H. (2003). Code compression using variable-to-fixed coding based on arithmetic coding. In *Data Compression Conference, 2003. Proceedings. DCC 2003* (pp. 382–391). Presented at the Data Compression Conference, 2003. Proceedings. DCC 2003. doi:10.1109/DCC.2003.1194029
- Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3), 337–343. doi:10.1109/TIT.1977.1055714
- Ziv, J., & Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5), 530–536. doi:10.1109/TIT.1978.1055934

**Appendix A**  
**Compression Corpora**

A.1	Calgary Corpus
A.2	Canterbury Corpus
A.3	Artificial Corpus
A.4	Large Corpus
A.5	Miscellaneous Corpus

## Compression Corpora

In order to evaluate the performance of various compression schemes, standard corpora are usually used, these include:

- (1) Calgary Corpus
- (2) Canterbury Corpus
- (3) Artificial Corpus
- (4) Large Corpus
- (5) Miscellaneous Corpus

This appendix provides a description of the above corpora and their constituent files.

### A.1 Calgary Corpus

The Calgary Corpus is the most referenced corpus in the data compression field, especially, for text compression and is the de facto standard for lossless compression evaluation. The corpus was founded in 1987 by Ian Witten, Timothy Bell and John Cleary (Bell & Witten, 1990; Ian H. Witten et al., 1987) There are two versions of this corpus:

- (1) Large Calgary corpus which consists of 18 files (Bib, Book1, Book2, Geo, News, Obj1, Obj2, Paper1, Paper2, Paper3, Paper4, Paper5, Paper6, Pic, Progc, Progl, Progp and Trans).
- (2) Standard Calgary Corpus which consists of 14 files (all files above except Paper3, Paper4, Paper5 and Paper6).

Nine different types of text are represented, and to confirm that the performance of schemes is consistent for any given type, many of the types have more than one representative. Normal English, both fiction and non-fiction, is represented by two books and six papers (labeled Book1, Book2, Paper1, Paper2, Paper3, Paper4, Paper5, Paper6). More unusual styles of English writing are found in a bibliography (Bib) and a batch of unedited news articles (News).

Three computer programs represent artificial languages (Progc, Progl, Progp). A transcript of a terminal session (Trans) is included to indicate the increase in speed that could be achieved by applying compression to a slow line to a terminal. All of the files mentioned so far use ASCII encoding. Some non-ASCII files are also included: two files of executable code (Obj1, Obj2), some geophysical data (Geo), and a bit-map black and white picture (Pic).

The file Geo is particularly difficult to compress because it contains a wide range of data values, while the file Pic is highly compressible because of large amounts of white space in the picture, represented by long runs of zeros. More details of the individual texts are given in (Bell & Witten, 1990). In addition, results of compression experiments on these texts are given in (Ian H. Witten et al., 1987)

Table (A.1) - Calgary Corpus.			
#	File Name	Size (Byte)	Contents
1	Bib	111261	Structured text (bibliography)
2	Book1	768771	Text
3	Book2	610856	Formatted text, scientific
4	Geo	102400	Geophysical data
5	News	377109	Formatted text, script with news
6	Obj1	21504	Program code (object file), executable machine code
7	Obj2	246814	Program code (object file), executable machine code
8	Paper1	53161	Formatted text, scientific
9	Paper2	82199	Formatted text, scientific
10	Paper3	46526	Formatted text, scientific
11	Paper4	13286	Formatted text, scientific
12	Paper5	11954	Formatted text, scientific
13	Paper6	38105	Formatted text, scientific
14	Pic	513216	Image data (black and white)
15	Progc	39611	Source code
16	Progl	71646	Source code
17	Progp	49379	Source code
18	Trans	93695	Transcript terminal data

## A.2 Canterbury Corpus

The Canterbury collection is the main benchmark for comparing compression methods. It was developed in 1997 by Ross Arnold and Tim Bell (Arnold & Bell, 1997) as an improved version of the Calgary Corpus. It consists of 11 files. The files were chosen because their results on existing compression algorithms are “typical”, and so it is hoped this will also be true for new methods. Ross Arnold and Tim Bell (Arnold & Bell, 1997) explain how the files were chosen, and why it is difficult to find “typical” files. This collection will not be changed so that it can be used as a benchmark in future.

#	File Name	Size (Byte)	Contents
1	alice29.txt	152089	English text
2	asyoulik.txt	125179	Shakespeare
3	cp.html	24603	HTML source
4	fields.c	11150	C source
5	grammar.lsp	3721	LISP source
6	kennedy.xls	1029744	Excel Spreadsheet
7	lcet10.txt	426754	Technical writing
8	plravn12.txt	481861	Poetry
9	ptt5	513216	CCITT test set
10	sum	38240	SPARC Executable
11	xargs.1	4227	GNU manual page

## A.3 Artificial Corpus

The Artificial Corpus is a collection that contains 4 files for which the compression methods may exhibit pathological or worst-case behavior - files containing little or no repetition (`random.txt`), files containing large amounts of repetition (`alphabet.txt`), or very small files (`a.txt`).

As such, “average” results for this collection will have little or no relevance, as the data files have been designed to detect outliers. Similarly, times for “trivial” files will be negligible, and should not be reported.

New files can be added to this collection, so the overall average for the collection should not be reported as a benchmark. Results on this corpus should be reported for individual files, or a subset should be identified. Existing files in the collection will not be changed or removed.

Table (A.3) - Artificial Corpus.			
#	File Name	Size (Byte)	Contents
1	a.txt	1	The letter "a".
2	aaa.txt	100000	The letter "a", repeated 100,000 times.
3	alphabet.txt	100000	Enough repetitions of the alphabet to fill 100,000 characters
4	random.txt	100000	100,000 characters, randomly selected from [a-z, A-Z, 0-9] (alphabet size 64)

#### A.4 Large Corpus

The Large Corpus is a collection of relatively 3 large files. While most compression methods can be evaluated satisfactorily on smaller files, some require very large amounts of data to get good compression, and some are so fast that the larger size makes speed measurement more reliable. New files can be added to this collection, so the overall average for the collection should not be reported as a benchmark. Results on this corpus should be reported for individual files, or a subset should be identified. Existing files in the collection will not be changed or removed.

Table (A.4) - Large Corpus.			
#	File Name	Size (Byte)	Contents
1	E.coli	4638690	Complete genome of the E. Coli bacterium.
2	bible.txt	4047392	The King James version of the bible (bible).
3	world192.txt	2473400	The CIA world fact book (world).

#### A.5 Miscellaneous Corpus

This is a collection of "miscellaneous" files that is designed to be added to by researchers and others wishing to publish compression results using their own files. New files can be added to this collection, so the overall average for the collection should not be reported as a benchmark.

Results on this corpus should be reported for individual files, or a subset should be identified. Existing files in the collection will not be changed or removed. There is only one file in this corpus till now.

Table (A.5) - The Miscellaneous Corpus.			
#	File Name	Size (Byte)	Contents
1	pi.txt	10000000	The first million digits of pi.

## Appendix B

### List of Characters and Characters Frequencies of Some Text File from Standard Corpora

Table	Filename	No. of Characters ( $N_c$ )
B.1	alice29.txt	73
B.2	bib	81
B.3	bible.txt	63
B.4	boo1	80
B.5	paper1	95

Table (B.1) – Filename: alice29.txt

alice29.txt						
Size (Byte)	$m$	$S_o$	$N_c$	$S_c$	$C$	
152089	5	1216712	73	795702	1.53	
$i$	Character	Occurrence	Character Frequency $f_i$ (%)	Group Frequency	Binary Representation	
					ASCII	E-HCDC
1	Space	28900	19.0	81.11	00100000	00000
2	e	13381	8.8		01100101	00001
3	t	10212	6.71		01110100	00010
4	a	8149	5.36		01100001	00011
5	o	7965	5.24		01101111	00100
6	h	7088	4.66		01101000	00101
7	n	6893	4.53		01101110	00110
8	i	6778	4.46		01101001	00111
9	s	6277	4.13		01110011	01000
10	r	5293	3.48		01110010	01001
11	d	4739	3.12		01100100	01010
12	l	4615	3.03		01101100	01011
13	LF	3608	2.37		00001010	01100
14	CR	3608	2.37		00001101	01101
15	u	3402	2.24		01110101	01110
16	g	2446	1.61		01100111	01111
17	w	2437	1.6	15.58	01110111	100000
18	,	2418	1.59		00101100	100001
19	c	2253	1.48		01100011	100010
20	y	2150	1.41		01111001	100011
21	f	1926	1.27		01100110	100100
22	m	1907	1.25		01101101	100101
23	'	1761	1.16		00100111	100110
24	p	1458	0.96		01110000	100111
25	b	1383	0.91		01100010	101000
26	`	1108	0.73		01100000	101001
27	k	1076	0.71		01101011	101010
28	.	977	0.64		00101110	101011
29	v	803	0.53		01110110	101100
30	I	733	0.48		01001001	101101
31	-	669	0.44		00101101	101110
32	A	638	0.42		01000001	101111
33	T	472	0.31	2.37	01010100	1100000
34	!	449	0.3		00100001	1100001
35	H	284	0.19		01001000	1100010
36	W	237	0.16		01010111	1100011
37	:	233	0.15		00111010	1100100
38	S	218	0.14		01010011	1100101
39	?	202	0.13		00111111	1100110

40	M	200	0.13		01001101	1100111
41	;	194	0.13		00111011	1101000
42	D	192	0.13		01000100	1101001
43	E	188	0.12		01000101	1101010
44	O	176	0.12		01001111	1101011
45	C	144	0.09		01000011	1101100
46	x	144	0.09		01111000	1101101
47	R	140	0.09		01010010	1101110
48	j	138	0.09		01101010	1101111
49	q	125	0.08		01110001	11100000
50	N	120	0.08		01001110	11100001
51	Y	114	0.07		01011001	11100010
52	"	113	0.07		00100010	11100011
53	L	98	0.06		01001100	11100100
54	B	91	0.06		01000001	11100101
55	Q	84	0.06		01010001	11100110
56	G	82	0.05	0.89	01000111	11100111
57	K	82	0.05		01001011	11101000
58	z	77	0.05		01111010	11101001
59	F	74	0.05		01000110	11101010
60	U	66	0.04		01010101	11101011
61	P	64	0.04		01010000	11101100
62	*	60	0.04		00101010	11101101
63	(	56	0.04		00101000	11101110
64	)	55	0.04		00101001	11101111
65	V	42	0.03		01010110	111100000
66	J	8	0.01		01001010	111100001
67	X	4	0		01011000	111100010
68	-	4	0		01011111	111100011
69	[	2	0	0.04	01011011	111100100
70	]	2	0		01011101	111100101
71	2	1	0		00110010	111100110
72	9	1	0		00111001	111100111
73	Z	1	0		01011010	111101000
Sum		152088	100%	100%		

**Table (B.2) – Filename: bib.txt**

bib.txt						
Size (Byte)	$m$	$S_o$	$N_c$	$S_c$	$C$	
111261	6	890088	81	611819	1.45	
$i$	Character	Occurrence	Character Frequency $f_i(\%)$	Group Frequency	Binary Representation	
					ASCII	E-HCDC
1	Space	13739	12.35	68.42	00100000	00000
2	e	6984	6.28		01100101	00001
3	LF	6280	5.64		00001010	00010
4	%	5556	4.99		00100101	00011
5	n	5462	4.91		01101110	00100
6	i	4887	4.39		01101001	00101
7	o	4756	4.27		01101111	00110
8	a	4737	4.26		01100001	00111
9	r	4664	4.19		01110010	01000
10	t	4502	4.05		01110100	01001
11	s	3451	3.1		01110011	01010
12	c	2665	2.4		01100011	01011
13	l	2442	2.19		01101100	01100
14	A	2035	1.83		01000001	01101
15	m	1994	1.79		01101101	01110
16	.	1984	1.78		00101110	01111
17	u	1617	1.45	18.24	01110101	100000
18	,	1607	1.44		00101100	100001
19	g	1569	1.41		01100111	100010
20	d	1540	1.38		01100100	100011
21	p	1489	1.34		01110000	100100
22	l	1486	1.34		00110001	100101
23	h	1364	1.23		01101000	100110
24	f	1235	1.11		01100110	100111
25	T	1219	1.1		01010100	101000
26	C	1165	1.05		01000011	101001
27	D	1089	0.98		01000100	101010
28	y	1057	0.95		01111001	101011
29	P	1039	0.93		01010000	101100
30	9	973	0.87		00111001	101101
31	J	930	0.84		01001010	101110
32	I	911	0.82		01001001	101111
33	8	864	0.78	8.85	00111000	1100000
34	-	842	0.76		00101101	1100001
35	S	814	0.73		01010011	1100010
36	M	696	0.63		01001101	1100011
37	E	682	0.61		01000101	1100100
38	b	671	0.6		01100010	1100101
39	2	637	0.57		00110010	1100110

40	R	613	0.55		01010010	1100111
41	3	581	0.52		00110011	1101000
42	K	579	0.52		01001011	1101001
43	O	540	0.49		01001111	1101010
44	N	523	0.47		01001110	1101011
45	v	507	0.46		01110110	1101100
46	4	457	0.41		00110100	1101101
47	5	425	0.38		00110101	1101110
48	w	416	0.37		01110111	1101111
49	*	402	0.36	3.97	00101010	11100000
50	7	396	0.36		00110111	11100001
51	V	381	0.34		01010110	11100010
52	B	370	0.33		01000001	11100011
53	6	343	0.31		00110110	11100100
54	k	318	0.29		01101011	11100101
55	L	309	0.28		01001100	11100110
56	H	304	0.27		01001000	11100111
57	0	287	0.26		00110000	11101000
58	G	278	0.25		01000111	11101001
59	W	220	0.2		01010111	11101010
60	x	194	0.17		01111000	11101011
61	F	177	0.16		01000110	11101100
62	U	168	0.15		01010101	11101101
63	:	135	0.12		00111010	11101110
64	z	133	0.12		01111010	11101111
65	Y	112	0.10	0.52	01011001	111100000
66	q	80	0.07		01110001	111100001
67	\	74	0.07		01011100	111100010
68	(	44	0.04		00101000	111100011
69	X	42	0.04		01011000	111100100
70	Z	42	0.04		01011010	111100101
71	'	35	0.03		00100111	111100110
72	)	33	0.03		00101001	111100111
73	/	32	0.03		00101111	111101000
74	j	22	0.02		01101010	111101001
75	l	21	0.02		01111100	111101010
76	?	10	0.01		00111111	111101011
77	`	10	0.01		01100000	111101100
78	Q	8	0.01		01010001	111101101
79	;	3	0		00111011	111101110
80	+	2	0		00101011	111101111
81	!	1	0	0.00	00100001	1111100000
Sum		111261	100%	100%		

Table (B.3) – Filename: bible.txt

bible.txt						
Size (Byte)	$m$	$S_o$	$N_c$	$S_c$	$C$	
4047392	4	32379136	63	20875652	1.55	
$i$	Character	Occurrence	Character Frequency $f_i$ (%)	Group Frequency	Binary Representation	
					ASCII	E-HCDC
1	Space	766111	18.93	86.38	00100000	00000
2	e	396042	9.79		01100101	00001
3	t	299633	7.40		01110100	00010
4	h	270179	6.68		01101000	00011
5	a	248716	6.15		01100001	00100
6	o	226152	5.59		01101111	00101
7	n	215496	5.32		01101110	00110
8	s	179075	4.42		01110011	00111
9	i	174140	4.30		01101001	01000
10	r	157355	3.89		01110010	01001
11	d	144021	3.56		01100100	01010
12	l	117300	2.90		01101100	01011
13	u	80762	2.00		01110101	01100
14	f	78370	1.94		01100110	01101
15	m	74364	1.84		01101101	01110
16	,	68389	1.69		00101100	01111
17	w	61051	1.51	11.72	01110111	100000
18	y	56323	1.39		01111001	100001
19	c	51317	1.27		01100011	100010
20	g	47279	1.17		01100111	100011
21	b	42888	1.06		01100010	100100
22	p	39885	0.99		01110000	100101
23	LF	30383	0.75		00001010	100110
24	v	29448	0.73		01110110	100111
25	.	25438	0.63		00101110	101000
26	k	20703	0.51		01101011	101001
27	A	17038	0.42		01000001	101010
28	I	12823	0.32		01001001	101011
29	:	12439	0.31		00111010	101100
30	;	9968	0.25		00111011	101101
31	L	8859	0.22		01001100	101110
32	O	8547	0.21		01001111	101111
33	D	8425	0.21	1.64	01000100	1100000
34	T	7424	0.18		01010100	1100001
35	R	7179	0.18		01010010	1100010
36	G	5943	0.15		01000111	1100011
37	J	5920	0.15		01001010	1100100
38	S	4618	0.11		01010011	1100101
39	B	4472	0.11		01000001	1100110

40	?	3179	0.08		00111111	1100111
41	H	3042	0.08		01001000	1101000
42	M	2954	0.07		01001101	1101001
43	E	2439	0.06		01000101	1101010
44	j	2388	0.06		01101010	1101011
45	W	2345	0.06		01010111	1101100
46	F	2292	0.06		01000110	1101101
47	'	1943	0.05		00100111	1101110
48	z	1828	0.05		01111010	1101111
49	N	1746	0.04		01001110	11100000
50	P	1718	0.04		01010000	11100001
51	C	1621	0.04		01000011	11100010
52	x	1423	0.04		01111000	11100011
53	q	930	0.02		01110001	11100100
54	Z	883	0.02		01011010	11100101
55	Y	529	0.01		01011001	11100110
56	K	519	0.01		01001011	11100111
57	!	308	0.01		00100001	11101000
58	U	275	0.01		01010101	11101001
59	(	214	0.01		00101000	11101010
60	)	214	0.01		00101001	11101011
61	V	99	0		01010110	11101100
62	-	23	0		00101101	11101101
63	Q	5	0		01010001	11101110
Sum		4047392	100%	0.26		100%

Table (B.4) – Filename: book1.txt

book1.txt						
Size (Byte)	$m$	$S_o$	$N_c$	$S_c$	$C$	
768771	6	6150168	80	4000973	1.54	
$i$	Character	Occurrence	Character Frequency $f_i(\%)$	Group Frequency	Binary Representation	
					ASCII	E-HCDC
1	Space	125552	16.33	82.77	00100000	00000
2	e	72431	9.42		01100101	00001
3	t	50027	6.51		01110100	00010
4	a	47836	6.22		01100001	00011
5	o	44795	5.83		01101111	00100
6	n	40919	5.32		01101110	00101
7	h	37561	4.89		01101000	00110
8	i	37007	4.81		01101001	00111
9	s	36788	4.79		01110011	01000
10	r	32889	4.28		01110010	01001
11	d	26623	3.46		01100100	01010
12	l	23078	3.00		01101100	01011
13	LF	16622	2.16		00001010	01100
14	u	16031	2.09		01110101	01101
15	w	14071	1.83		01110111	01110
16	m	14044	1.83		01101101	01111
17	c	12685	1.65	14.92	01100011	100000
18	g	12303	1.6		01100111	100001
19	f	12237	1.59		01100110	100010
20	y	11986	1.56		01111001	100011
21	,	10296	1.34		00101100	100100
22	p	9332	1.21		01110000	100101
23	b	9132	1.19		01100010	100110
24	.	7170	0.93		00101110	100111
25	'	6470	0.84		00100111	101000
26	v	5382	0.7		01110110	101001
27	k	4994	0.65		01101011	101010
28	-	3955	0.51		00101101	101011
29	I	2899	0.38		01001001	101100
30	"	2468	0.32		00100010	101101
31	T	1966	0.26		01010100	101110
32	B	1463	0.19		01000001	101111
33	H	977	0.13	1.53	01001000	1100000
34	A	967	0.13		01000001	1100001
35	x	861	0.11		01111000	1100010
36	O	856	0.11		01001111	1100011
37	S	850	0.11		01010011	1100100
38	!	832	0.11		00100001	1100101
39	;	762	0.1		00111011	1100110

40	?	759	0.1		00111111	1100111
41	W	753	0.1		01010111	1101000
42	P	693	0.09		01010000	1101001
43	+	691	0.09		00101011	1101010
44	C	580	0.08		01000011	1101011
45	G	575	0.07		01000111	1101100
46	M	565	0.07		01001101	1101101
47	q	520	0.07		01110001	1101110
48	N	502	0.07		01001110	1101111
49	<	498	0.06	0.67	00111100	11100000
50	>	498	0.06		00111110	11100001
51	j	468	0.06		01101010	11100010
52	E	444	0.06		01000101	11100011
53	Y	416	0.05		01011001	11100100
54	F	413	0.05		01000110	11100101
55	L	413	0.05		01001100	11100110
56	D	269	0.03		01000100	11100111
57	z	264	0.03		01111010	11101000
58	J	253	0.03		01001010	11101001
59	R	245	0.03		01010010	11101010
60	1	240	0.03		00110001	11101011
61	:	220	0.03		00111010	11101100
62	2	185	0.02		00110010	11101101
63	3	184	0.02		00110011	11101110
64	4	151	0.02		00110100	11101111
65	U	103	0.01	0.11	01010101	111100000
66	0	98	0.01		00110000	111100001
67	5	96	0.01		00110101	111100010
68	6	87	0.01		00110110	111100011
69	7	85	0.01		00110111	111100100
70	8	85	0.01		00111000	111100101
71	9	82	0.01		00111001	111100110
72	V	64	0.01		01010110	111100111
73	K	45	0.01		01001011	111101000
74	(	43	0.01		00101000	111101001
75	)	40	0.01		00101001	111101010
76	Q	14	0		01010001	111101011
77	=	5	0		00111101	111101100
78	X	5	0		01011000	111101101
79	&	1	0		00100110	111101110
80	*	1	0		00101010	111101111
Sum			100%	100%		

**Table (B.5) – Filename: paper1**

paper1						
Size (Byte)	$m$	$S_o$	$N_c$	$S_c$	$C$	
53161	6	425288	95	286499	1.48	
$i$	Character	Occurrence	Character Frequency $f_i$ (%)	Group Frequency	Binary Representation	
					ASCII	E-HCDC
1	Space	7301	13.73	74.05	00100000	00000
2	e	4689	8.82		01100101	00001
3	t	3048	5.73		01110100	00010
4	i	2879	5.42		01101001	00011
5	o	2568	4.83		01101111	00100
6	n	2503	4.71		01101110	00101
7	a	2441	4.59		01100001	00110
8	s	2374	4.47		01110011	00111
9	r	2058	3.87		01110010	01000
10	l	1593	3.00		01101100	01001
11	h	1485	2.79		01101000	01010
12	c	1476	2.78		01100011	01011
13	d	1431	2.69		01100100	01100
14	LF	1250	2.35		00001010	01101
15	m	1154	2.17		01101101	01110
16	f	1118	2.10		01100110	01111
17	u	1069	2.01	17.76	01110101	100000
18	p	961	1.81		01110000	100001
19	\	891	1.68		01011100	100010
20	.	839	1.58		00101110	100011
21	g	778	1.46		01100111	100100
22	b	715	1.34		01100010	100101
23	0	666	1.25		00110000	100110
24	w	540	1.02		01110111	100111
25	y	503	0.95		01111001	101000
26	\$	480	0.9		00100100	101001
27	,	431	0.81		00101100	101010
28	l	392	0.74		00110001	101011
29	v	353	0.66		01110110	101100
30	TAB	301	0.57		00001001	101101
31	(	277	0.52		00101000	101110
32	)	243	0.46		00101001	101111
33	x	227	0.43	4.79	01111000	1100000
34	'	226	0.43		00100111	1100001
35	~	224	0.42		01111110	1100010
36	2	214	0.40		00110010	1100011
37	-	195	0.37		00101101	1100100
38	I	179	0.34		01001001	1100101

39	T	157	0.30		01010100	1100110	
40	3	143	0.27		00110011	1100111	
41	9	137	0.26		00111001	1101000	
42	q	137	0.26		01110001	1101001	
43	R	134	0.25		01010010	1101010	
44	5	128	0.24		00110101	1101011	
45	4	122	0.23		00110100	1101100	
46	[	121	0.23		01011011	1101101	
47	_	102	0.19		01011111	1101110	
48	F	101	0.19		01000110	1101111	
49	+	100	0.19		2.24	00101011	11100000
50	8	96	0.18			00111000	11100001
51	k	96	0.18			01101011	11100010
52	6	95	0.18			00110110	11100011
53	A	91	0.17	01000001		11100100	
54	]	84	0.16	01011101		11100101	
55	C	80	0.15	01000011		11100110	
56	7	75	0.14	00110111		11100111	
57	H	74	0.14	01001000		11101000	
58	=	69	0.13	00111101		11101001	
59	"	66	0.12	00100010		11101010	
60	L	60	0.11	01001100		11101011	
61	E	56	0.11	01000101		11101100	
62	N	55	0.10	01001110		11101101	
63	B	49	0.09	01000001	11101110		
64	;	44	0.08	00111011	11101111		
65	/	43	0.08	1.16	00101111	111100000	
66	<	42	0.08		00111100	111100001	
67	S	42	0.08		01010011	111100010	
68	M	39	0.07		01001101	111100011	
69	l	38	0.07		01111100	111100100	
70	P	34	0.06		01010000	111100101	
71	z	32	0.06		01111010	111100110	
72	{	30	0.06		01111011	111100111	
73	`	28	0.05		01100000	111101000	
74	}	28	0.05		01111101	111101001	
75	X	27	0.05		01011000	111101010	
76	O	24	0.05		01001111	111101011	
77	V	22	0.04		01010110	111101100	
78	W	22	0.04		01010111	111101101	
79	j	22	0.04	01101010	111101110		
80	U	21	0.04	01010101	111101111		
81	*	19	0.04		00101010	1111100000	
82	:	16	0.03		00111010	1111100001	
83	!	15	0.03		00100001	1111100010	
84	%	15	0.03		00100101	1111100011	

85	&	14	0.03		00100110	1111100100	
86	D	14	0.03		01000100	1111100101	
87	G	9	0.02		01000111	1111100110	
88	>	6	0.01		00111110	1111100111	
89	J	4	0.01		01001010	1111101000	
90	K	3	0.01		01001011	1111101001	
91	?	2	0		00111111	1111101010	
92	Z	2	0		01011010	1111101011	
93	^	2	0		01011110	1111101100	
94	Q	1	0		01010001	1111101101	
95	Y	1	0		01011001	1111101110	
Sum		53161	100%		100%		