UNIVERSAL LOSSLESS COMPRESSION TECHNIQUE WITH BUILT IN ENCRYPTION

By

Name:Kattan, Ahmed
Registration Number: 0540714
Login: akatta

A thesis submitted in partial fulfilment of the requirements
for the degree of

Masters in Computer Science

2006

Tutor
Prof.Riccardo Poli

Coordinator
Dr.Ramaswamy Palaniappan

14 September 2006

SURNAME: Kattan
OTHER NAMES: Ahmed
QUALIFICATION SOUGHT: Msc Computer Science
TITLE OF PROJECT: Universal lossless compression technique with built in encryption
SUPERVISOR: Professor Riccardo Poli
DATE (month and year): 14 September 2006

Abstract

The proposed algorithm suggests a lossless data compression with encryption technique that depends on reversing the usage use of k-map. On the compressor side, the input bit-stream is chopped into chunks of 16-bits each, and a minimised expression is found for each of the chunks. The minimised expressions of the input data stream are stored. Later, the Huffman tree algorithm is applied to the stored expressions "treating each term as a single unit instead of treating each character as a single unit". The obtained Huffman code is used to convert the original file into a compressed one. A shuffle is added to the Huffman tree which dramatically changes it, therefore it cannot be decoded without the identically shuffled tree, this shuffle is based on a generated of key. On the decompression side, the Huffman tree is used to retrieve the original file. The proposed algorithm can be used for various file formats such as images, videos and text.

A full cycle for the proposed algorithm is to compress a file, encrypt it, decrypt it, and finally decompress it back identically to the original file.

Keywords: Compression, Decompression, Huffman tree, k-map, Encryption, Decryption.

Acknowledgments

The completion of this dissertation was made possible through the support and kind assistance of many people. To all of these, and many others, who contributed directly or indirectly, I owe a debt of gratitude.

First, I must record my immense gratitude to my supervisor Professor Riccardo Poli who patiently listened to many fragments of data and arguments and was able to make the discussions very stimulating. His guidance and conclusive remarks had a remarkable impact on my dissertation.

Further, I would like to express my deepest gratitude to Dr. Ramaswamy Palaniappan, the second assessor of my dissertation, for his pertinent responses to all my queries and questions.

I am also indebted to all my colleagues who supported me during this painstaking academic year, especially Mohammad Al-Mulla and Abdullah Shata. Their cheerfulness and sense of humour would always brighten a bad day; I would have never made it without their unfailing support.

Last but not least, as always, I owe more than I can say to my exceptionally loving parents whose blessings and nurture pave every step of my way.

Ahmed Jamil Kattan
14 September 2006

Content

List of tables

List of figures

Chapter 1 Introduction

1.1 Preliminaries

> I am now going to begin my story
> (said the old man), so please attend.
>
> — Andrew Lang
>
> The Arabian Nights Entertainments (1898)

One of the paradoxes of the technology evolution is that despite the development of the computer and the increased need for storing information, there is a lack of development of compression techniques. As the evolution of computer systems progresses very quickly, the amount of stored information will increase at the same fast rate.

There are several reasons for reliance on compression, one of the reasons is that the demand for online storage increases at the same speed as storage capacity development, a second reason is the limited bandwidth for some communication channels, such as dialup internet connection, where the maximum bandwidth for this type of connections is 56K/second [1]. The question is why is it important to reduce the size of data? The answer to this is simple, to reduce costs; cost of data storage and the cost of data transmission. This will save memory space which can be used to keep other information, and save time in order to be able to transmit more data in less time.

Data compression aims to condense the data in order to reduce the size of a data file. For example, an ASCII file is compressed into a new file, which contains the same information, but it is smaller in size. The compression of a file into half of its original size increases the free memory that is available for use [2]. The same idea applies to the transmission of messages over the internet with limited bandwidth channels.

A sequence related to stored data often contains some regularities, which motivate the researcher to find a way to avoid these redundant sequences, for example in the English language some of the most common words which are often repeated regularly are: "*the, is, a, of, for, …etc*", and in videos which consists of a sub-sequence of frames, what can happen in two sequenced frames? It is assumed, very little, the same could also be said for pictures which contain regular sequences of pixels [3].

Thus, based on the fact that there are some repetitions in the stored data, we can suggest a definition for the data compression as, the process of observing the regularities in a sequence of data and then trying to reduce it. For example, let S = a sequence of data where $S \in \sum\{0,1\}^+$, F = Compression process, $F^\wedge$ = Decompression process, and $S^\wedge$ = Compressed sequence of data where $S^\wedge \in \sum\{0,1\}^+$. So;

$S^\wedge = F(S)$ can represent the compression process for a sequence of data **(C1)**.

$S = F^\wedge(S^\wedge)$ can represent the decompression process for a compressed a data **(C2)**.

So we can say that $F^\wedge(F(S)) = S$ **(C3)**.

The compression ratio = $|S^\wedge| / |S|$.

One of the factors that helps in developing a strong data compression technique is to have a prior knowledge of the data to be compressed. According to this fact, compression researchers' tends to develop specialised compression techniques which target a specific kind of data, for example there are some compression techniques which are designed to work on text such as the Huffman tree [4],LZW [5], other techniques designed to work on images, and others techniques specialised in multimedia, and so on.

Data compression has a wide range of applications, especially in data storage and data transmission [6][7], some of these applications include: (i) Many archiving systems such as ARC [8], and PKARAC [9], and (ii) Telecommunications such as voice mail and teleconferencing. Currently, communications through networks have resulted in large amounts of data being transferred daily, especially multimedia data, which slows the network due to the large sizes of its files. Therefore, the use of efficient compression techniques will reduce the time of data transmission and the cost of communications [10][11]. Compression has an important role in the spread of Web, since any improvements will decrease the amount of transmitted data over the Internet.

Another area of research is Cryptography, which is known for protecting data. On the encryptor side the plain text is encrypted using an algorithm, to produce another format called cipher text, using an encryption key, while on the decryptor side, the cipher text is decrypted using the same algorithm back to the plain text using a decryption key. Cryptography comes from a Greek word which means “secret writing” [12], thus to ensure privacy the information must be kept hidden.

For example, let P= plaintext, C= cipher text, k =key, E= encryption process, and D= decryption process. So;

$C = E_k(P)$ can represent to mean that encryption plaintext P using key k **(E1)**.

$P = D_k(C)$, to mean that the decryption of C to get plaintext again **(E2)**.

So we can say that $D_k(E_k(P)) = P$ **(E3)**. [39]

One of the fundamental of cryptography is to assume that the cryptanalyst knows the methods used for the encryption and decryption. Thinking of the algorithm as a secret is no harm any more.

Kerckhoff's principle: All algorithms must be public; only the keys are secret.

Named after the Flemish military cryptographer Auauste Kerckhoff who first stated it in 1883[13].

Absolute security is not guaranteed, however there are some characteristics that identify a strong cryptography system. The concealment of the key rather than the encryption technique is one of the most important features of a strong cryptography system [14]. The length of the encryption key is another factor as this will leave huge possibilities for the cryptanalyst to guess the right key [14]. The outcome from a strong cryptosystem should be a random meaningless file, which makes it harder for the cryptanalyst to find some regularities or relationship in the encrypted file [14]. If a cryptography system satisfy these entire characteristic, yet it dose not mean that the encryption system is 100% secure.

With the increasing amount of data stored on computers, the need for security in transmission and the reduction of the storage becomes greater everyday. Compression aids encryption by reducing the file size, "the compression scheme shortens the input file, which shortens the output file and reduces the amount of CPU required to do the encryption algorithm, so even if there were no enhancement of security, compression before encryption would be worthwhile."[15]. However, concerning compression after encryption it is stated; "If an encryption algorithm is good, it will produce output which is statistically indistinguishable from random numbers and no compression algorithm will considerably compress random numbers" [15]. "On the other hand, if a compression algorithm succeeds in finding a pattern to compress out of an encryption's output, then a flaw in that algorithm has been found. In the majority of encryption utilities (e.g., PGP) the data is first compressed before it is actually encrypted" [15].

One of the famous methods to combine those two areas (compression and encryption) was **PGP**; PGP combines some of the best features of both conventional and public key cryptography [17]. PGP is a *hybrid cryptosystem*, when the user encrypts plaintext with PGP. PGP first compresses the plaintext [17]. This will then save the amount of transmitted data over the network, as well as storage space. "PGP, short for (Pretty good privacy) is a public key encryption program originally written by Phil Zimmermann in 1991. Over the past few years, PGP has got thousands of adherent supporters all over the globe and has become a de-facto standard for encryption of email on the internet"[17].

So by combining the previous equations C1, C2, C3, E1, E2, and E3 we can conclude that;

$E_k[\ F(S)\ ] = CS^{\wedge}$ to represent the process of encrypt a compressed data **(EC1).**

$F^{\wedge}\ [D_k(E_k(F(S)))\ ] = S$ **(EC2)**.

1.2 Definitions

This section introduces some terms and concept in both compression and encryption, which will be used in the rest of this thesis.

In general, the compression process consists of two steps: (i) modeling and (ii) coding [18]. And the difference between them is significant as the coding is the process of replacing the input symbols with alternative symbols which should be smaller than the input, however the output from this process is based on the model [18]. Another common term which will often be used is Entropy, which refers to the quantity of encoded information in a message [18]. Regarding cryptography, there are some common terms which are often used among the researchers of that field; starting with **cryptosystems,** which refers to the method of concealment of the data from unauthorised access. Further, **cryptography** is the science of the design of cryptosystems; the cryptosystems takes a message as input, this message is usually called **plain text**, the result of the cryptosystem is a deformed representation of the plain text called **cipher text**, and process of converting the plain text into cipher text is called **encryption**, while the process of converting the cipher text back to plain text is called **decryption**, as the cryptography is the science of encrypting data from unauthorised access, **cryptanalysis** is the science of breaking the encrypted data, each cryptosystem uses a kind of password to protect the ciphered text, this password is usually called the **key**[19]. The estimated period of a specific cryptosystem to keep the data secure is called **algorithm security life time** [20].

1.3 Compression Categories

Data compression studies generally can be divided into two kingdoms: (i) lossy (irreversible) and (ii) lossless (reversible). Lossy technique concedes a certain loss for data in exchange with the high compression ratio. Generally lossy techniques are applied for those kinds of data which accept some loss such as images, videos and audios, as the human senses are imperfect, it will not notice the absence of few pixels in a picture, or a few frames in a video, or even the absence of some background tones in an audio file. On the other hand some kinds of data could not accept any loss (ex. Database records, executable files and word processing files), otherwise the data will be degraded, and this is where the lossless techniques have a role.

Lossless compression consists of those techniques that guarantee that the restored data is identical to the original file after the compression/decompression cycle. Currently, there are many lossless compression techniques and algorithms. Generally lossless is implemented using two different types of modelling: (i) Statistical, and (ii) Dictionary based, in the first type the data is compressed using the probability of the characters appearance, one of the most known algorithms which uses this type of modelling is the Huffman coding. Meanwhile in the second type, a sequence of characters is replaced by a shorter sequence, LZW is a well known algorithm which use this type of modelling [18].

None of the previous algorithms use Boolean minimisation. The use of Boolean minimisation has not been thoroughly investigated thus far. The first work was done by Augustine and others [21]. The second one was done by Agauan and his colleagues [22]. However, these two works are designed for images only and do not integrate Boolean minimisation with other techniques.

1.4 Encryption categories

Encryption methods have historically have been divided into two categories: (i) Substitution Ciphers, (ii) Transposition Ciphers [12]. In Substitution Ciphers each letter or group of letters is replaced by another letter or group of letters, the oldest known ciphers is the **Caesar Cipher**[12]; When Julius Caesar wanted to send a message to his generals and he did not trust the messenger so he shifted each letter in the message by three letter, C instead of A, E instead of B and so in (shift by 3) [23]. Transposition Cipher, in contrast reorders the letters but to does not disguise them [12]. Although there are many different cryptography systems, two principles are common among all of them.

Cryptography principle 1: Message must contain some redundancy.

Cryptography principle 2: Some method is needed to foil replay attacks. [12].

However, cryptography algorithms are classified into three classes: (i) Symmetric keys algorithms, (ii) asymmetric keys algorithms, and (iii) hash algorithms [20]. In the symmetric algorithms, the encryption/decryption cycle is completed by a key, this key is used to convert the plain text to cipher text, and used once again to convert the cipher text back to plain text. Nevertheless, this technique has a major disadvantage, which is the difficulty of distributing the key safely over the networks [23]. The second type covers this disadvantage by provide the concept of public keys, which was introduced by Whitfield Diffie and Martin Hellman in 1976 [24]. This model use asymmetric keys, one is public for anyone to use it to encrypt data, but they cannot decrypt it, however only the person who has the corresponding private key can decrypt the data. Finally the third type uses a hash function instead of key, this hash function generates a number based on the input and this number is unique to that input.

1.5 Organisation of the thesis

As chapter 1 gave the reader a preliminary introduction of the problem domain and showed the ability to link between compression and encryption, the structure of this thesis will be as follows:

Chapter 2 will introduce a sufficient explanation for each of the Huffman tree algorithm and the Karnugh Maps technique.

Chapter 3 will introduce the proposed algorithm based on the described algorithms in the previous chapter as it will focus on the dissertation thesis.

Chapter 4 will provide dissections of the algorithm stages overall and show the results of the algorithm after explaining the tests, and the tests environment. Finally this thesis will end in chapter 5, which will include some conclusive remarks and show where this technique can be expanded in the future.

Chapter 2 Background

This chapter will provide a sufficient explanation for each of the Huffman tree algorithm and the Karnugh Maps technique.

2.1 Huffman Tree

The Huffman tree was first introduced in 1952 by David Huffman [4]. The trick of this algorithm is to give the most frequent characters which appear in file shorter codes than those characters which have less frequency. The Huffman tree creates a unique variable length code [18]. For example assume that we have a text file which contains the following text:

"XXXXXXXXXXXXXXXXXXXXYYYYYYYYYYYYYYYYYYYZZZZZZZZZZZZZ".

The first thing the Huffman tree algorithm will do is scan the file and count the repetition of each character, after that it will construct a descending probability table for each character, as illustrated in (Figure 1 part A). The next step is to find the summation of the first two entries in the probability table and construct a new entry. According to our example, the first entry in the table was the character "Z" which was repeated 13 times in the file, while the second entry is "Y" which was repeated 19 times. 19 + 13 = 32. The new entry in the table will be now 32 instead of "Y" and "Z", and this entry will refer to both Y and Z, as illustrated in (Figure 1 part B). Note that the new entry is inserted in a position to let the probability table remain ordered in a descending way. The same step is repeated again to add the first two entries in the table, 32 which was the summation of "Y" and "Z", and 20 which represent the repetition of character "X", so 32+20 = 52 to construct a new entry in the table which will refer to both old entries (20 and 52), as shown in (Figure 1 part C). The question now is how will this tree compress this text? In response to this question, we have to explain that after the generation of the tree, the algorithm will theoretically stick a label on each edge of the tree that links any two nodes together. Label 1 will be on all the right edges of the tree, while label 0 will be on all the left edges of the tree, this illustrated in (Figure 2). It is well known that computers represent each English character by 8-bits according to the ASCII encoding system. Based on the previous example "X" was repeated 20 times, "Y" 19 times, and "Z" 13 times. The total characters in the file is 20+19+13 = 52 characters, 52 * 8 bits = 416 bits stored in the computer, before apply the Huffman tree, now lets see the difference after applying the Huffman tree. From (Figure 2) we can conclude that the new coding for X = 0, Y=11, and Z=10. The new size for the file is now (20*1) + (19 * 2) + (13*2) = 84 bits. In comparison with the original size 416, the Huffman tree reduced 95% of the original file size. Each character has a unique code, thus the Huffman tree can unambiguously decode the characters as it reads the streams of the compressed file [18].

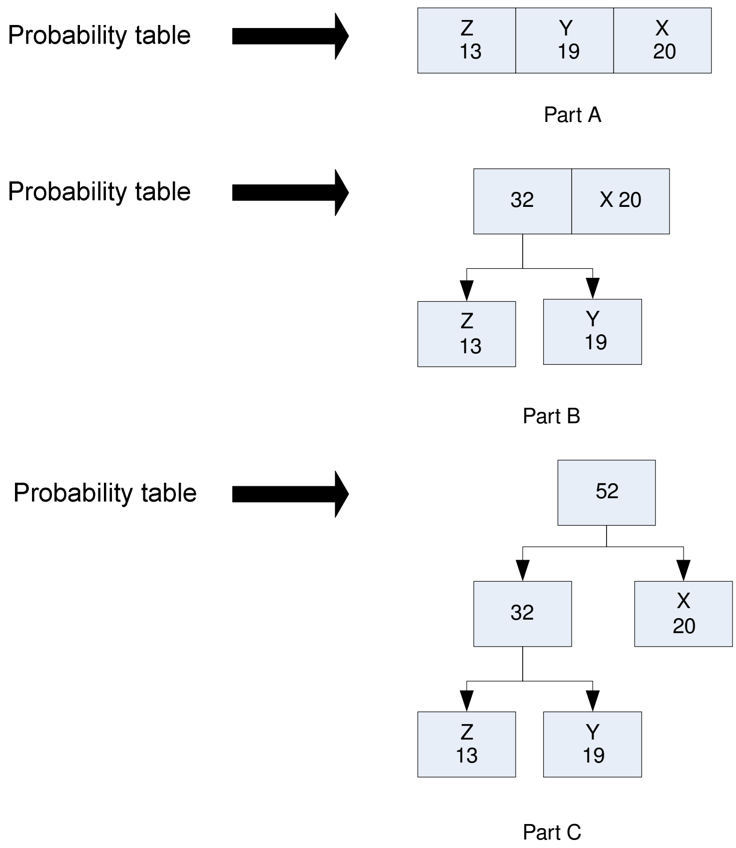


Figure 1: Steps for generating the Huffman tree

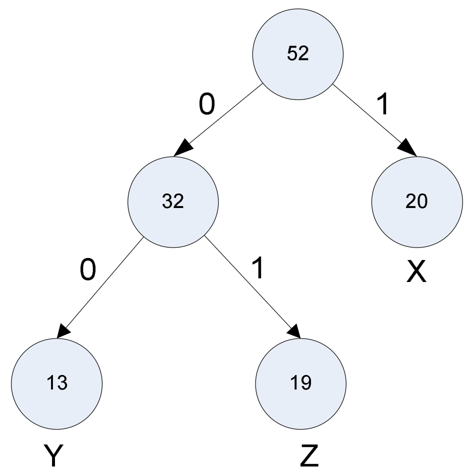


Figure 2: Constructed Huffman tree

As with any algorithm, the Huffman tree suffers from some disadvantages. The Huffman tree is a *statistical* method (see section 1.3), statistical methods (off-line) need to pass over the input file once to gather the statistics, and once again to compress the input file based on the first round, thus they leave an overhead that can be heavy, unlike the adaptive methods (online) which works on the input file as they process it [25].

Another disadvantage of the Huffman tree is that it usually requires transmitting a large coding table on the decompression side as it is an essential requirement to decode the data. For these reasons the Huffman tree has been developed by many researchers, and many variation of Huffman coding have been published, some of them use the same behaviour of the binary tree and others use a unique prefix code. One of the most significant developments of the Huffman tree is the Adaptive Huffman tree [26].

2.1.1 Pseudocode for the Huffman tree algorithm

Compression algorithm (input: text_file)

Begin

```
   Open the file to be compressed.
   While (read != EOF)
   {
// count the repetition of each character.
//Generate a descending probability table for all characters in
the file.
   }
```

```
While(probability_table.size != 1)
{
New_entry = probability_table[0] + probability_table[1]
New_entry.Right_link = probability_table[0]
New_entry.Left_link = probability_table[1]
// Remove the first two entries from the probability table
probability_table[0] = NULL
probability_table[1] = NULL
//insert the new entry
probability_table.insert(New_entry)
//reorder the table in descending order
probability_table.sort(descending)
}
Huffman_tree = probability_table[0]
Return Huffman_tree
```

End

Decompression Algorithm (input: compressed file)

Begin

```
While(compressed_file != EOF)
{
//Read 1 bit from the compressed_file
compressed_file.read(1, buffer)
    if(buffer == 1 )
    {
    Huffman_tree.Walk_to_Right_node;
    If(Huffman_tree == leaf node)
    Output_file. Print(leaf node)
    }
    (buffer == 0 )
    {
    Huffman_tree.Walk_to_Left_node;
    If(Huffman_tree == leaf node)
    Output_file. Print(leaf node)
    }
```

```
}Return Output_file
End
```

2.2 Karnugh Maps

2.2.1 Boolean algebra

Boolean algebra is algebra that deals with binary variables, it was first introduced in 1854 by George Boole [27]. The Boolean variables are designed by letters combined by three basic logic operations (AND, OR, NOT) [28]. The Boolean expression is identified by a sequence of Boolean variables. The bigger representation of the Boolean expressions is the Boolean function which is formed by combination of Boolean expressions. Normally the output of a Boolean function is 0 or 1. For example if we have the following Boolean function: *F*(A,B,C,D) = AB+C`D then AB+C`D is an expression which consists of two parts, AB and C`D which usually called *terms* [28]. A Boolean function can always be represented in a truth table. The truth table representation of a function will show all the possible combinations of the Boolean variables and the output result of the function in each case. As the most important implementation of Boolean algebra is in digital computing and computer chips, a Boolean function can be transformed into a digital circuit which is composed of AND, OR, and NOT gates [28], as illustrated in (Figure 3.).

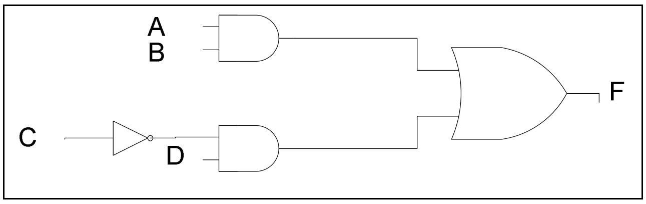


Figure 3: Digital circuit (F=AB + C`D)

2.2.1.1 Minterms and Maxterms

There are two standard forms to represent a Boolean function: (i) product of terms, and (ii) sum of terms [28].This thesis is interested in four variables minterms. Therefore most of the examples will focus only on this category. The product of terms usually represents one combination of the Boolean variables in the function truth table which called *minterm*, and it referred to as m_j where m is the minterm expression and j is the decimal equivalent of the Boolean variables combinations in the minterm truth table. For example consider the following table (Table1):

A	B	C	D	Symbol	m_0	m_1	m_2	m_3	m_4	m_5	M_6	M_7	M_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}
0	0	0	0	m_0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	m_1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	m_2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	m_3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	m_4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	m_5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	m_6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	m_7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	m_8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	m_9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	m_{10}	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	m_{11}	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	m_{12}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	m_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	m_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	m_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 1: Mainterms table for 4 variables

It is observable from the previous table that minterm is an expression which is equal to 1 and it always takes the form of the product of Boolean variables, for example m_{15} = ABCD. We can write a Boolean function on the form of *F*(ABCD) = $\sum$ m(1, 3, 7) meaning that function *F* equals minterms expression 1, 3, and 7. The complete opposite of the minterm is the maxterm. The sum of terms is usually called the *maxterm*. The symbol M_j is used to refer to a maxterm expression, where M is the maxterm expression and j is the decimal equivalent of the Boolean variables combination in the maxterm table, an example of four variables of the maxterm truth table is illustrated in (Table 2). We can find the opposite observation of (Table1) in (Table2), where the maxterm expressions are always equal to 0. Maxterms always take the form of the summation of the Boolean variables, for example M_{15} = A+B+C+D. Same as minterms, we can write a Boolean function on the form of *F*(ABCD) = $\prod$ M(1, 3, 7) meaning that function *F* equal maxterms expression 1, 3, and 7. Since any Boolean function can be transformed into a truth table (see section 2.1.1), we can conclude that any Boolean function can be represented as the sum of minterms.

A	B	C	D	Symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
0	0	0	0	M_0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	M_1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	M_2	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	M_3	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	0	M_4	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
0	1	0	1	M_5	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	1	1	0	M_6	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
0	1	1	1	M_7	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
1	0	0	0	M_8	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
1	0	0	1	M_9	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	0	1	0	M_{10}	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
1	0	1	1	M_{11}	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	1	0	0	M_{12}	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1	1	0	1	M_{13}	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	0	M_{14}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	M_{15}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Table 2: Maxterms table for 4 variables

2.2.2 The Maps technique

The Boolean function complexity is completely related to the Boolean expressions, thus it is important to simplify the function in order to minimise it, and save its cost. However this simplification should not change the meaning of the function, for example let us consider function *F1*= A\`B\`C\`D\` + AB\`C\`D\`, after manipulate of this function by the Boolean algebra rules, we get *F2*= B\`C\`D\`, where *F1* = *F2* and F2 has less variables.

The maps technique is a straightforward method for simplifying Boolean functions. Usually this method is called K-map, or Karnugh map [28]. There are different sizes for the k-maps such as: the two variables k-map which is designed to minimise those functions that have two variables in its Boolean variables set, and the three variables k-map for three variable functions, and so on. However this thesis is interested in the four variables k-map so all of the examples and the analysis in the rest of this dissertation will focus only on this category of k-maps.

K-map is a technique for presenting Boolean functions in its optimal form. It comprises of a box for every variable (represented by a column) in the truth table. All the inputs in the map are arranged in a way that keeps Gray code true for every two adjacent cells [29]. Each box in the k-map represents one minterm, the combination of the row and the column values can identify that minterm value, for example the value of the third column is 11 and the value of the first row is 00, so the combination of them is 0011 which is equal to the decimal value 3, this mean that the minterm in the third column and the first row is m_3. (Figure 4) illustrated 4 variables k-map.

cd \ ab	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

Figure 4: 4 variables k-map

Some of these squares will take the value 1 if and only if the function output is 1 in the corresponding minterm, otherwise it will take the value 0. The idea of k-map minimisation is to group an even number of adjacent squares (horizontally and vertically) that hold the value of 1. As the k-map columns and rows are numbered according to Gray code, this mean that any two adjacent squares will be different only in one bit or one variable in the corresponding minterm, even those squares on the edges of the map. Therefore grouping the adjacent squares lead to avoid the different bits or alternatively the different variables, and thus minimise the Boolean expression. A visual example is illustrated in (Figure 5).

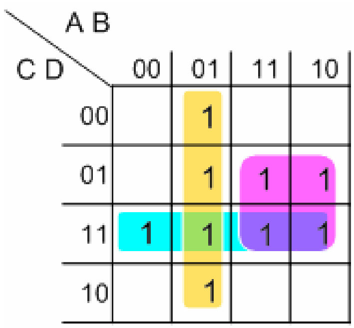


Figure 5: K-map example (F= A'B + CD + AD)

The squares can be grouped as the flowing: one square represents four variables, two squares represents three variables, four squares represents two variables, eight squares represents one variable and the biggest group can be made is 16 square which represents 1 or 0 [30]. The covered minterm in the map called *prime implicant* if it cover the maximum possible of adjacent 1's, and called *essential prime implicant* if there only one unique way to cover it by a prime implicant [30].

2.2.3 Analysis of 4-variable K-map

Since this thesis will focus on the 4-variables k-map, as it will be used in the proposed compression algorithm, this section will provide some observation on this category of k-maps.

The four variables k-map is composed of 16 bits, thus there are 2^{16} possible combinations which might be filled into the map. The four variables k-map simplifies a four variables Boolean function, assuming that the variables are (A, B, C, D). The function's expressions will be composed of a combination of these variables complemented or un-complemented. The minimised term might be composed of 1, 2, 3 or 4 variables, thus the total number of possible combinations is 82 distinct terms. For example, the number of different terms that contain exactly one variable is 10 out of 82 terms. For the input variables (A, B, C, D), the set S_1 that contain exactly one variable will be S_1= {A, A`, B, B`, C, C`, D, D`, 0, 1}. The distribution of these 82 expressions and the number of variables is illustrated in (Table3).

No. of terms	1-variable	2-variables	3-variables	4-variables
	10	24	32	16

Table 3: Distribution of all possible combination Source:[39]

Following is (Table 4) which contains all the possibilities of the minimised terms with the corresponding distinct variables.

1-varaible	2- variables			3- variables				4- variables	
A	AB	A`D	CD	ABC	AB`D	A`CD	B`C`D	ABCD	A`BC`D
B	AB`	A`D`	CD`	ABC`	AB`D`	A`CD`	B`C`D`	ABCD`	A`BC`D`
C	A`B	BC	C`D	AB`C	A`BD	A`C`D		ABC`D	A`B`CD
D	A`B`	BC`	C`D`	AB`C`	A`BD`	A`C`D`		ABC`D`	A`B`CD`
A`	AC	B`C		A`BC	A`B`D	BCD		AB`CD	A`B`C`D
B`	AC`	B`C`		A`BC`	A`B`D	BCD`		AB`CD`	A`B`C`D`
C`	A`C	BD		A`B`C	ACD	BC`D		AB`C`D	
D`	A`C`	BD`		A`B`C`	ACD`	BC`D`		AB`C`D`	
0	AD	B`D		ABD	AC`D	B`CD		A`BCD	
1	AD`	B`D`		ABD`	AC`D`	B`CD`		A`BCD`	

Table 4:All possible combinations for 4 input K-map Source:[39]

These are the only possible terms that can be generated from the 4-variable K-map. Note that a set of these terms will constitute the expressions of the Boolean functions.

Chapter 3 *The proposed algorithm*

This chapter will introduce a sufficient explanation for the different stages of the proposed algorithm based on the described algorithms in the previous chapter.

3.1 The proposed technique

This thesis proposes a new technique to compress data and encrypt it. The proposed technique is based on combining some independent algorithms.

Dissertation thesis: a combination between each of the static Huffman tree algorithm, which is used for text compression and the 4-variables k-map technique, which is used for logic digital minimisation, to invent a new lossless compression algorithm, a manipulation of the Huffman tree was added to encrypt the compressed file.

The use of Boolean minimisation has not been thoroughly investigated thus far. The first work was done by Augustine and others [21]. The second one was done by Agauan and his colleagues [22]. However, these two works are designed for images only and do not integrate Boolean minimisation with other techniques.

The proposed algorithm suggests a lossless data compression with encryption technique that depends on reversing the usage use of k-map. On the compressor side, the input bit-stream is chopped into chunks of 16-bits each, and a minimised expression is found for each of the chunks. The minimised expressions of the input data stream are stored.

Later, the Huffman tree algorithm is applied to the stored expressions "treating each term as a single unit instead of treating each character as a single unit". The obtained Huffman code is used to convert the original file into a compressed one. A shuffle is added to the Huffman tree which dramatically changes it, therefore it cannot be decoded without the identically shuffled tree, this shuffle is based on a generated of key. On the decompression side, the Huffman tree is used to retrieve the original file. The proposed algorithm can be used for various file formats such as images, videos and text.

A full cycle for the proposed algorithm is to compress a file, encrypt it, decrypt it, and finally decompress it back identically to the original file.

3.2 Compression stage

Although the computer provides readable data for humans, such as images, videos and documents, data is stored inside the computer as a stream of 0's and 1's.The proposed method reads any input file, which consists of 0's and 1's, then starts processing every 16-bit (2 bytes) as one block. Each block will be used to fill in a 4-variables k-map and then find the minimal expressions for that block. Between each two k-maps there is a separator character "-". This process is illustrated in (Figure 6).

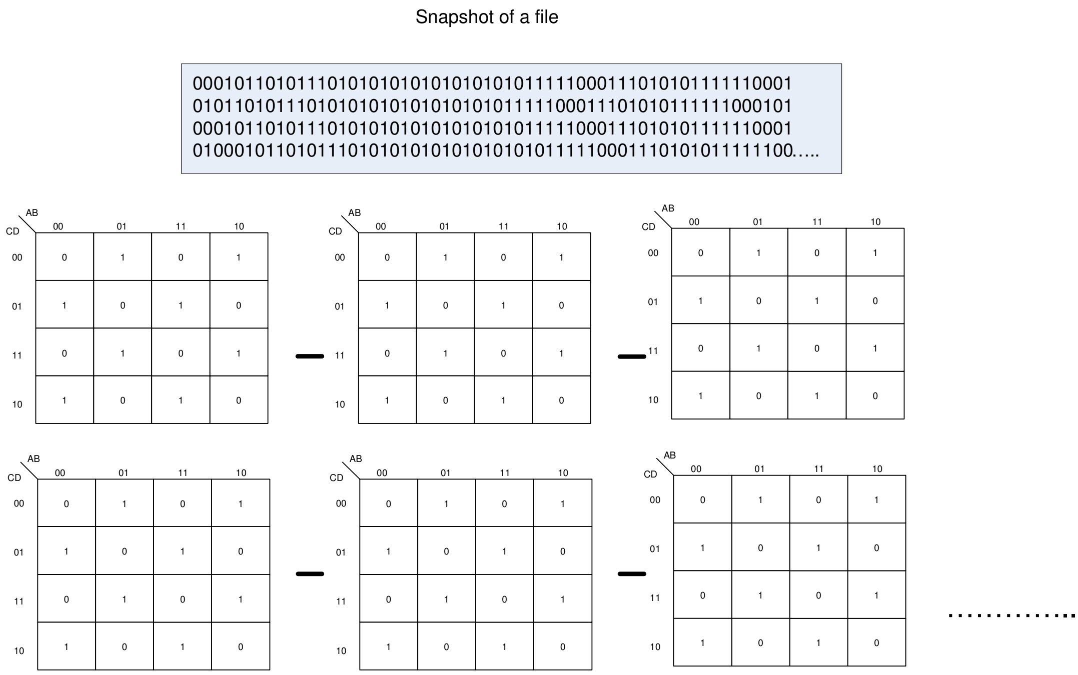


Figure 6: Mapping a file into k-maps

Consequently the binary data is converted into a big collection of Boolean functions separated by a *dash* character. Fortunately the 4-variables k-map produces only 10 distinct variables, and 82 distinct terms (see section 2.2.3), therefore this process will convert the binary data into a limited set of characters. These collections of Boolean functions that have been produced by the k-maps will be called *Intermediate Representation* or *IR*, as it is neither yet the compressed file nor the original file (the file to be compressed). (Figure 7) shows a snapshot for an IR, note that a *dash* character separates every two subsequent Boolean functions.

```
ABCD-A`B`C`D`-A`B`C`D`+ACD-0-AB`CD`-ABC`D-
A`B`CD+A`BCD`+AB`C`D+ABCD-A`B`C`D`+ABC`D`+ABCD-
A`B`C`D+AB`C`D`-A`B`CD`-A`BC`D`+AB`C`D-B`C`D`+ABCD-
AB`CD`-0-A`BC`+B`C`D-ABC`-0-A`BC`D`+A`BCD-
A`BC`D+AB`CD`+ABC`D`-0-ABCD`-A`BCD`-
A`BC`D+A`BCD`+AB`CD-AB`C`D`-
```

Figure 7: Snapshot for an IR

It is not necessary for the size of the input file to be modules of 16 bits, however this issue is not a big deal as the compression procedure will ignore the last remaining few bits "15 bits at most", and put some flags to indicate these bits during the decompression process.

The compression procedure will scan the IR representation to generate a lookup table. This table will include the terms and its frequency in the IR. Owing to the fact that the 4-variables k-map produces 82 distinct terms (see section 2.2.3), the maximum size of the lookup table might be 82 + 1 (the dash character), 83 different entries. However it is not necessary that the lookup table will include the whole 82 elements, "product of terms". Once the frequency lookup table is computed for the whole IR, the compression procedure will pass it to the Huffman tree algorithm in order to generate a binary code for each entry in the table. Applying the Huffman tree algorithm will result in a binary tree, that holds only IR expressions in its leaves, ignoring the "+" sign between different terms in any single Boolean function, and thus it will not be included in the Huffman tree. The generated Huffman code will be used to convert the original file into a compressed one by replacing the expressions in the IR by its corresponding code from the generated Huffman tree. A complete example for a compression process is illustrated in (Figure 8).

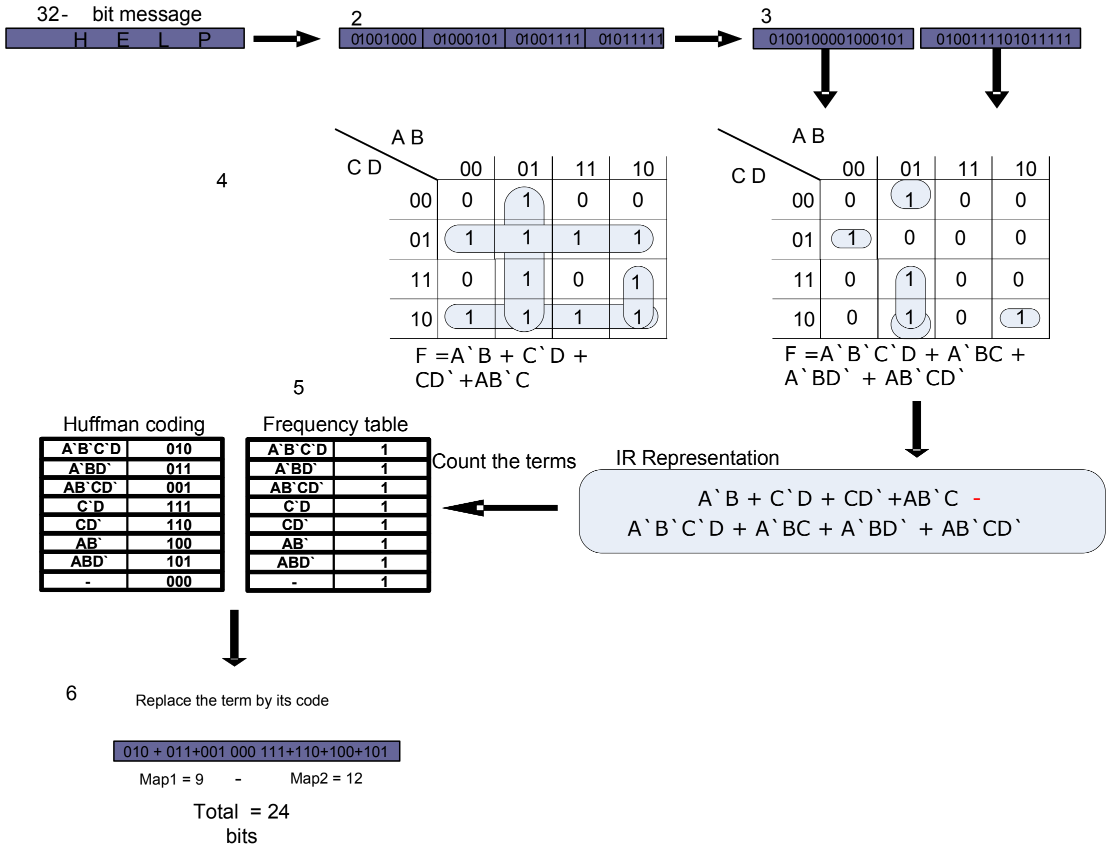


Figure 8: A compression example for a text file that contains the word "HELP"

In the previous figure the word "HELP" was converted into its binary representation and divided into chunks of 16 bits. The blocks have been filled into a 4-variables k–map to assemble the IR representation. In the fifth step the frequency table has been computed, and Huffman code has been generated based on the input from the frequency table. Finally the generated Huffman code will used to convert the 32 bits message file into a compressed one. Note that compress a 32 bits message is not a real life example. However the purpose of this example is to show the different steps through the compression procedure.

3.2.1 The Bad maps problem

The proposed compression technique is based on the idea of converting the binary data into a collection of Boolean functions using the 4-variables k-map, then employs the feature that accompany this category of k-maps -which is, it produce a limited number of expressions- to observe regularities in the produced expressions. Applying the Huffman tree algorithm will reduce the size of the Boolean functions collection, especially given that the surplus expressions will be high.

The 4-variables k-map takes 16 bits and produces one Boolean function. In order to reduce the size of the data, the process of replacing the terms by its equivalent code from the Huffman for each Boolean function tree should cost less than 16 bits, however in some cases the length of the encoded Boolean functions might be higher than 16 bits, which leads to increasing the data size instead of reducing it, for example consider the following sequence of data "0101101001011010", which has the following k-map in

CD \ AB	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

Figure 9: Bad k-map example

(Figure 9). The output of this k-map is “F= A`BC`D` + AB`CD` + A`B`C`D + ABC`D + A`BCD + AB`CD + A`B`CD` +ABCD`”. Let us make an optimistic assumption, that each product term in the previous sum of products is replaced by three bits. As a result, it will take (8x3) = 24 bits. This means that there will be no saving with the compression for those kinds of cases. However, in reality, some of the produced terms might be replaced by more than three bits, depending on the generated Huffman tree. This problem will be called the “*bad maps*” problem.

A solution for the bad maps problem has been proposed. This solution is launched as soon as the compression procedure detects a bad map. The compression procedure will keep track of those bad maps in a *bad map table.* This table will keep the maps bits and a pointer to its location in the file. After the compression procedure finishes scanning the input file, each bad map will be replaced by its corresponding index in the bad map table. Thus we have to convert the indexes into binary numbers of the same length. An example should explain these steps to enforce the understanding of this solution, consider (Figure 10) where the red streams represents bad maps; as soon as the compression procedure detects a bad map situation, it generates a bad map table, where it keeps the red streams and the pointers which refer to them. At the lower part of (Figure10) each red stream has been replaced by its corresponding index -blue stream-. The indexes have been converted to binary numbers, and the length of these binary numbers is the same for all of the bad maps table, note that the first index is "000" not "00", or "0", the reason for this is that the bad maps table in that specific example has seven cases, so an index of three bits has been used.

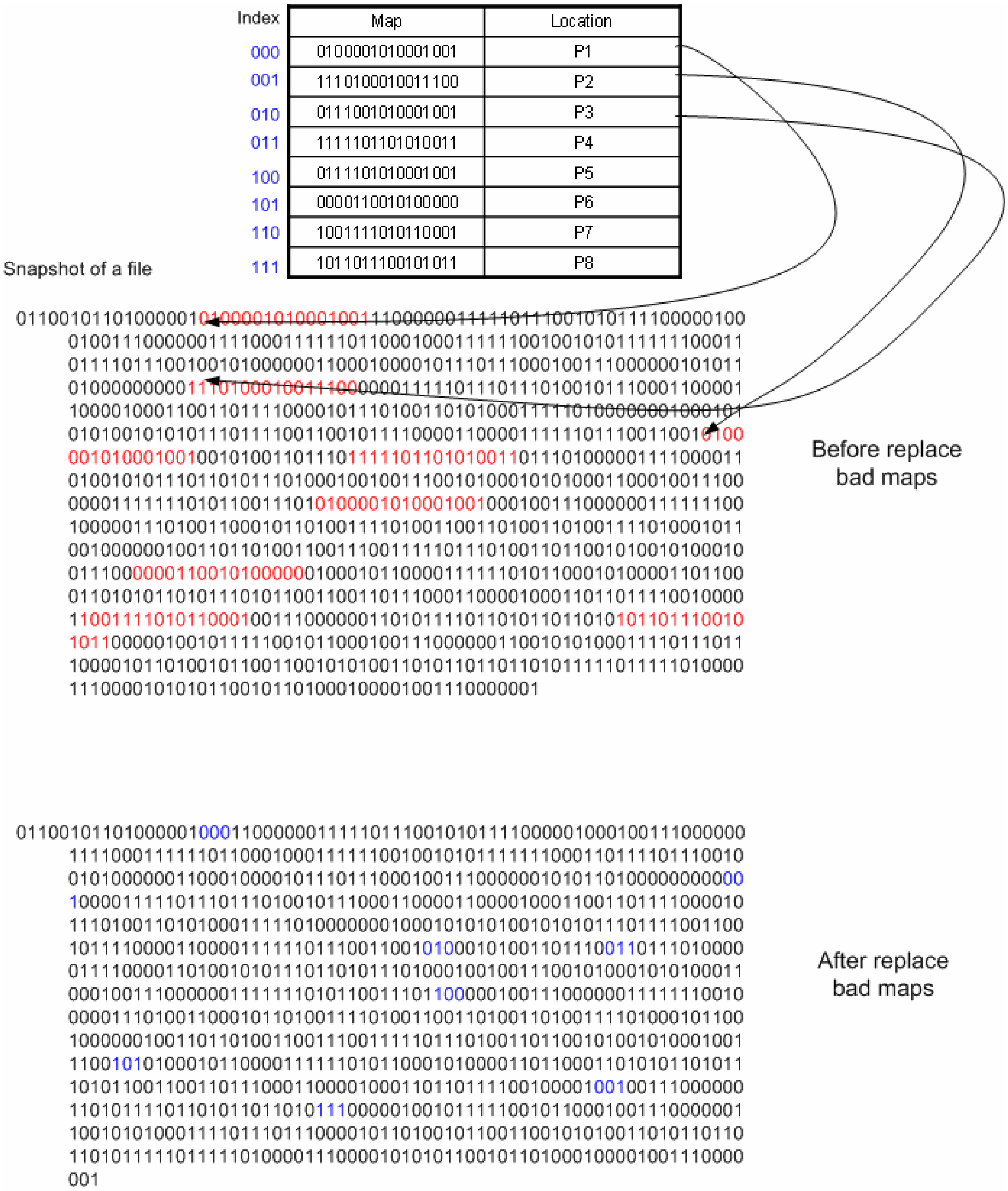


Figure 10: solution steps for bad maps problem

In this specific example three bits are used instead of each bad map, however in the real data this table can be large enough to form binary indexes bigger than three bits, fortunately the length of these indexes will not exceed 15 bits, and thus a saving of at least one bit for each bad map will be provided by this solution. Unfortunately this solution can leave an overhead that can be heavy, as it is difficult for the decompression procedure to decode the bad maps correctly without the bad map table, however this table is composed of several entries, each entry will occupy 16 bits, and thus will not leave a big difference in the compression ratio, even in the small files as the size of this table is related to the size of the input file; note that it is not necessary for this table to exist. (Figure 11) illustrate a flowchart for the compression procedure.

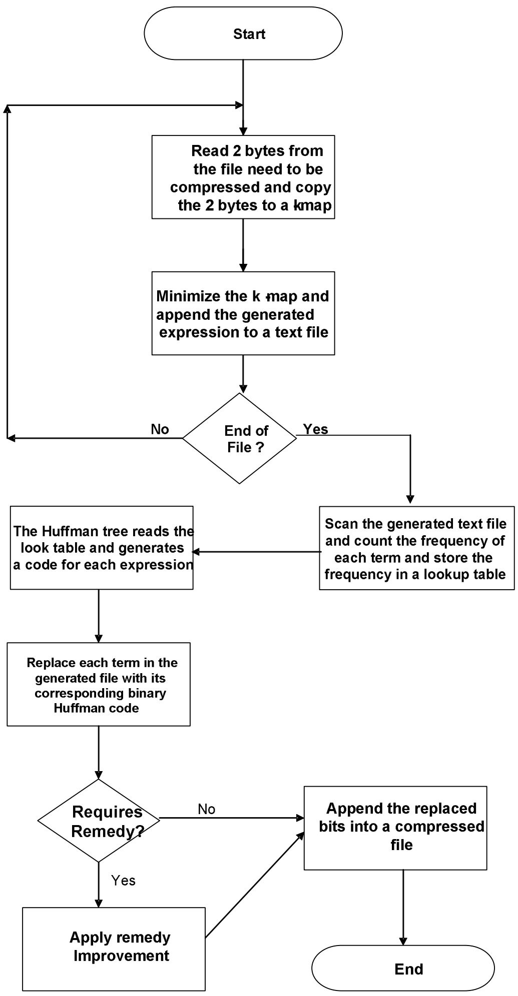


Figure 11: A flowchart of the proposed compression technique

3.2.2 Pseudocode of the compression procedure

```
Compression procedure (Input: File)
Begin
   While (file.read != EOF)
      {
         file.read_streams(buffer, 16)
      //a k-map class, receive a 16 bits and return a Boolean function
      //append the results in IR representation
      IR_representation +=   k-map.get_block(buffer)
      }
   While (IR_representation.read != EOF)
      {
      //get the expressions form the IR
      IR_representation.read(buffer)
      //generate  lookup  table  that  contains  the  expression
      frequencies
      Lookup_table.count (buffer)
      }
//generate the Huffman tree
Huffman_tree.generate(Lookup_table)
//create a new file
Create_file("Compressed" )
While (IR_representation.read != EOF)
      {
      //encode each map in the IR representation and store the
      size of the compressed map in integer variable
      int size = Huffman_tree.encode(IR.get_onemap() )
      If(size > 16 bit) then
      //apply bad maps table solution
      else  //append the compressed stream in the new file
}End
```

3.3 Decompression stage

Based on the previous section, the events of the decompression procedure are predictable. The decompression process requires that the receiver receives the Huffman tree or Huffman code table and the bad maps table -if it is exist- at the beginning, which contains the associated expressions and the corresponding code for them. This seems to be an extra overhead. However, the received Huffman tree contains 83 nodes at most (see section 2.2.3). It is necessary to have a Huffman tree for the decompression process of the received file as it is difficult to decode the data correctly without it. The decompression procedure reads the compressed streams in order to replace them with the corresponding expressions using the Huffman tree. This process should consider the bad maps in the bad maps table, if there are any, and the last few bits, which were not enough to be filled into one k-map in the original file -if there are any-. In other words, the decompression procedure will process the compressed stream by decoding it from the Huffman tree, once it reaches a bad map location, it reads the index stream and replaces it with the corresponding map from the bad maps table. The decompression procedure should ignore the last few bits if the size of the original file was not modules of 16 bit. As a result of this process, the received file will be converted back to an IR representation. Finally from the IR representation we can get back to the k-maps and consequently to the identical original stream. An overall example, which shows a simple text file, containing the word "HELP", and its corresponding compression/decompression cycle as presented in (Figure 12).

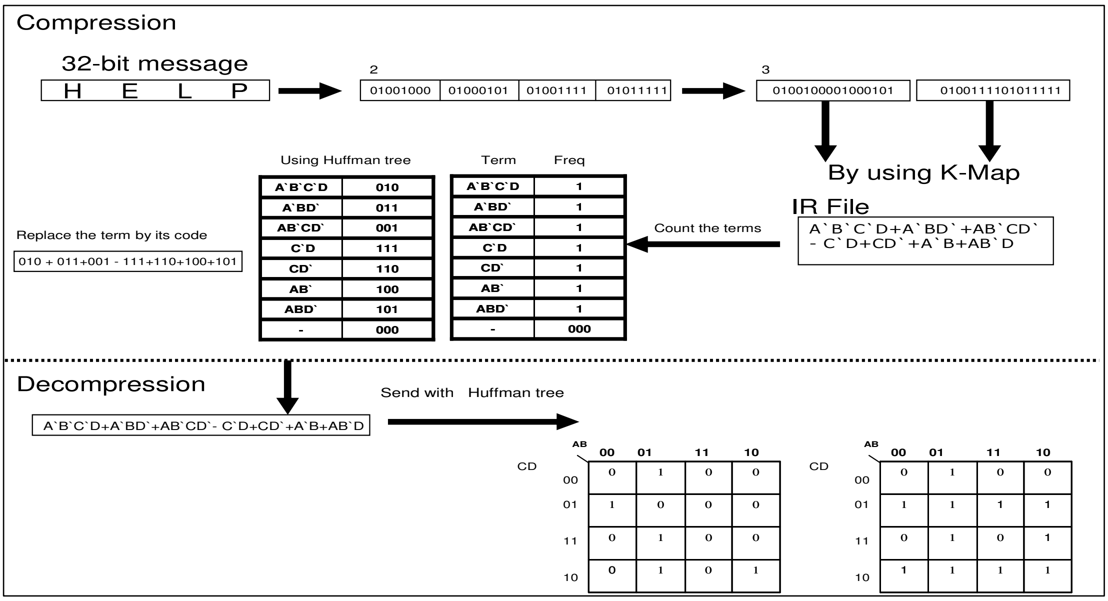


Figure 12: : The complete cycle of compression, encryption and decryption, decompression for a 32-bit message contain the text "HELP"

The decompression process is the complete opposite of the compression. (Figure 13) illustrates a flowchart for this process.

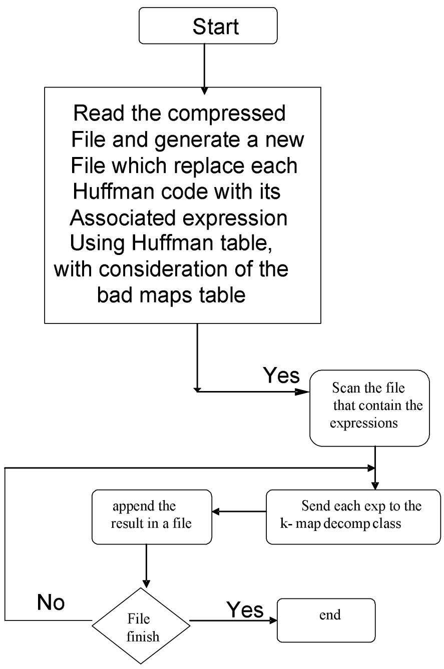


Figure 13: A flowchart of the proposed decompression technique

3.3.1 Pseudocode of the decompression procedure

```
Decompression procedure (Input: compressed_file)
Begin
Create_file("Decompressed")
   While (compressed_file.read != EOF)
{
    file.read_streams(buffer, 1)

if(file.read_streams(buffer,1) = = bad_location)
{
//check the bad maps table
}
Else
{
Huffman_tree.decode(buffer)
if(Huffman_tree == leaf_node)
IR_representation+= Huffman_tree.print(leaf_node)

// append the "+" sign after decode each expression
IR_representation+= "+"
}

//return the IR_representation to binary streams
While (IR_representation.read != EOF)
{
String decompressed_stream = IR_representation.read_one_map();
//return the map to a binary stream

Decompressed+= decompressed_stream
}
}End
```

3.4 Encryption stage

The Huffman tree plays an essential role in the proposed compression procedure (see section 3.2). In the decompression process the Huffman tree is required as it is difficult to decode the data correctly without it. The proposed encryption technique is based on manipulating the weakest ring in the compression/decompression chain, which is the Huffman tree. The idea of manipulation the Huffman tree in support of encryption was introduced by Chung-E Wang [31], however the Huffman tree that the proposed algorithm uses has different characteristics, as the generation of it is based on the IR representation (see section 3.2). The Huffman tree is limited by 83 leaf nodes, therefore the size of the Huffman tree will start from two leaf nodes to 83 leaf nodes.

A shuffle is added to the Huffman tree which dramatically changes it and it cannot be decoded without the identically shuffled tree, this shuffle is based on a randomly generated key, as illustrated in (Figure 14).

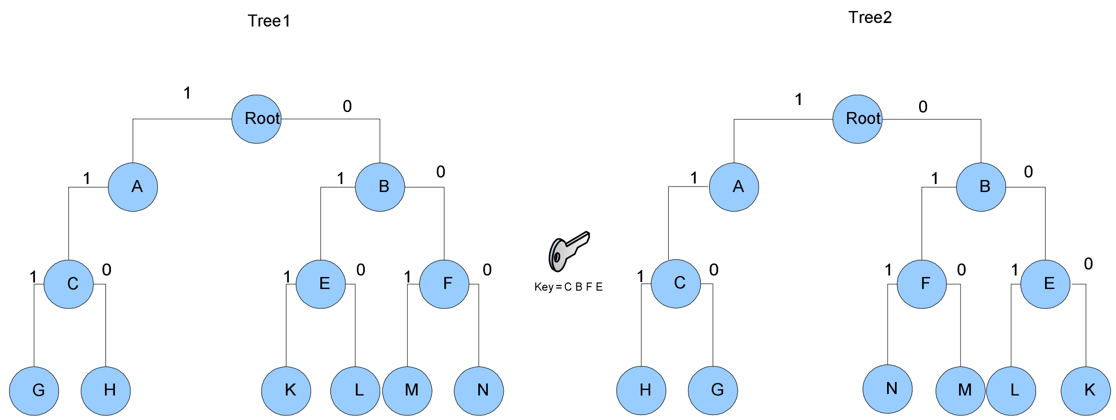


Figure 14: Shuffling the huffman tree

As we can see from the previous diagram, the key value is “CBFE“, this mean that the children of node C will replace each others location as well as the children for nodes B, F and E. The code of node K in tree1 is 011 while the code for the same node in tree2 is 000, thus the difference in coding the same node after shuffling the tree code, results in deformation of the tree structure, and cannot be decoded without the identically shuffled tree. Owing to the fact that the lookup table encodes 83 different entries (see section 3.2), there are a huge number of possibilities to shuffle the Huffman tree.

In order to find all the possibilities of the key that might be generated, let us try to decode a compressed stream without knowing the Huffman tree, we would try to guess our own tree, starting with a tree which has two leaf nodes, until the tree is composed of the whole 83 nodes, and in each tree we will try to put a different combination of terms from the 82 that might be produced out of the k-map (see section 2.2.3). Thus the number of possibilities to shuffle binary tree is $2^{i-1,}$ where "i" is the number of leaf nodes in the binary tree. In each tree 83 different permutations might be possible. Therefore we can say that the total number of possible encryption keys is;

$$\sum_{i=2}^{83} \binom{i}{83} * 2^{i-1},$$

Where "i" define the number of leaf nodes in the tree.

The permutation of $\binom{i}{83} = \frac{83!}{i!(83-i!)}$ forms a huge encryption key. However a huge key does not necessary means absolute security. The proposed encryption technique provides a level of security for the compressed streams, unfortunately, like many other encryption algorithms there is no guarantee of absolute security. Note that the proposed encryption method is classified as a symmetric key encryption algorithm (see section 1.4), since the same key is used in both the encryption and decryption process.

3.5 Decryption stage

Based on the previous section, the events of the decryption process are predictable, as it is the opposite of the encryption. The decryption process requires that the receiver receives the key, in order to find where the nodes are which need to be shuffled back. Note that the order of the received key is not important as shuffling the nodes back in a different order does not make a difference. For example, based on (Figure 14), where the key value was “CBFE”; if the receiver received the key in different order like "FBCE", the decryption process would complete successfully. The decryption procedure would go to the children of node F and shuffle them back, as well as node B, C, and E. Once the procedure finished reading from the key, all the nodes will be already shuffled back.

Chapter 4 Evaluation

This chapter will trigger a dissection of the overall algorithm stages, showing the criteria that affect the algorithm performance, and show the results of the algorithm after explaining the tests, and the tests environment.

4.1 Overall Algorithm Stages

The proposed algorithm consists of four parts: (i) Compression procedure, where the data to be compressed, (ii) Encryption procedure, (iii) Decryption procedure, and finally (iv) Decompression procedure. The input file starts in the compression stage then manipulates the Huffman tree to protect the data, on the receiver side for the Huffman tree to be returned and decompress the data, a complete cycle of the proposed algorithm stages is illustrated in (Figure 15).

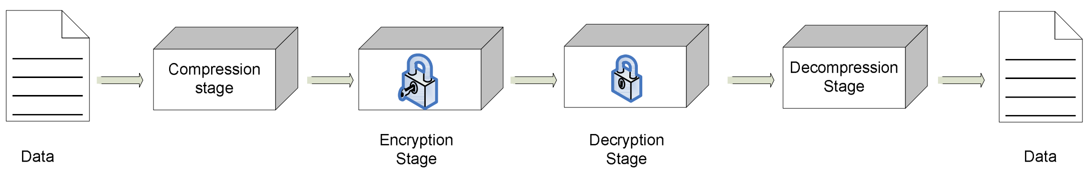


Figure 15: The complete stages of the proposed algorithm

4.2 Compression performance criteria

There are three factors that distinguish the compression algorithms: (i) compression ratio, (ii) compression time and, (iii) decompression time [3]. The ratio of the compression can be measured by dividing the size of the compressed file over the size of the original file; the smaller the result the better the compression ratio. The speed of the compression and decompression processes can be measured by the number of kilo bytes that have been processed per second as KB/s [3]. To invent a new compression technique which does these three aspects perfectly is a very difficult mission. Therefore the researchers are sometimes required to trade off one or two of these factors in order to increase the other factors. It depends on the environment that the compression algorithm works with. For example in the archiving system the data is to be compressed often and it is rarely or never to be decompressed, in this case it is acceptable that the decompression time be slower than the compression. In a contrary situation when users download compressed data from a remote server, it is important that the decompression time is faster than compression, as the data will be compressed only once and decompressed many times. But sometimes the speed of both compression and decompression is important. For example when the users send compressed data over the network, it is important that both the sender and receiver can process the data in an acceptable time.

Data compression has been defined as the process of observing the regularities in a sequence of data and trying to reduce it (see section 1.1). Based on the suggested definition, the proposed compression technique is based on the idea of unifying the binary data into a limited set of letters in support of increasing the regularities. The 4-variables k-map has been employed for this idea, as it produces a limited number of terms and takes full independent chunks of data -2 bytes- (see section 2.2.3). With the cooperation of the Huffman tree algorithm (see section 2.1), these regular expressions can be minimised.

One of the most important factors in developing a strong compression technique is to have a prior knowledge of the data (see section 1.1), however the proposed technique is assumed to be universal. In other words it can work on any kind of data such as images, text, and multimedia. Therefore it can not predict the input file.

Since the proposed algorithm works with a low level of the data, which are the binary streams, the performance of the compression is related to the order of the 0's and the 1's in the input file. There are three possibilities for the input stream. The most obvious one is that the 1's are more than the 0's in a single block of data (2 bytes). A second is that the 0's are more than the 1's. The third is that the block is a mix of 1's and 0's. Each of these three cases has a different effect on the k-map performance. In the first case, where the 1's are more than the 0's, the k-map will cover the maximum possibility of adjacent 1's, and thus it produces a minimised Boolean function. In the second case, where the 0's are more than the 1's, the k-map will cover only those 1's and produce a small Boolean function, finally in the third case, where 1's and 0's are diverse, the k-map will produce a big Boolean function. A Boolean function composed of eight different expressions is the worst form it can be (see section 3.2.1). (Figure 16) illustrates a sample of the three cases that might be filled into the k–maps.

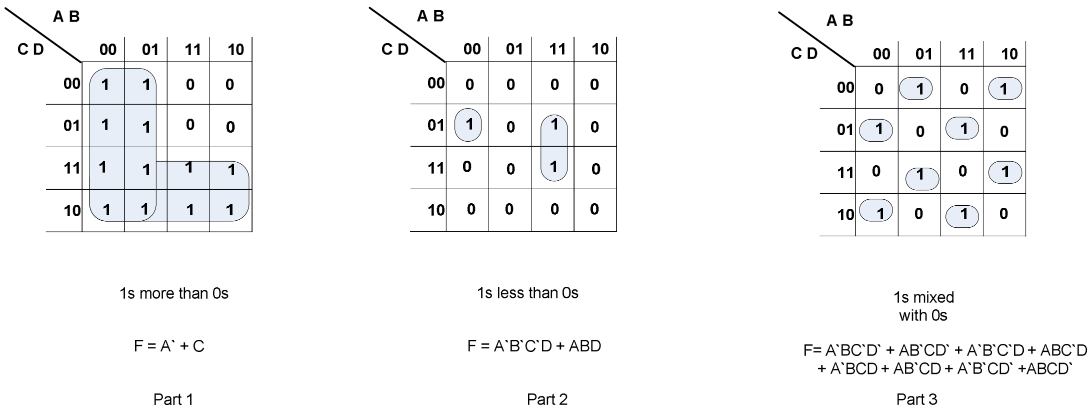


Figure 16: The tree cases of the input stream

(Figure 16) shows a sample of the discussed three different cases, however there are 2^{16} possible entries to the 4-variables k-map. A computer programme has been developed to generate the whole 2^{16} possibilities and run a statistical analysis on the produced Boolean functions. The results of this statistics have been illustrated in (Table 5).

No. of terms	1-term	2-terms	3-terms	4-terms	5-terms	6-terms	7-terms	8-terms	Total
	666	3948	18640	27064	12224	2536	432	26	$2^{16} = 65536$

Table 5: Number of terms in each possible block

It observable from the previous table that there are:

- 1.1% Boolean function which consists of 1-term.
- 6% Boolean function which consists of 2-terms.
- 28.44% Boolean function which consists of 3-terms.
- 41.296% Boolean function which consists of 4-terms.
- 18.652% Boolean function which consists of 5-terms.
- 3.86% Boolean function which consists of 6-terms.

- 0.659% Boolean function which consists of 7-terms.
- 0.039% Boolean function which consists of 8-terms.

As the Huffman tree will encode only 83 entries (see section 3.2), the longest depth of the generated tree is 7 bits. Note that it is not necessary that the encoded IR representation include all 83 entries. Assume for simplicity that each term will be encoded into 4 bits from the corresponding code that is generated by the Huffman tree. Note that this assumption is not realistic as each term might have a different length.

Based on this assumption a Boolean function which is composed of 1-term will be encoded by 4 bits, 2-terms encoded by 8 bits, 3-terms by 12 bits, 4-terms encoded by 16 bits and so on. This assumption shows that k-maps of size 5, 6, 7 and 8-terms will cost more than 16 bits, and thus leave a frailer block compression. However the proposed solution the "bad maps table" aids this problem (see section 3.2.1). From (Table 5) and the previous assumption it is obvious that the probability of Boolean functions being composed of 5, 6, 7 or 8 terms is 18.652% + 3.86% + 0.659% + 0.039% = 23.21%. In other words there is a 23.21% chance that the proposed compression procedure will use the "bad maps table" solution **(assumption 1)**.

However to assume that each encoded term will cost 4 bits is a little bit pessimistic, therefore we make a new assumption that each term will be encoded by 3 bits from the corresponding code that are generated by the Huffman tree. Based on this new assumption a Boolean function which is composed of 1-term will be encoded by 3 bits, 2-terms encoded by 6 bits, 3-terms by 9 bits, 4-terms will be encoded by 12 bits and so on. With this new assumption the Boolean functions which are composed of 6, 7 or 8 terms are considered to be frailer blocks compression, as they will cost more than 16 bits.

Once again the probability of the Boolean functions being composed of 6, 7 or 8 terms is 3.86% + 0.659% + 0.039% = 4.588%. That means there is a 4.588% chance that the proposed compression procedure will use the "bad maps table" solution **(assumption 2)**. From assumption 1 and 2, there are;

$$\frac{4.588 + 21.23}{2} = 13.899\%$$

that the proposed compression procedure will use the "bad maps table" solution. We can conclude that the compression performance is completely related to the structure of the binary representation of the file.

4.3 Encryption strength

As stated earlier there is no absolute security in the science of cryptography. Unlike many others areas of computer science, cryptograph performance is measured by breakability, thus it is hard to find out whether an algorithm provides a level of security unless it is attacked. A cryptosystem that has never been attacked is not trustable. However the researchers of that field have some criteria to measure the strength of the encryption algorithms and compare their performance. The key size and the randomness of the encrypted plaintext are two major factors to compare the cryptosystems. Two algorithms are comparable if the efforts needed to break them are the same or almost the same for a given resource [20]. The amount of time that required to break a cryptosystem can be measured by calculating

$$2^{k-1}T,$$

Where k is the size of the encryption key,

T is the amount of time for needed to encrypt a plaintext and compare its result with the ciphertext[20].

The proposed encryption technique is classified as symmetric encryption key (see section 3.4), since the same key is used for both encryption and decryption. As stated earlier symmetric key encryption algorithms suffer from a major problem, which is the difficulty of distributing the key safely over the networks, and thus asymmetric key encryption has been introduced in order to cover this weakness. However most asymmetric key algorithms are much slower than symmetric ones, therefore symmetric methods are used to process long messages [32]. The key size of the proposed encryption process is;

$$\sum_{i=2}^{83} \binom{i}{83} * 2^{i-1} \textbf{ (formula 1)}.$$

Where "i" define the number of leaf nodes in the tree.

The permutation of $\binom{i}{83} = \frac{83!}{i!(83-i!)}$ (see section 3.4).

Several studies are made to estimate the cryptosystems security lifetime, these studies considered many factors that might effect the lifetime such as the development of the quantum computing, improved attack technology that might threat cryptosystems, and the development of computers in the future. One of these studies is illustrated in (Table 6), this table has been suggested by NIST [20]. Note that this table focuses on symmetric key algorithms only.

Algorithm security lifetimes	**Symmetric key algorithms**
Through 2010 min. of 80 bits of strength)	2TDEA[23] 3TDEA AES-128 AES-192 AES-256
Through 2030 (min. of 112 bits of strength)	3TDEA AES-128 AES-192 AES-256
Beyond 2030 (min. of 128 bits of strength)	AES-128 AES-192 AES-256

Table 6: recommended algorithms (source: [20] NIST)

After finding out the total number of possible keys, it is possible to include the proposed encryption technique to (Table 5). To append the proposed encryption technique does not means that this security life time estimation is accurate or realistic, as there are many factors which have been ignored, however it gives a rough prediction of the technique strength.

4.4 Method of testing

Like any other new algorithm which has been proposed, it should be tested and its results compared with other existing methods. There are two ways to test a new compression technique: (i) compress well known files and compare the results with previous tests (Empirical), or (ii) create special files and compress them (Analytical) [33] [3]. The first method is useful as it is easy to compare the results with other algorithms and evaluate the general performance of the new technique in comparison with the previous tests. However the second method is valuable as generating special files is a useful technique to address the algorithm behaviour under certain situations, which evaluates the algorithm performance. The two methods will be applied on the proposed algorithm in order to investigate the performance and study the behaviour under different circumstances.

4.4.1 Data sets

There are three well known data sets among the compression researchers: (i) Calgary corpus which was introduced in 1989 [34], (ii) Canterbury corpus which was introduced in 1997 [33] and, (iii) large Canterbury corpus [35]. These data sets are groups of files that are chosen to cover most of the real files. In the first type, the data are quite old as some of them are not in use these days [3]. The second data set was introduced at 1997, a little bit of a newer collection, however this collection faced a major disadvantage, in that all of its files are relatively small in size [3]. This problem was amended in the third type where it is composed of three large files, two of them are English text and the last one is binary data [3].However this data set faces some disadvantages, one of which is that it has a limited collection of files (only three) which is not enough to provide a sufficient test on an algorithm. Another reason is that two out of the three files are English text, while in practice there are many other different languages than English [3].

As stated earlier the definition of the data compression, is the process of observing the regularities in a sequence of a data and trying to reduce it (see section 1.1). Based on the previous definition, this thesis suggests a new data set to measure the performance of the proposed compression technique. The new data set will be called "*probability corps*". This data set aims to study the behaviour of the suggested compression algorithm. Binary streams were generated for experimental purposes in different sizes such as: 1 KB, 10 KB, 512 KB, and 1 MB. Each of these files is composed of both 0's and 1's, which are generated randomly with a specific probability. The new data set manipulates the regularities of the data by changing the frequency of the 1's over the 0's, and thus put the compression procedure under different forms of regularities. For example, if the probability of the generated file was 10%, that means the ratio of the 1's are only 10% in each block (16 bits) and the 0's are 90%, if the probability is 50%, then half of the bits in each block (16 bits) are 1's and the rest are 0's. Finally if the probability were 90%, this means that in each block there are 90% of 1's and 10% of 0's. Note that the suggested data set supports different forms of the three cases that each block might have (see section 4.2–Figure 16). Applying the probability data set will put the proposed compression procedure under stress, and thus will provide a sufficient study of the algorithm behaviour under different circumstances.

4.5 The experiments

Experiments have been conducted in order to investigate the performance of the proposed algorithm. In order to test this kind of algorithm, there are several factors which have to be considered in the programming. One of them is the need to have full control over the variables and the classes' objects. A strong debugging tool is needed in order to provide a flexible trace of the code at the run time. Finally a fast complier is needed to execute the written code. Therefore, based on the previous factors, C++ was used as a programming language and Microsoft Visual studio 6.0 as a development tool to develop the proposed algorithm and test it. The developed code has been tested on computers which have the following specification:

- Windows XP operating system, Service Pack 2
- Intel Pentium 2.00 GHz processor
- 1 GB RAM

The aim of the proposed experiments is to investigate the performance of the compression procedure. At this level the speed of the compression and decompression procedures will be omitted, in order to focus the intention compression ratio. The experiments are classified into two groups: (i) analytical, and (ii) empirical. The first group will apply the probability data set (see section 4.4.1), in order to study the behaviour of the compression procedure under different forms of regularities. The second group will extend the experiments to real data and compare the results with other techniques such as WinZip and WinRar.

4.6 Outgoing results

4.6.1 Analytical experiment

As illustrated earlier the analytical experiments aim to address the algorithm behaviour under certain situations (see section 4.4). Five categories of files have been implemented: 5MB, 1MB, 512 KB, 10 KB and 1 KB. In each of these categories there are nine files which have been generated, starting from 10% until 90% probability of 1's, as illustrated in (Table 7).

Probability of 1's	5 MB	1 MB	512 KB	10 KB	1 KB
0.1	52.9587%	52.0591%	51.9778%	48.5486%	45.7886%
0.2	77.5244%	75.1642%	75.0928%	65.0024%	57.0923%
0.3	92.4778%	92.4805%	92.4477%	73.9136%	60.7178%
0.4	97.467%	97.4629%	97.4434%	80.7263%	57.9712%
0.5	98.3823%	98.3889%	98.3789%	80.7983%	57.5684%
0.6	97.5738%	97.5937%	97.5772%	80.3601%	56.6895%
0.7	95.3191%	95.367%	89.892%	73.5608%	56.9824%
0.8	86.2267%	86.2541%	81.7279%	67.8442%	58.2642%
0.9	72.8143%	69.197%	69.1746%	58.1519%	46.2036%

Table 7: Analytical experiments on the probabilty data set

For example the compression ratio of a 5MB file, which has probability 10% of 1's is 52.9587%, that mean the compression procedure saved 47.0413% of the 5MB. (Figure 17) illustrates a line chart for the previous table in order to observe the changes in the compression ratio.

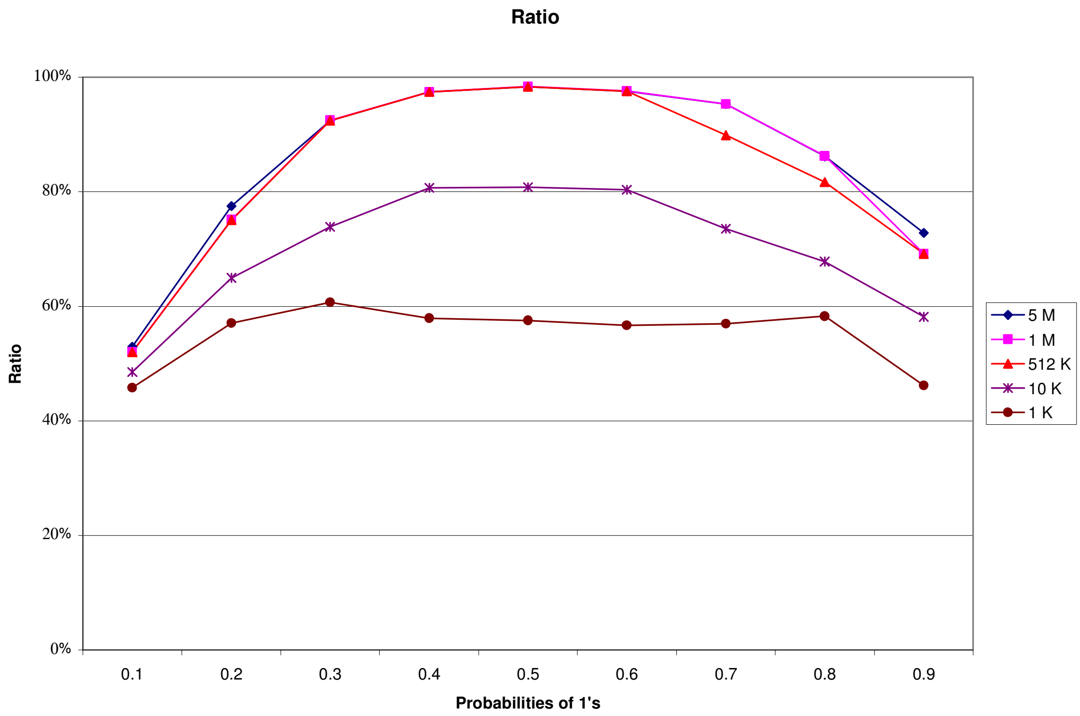


Figure 17: Line chart for the probability set

The x-axis shows the probabilities of the 1's on the generated files, while the y-axis shows the ratio of the file size after applying the compression procedure. Each colour addresses one category of files. For example, if the probability is 0.5, then mostly half of the bits are 0's and the rest are 1's. If the probability is 0.4 it means that the number of the 1's in a 16-bit block is approximately $0.4 * 16 \approx 6$ bits. We should report here that we generate each bit of the 16-bit individually rather than generating the whole number. The reason for this is to get more uniform and fair randomness. The random number generator, which used, is listed in [36]. It is observable from the previous line chart that the lines obtain a curved shape, where they reach their highest level when they intersect with the 0.5 in the x-axis. This show that the compression ratio is worst as the probability of 1's reaches 50%. For example the compression ratio of 5MB and 50%

of 1's is 98.3823%, meaning that there is only 1.6177% saving from the 5MB, which is not encouraging. The reason for this is that when the blocks of the binary data have a diverse order of 0's and 1's the k-maps will produce big Boolean functions. Note that a Boolean function composed of eight different expressions is the worst form it can be in (see section 3.2.1). Consequently the encoding process of the IR representation will lead to less saving for each k-map, and thus the saving ratio will be little. However the improvement of the compression ratio performance is associated with the density of the binary streams. In other words whenever the probability of 1's goes far from the 50% the ratio of compression will be better. The reason for that is the k-maps will produce either a minimised Boolean functions if the 1's were more than the 0's or a short Boolean functions if the 0's were more than the 1's. In both cases the encoding process for the IR representation will result in a good saving for each k-map, and thus result in a high compression ratio.

The compression ratio for large files such as 512 KB, 1 MB and 5MB are almost the same. We have noticed that there are similarities in the compression ratio of the large files. The reason for this is that large files generate almost the same Huffman code for the 83 entries in the frequency lookup table. However the 1KB line behaves differently from the rest of the categories, as the shape of the line is not in the form of a curve. The reason for that is that 1KB is a relatively small file and it reflects the behaviour of the algorithm under a specific circumstance. Owing to the fact that 1KB is the smallest file in the experiment categories, the generated lookup table contains less entries out of the 83 different entries (see section 3.2). Therefore the ratio varies.

We can note that the worst compression performance for the 1KB is with 0.3 where the ratio is 60.7178%. Based on the previous reasons, we can conclude that 1KB is a special case, due to its limited size. Based on the previous experiment, we can conclude that the major factor that affects the compression ratio is the density of the binary streams.

4.6.2 Empirical experiment

As illustrated earlier, the empirical experiments aim to address the algorithm behaviour on the real data files, as it is easier to evaluate the general performance of the new technique in comparison with the previous tests. The experiment included three categories of data. The first category was ASCII code files which represent the English text. The second was Unicode files which can represent different languages other than English, such as Arabic and Chinese. The last category included images, which were stored in the form of JPG. This category contained a collection of coloured and grey images files. In the first two categories the results were compared with, WinZip [16], WinRar [37], and the Huffman tree algorithm (see section 2.1). The JPG category has been compared with only WinZip and WinRar. Note that both WinZip, and WinRar were used in the experiment as they are the most common compression tools used among the users, and they have thousands of adherent supporters all over the globe.

4.6.2.1 ASCII files

ASCII text files have been used to investigate the performance of the compression ratio for general English text files. The experiment results are illustrated in (Table 8). Four files were included in this category. The first column shows the algorithm that has been used to compresses the file. The second column shows the file name that has been compressed. The third one gives a brief description of the data in the file. Note that all of these files are included in (Appendix B). Columns four, five and six are clearly comprehended from the table. The last column illustrates some notes in each case if needed, such as the number of expressions that were found in the IR representation. Note that the maximum possible number is 83 different expressions (see section 3.2).

Algorithm	File name	Description	Size before compression	Size after compression	Ratio	Note
Proposed Compression	pghandbook0506	Essex postgraduate student hand book	2127888 bits	1524940	71.6646%	Expression = 69
WinZip	pghandbook0506	Essex postgraduate student hand book	2127888 bits		26%	
WinRar	pghandbook0506	Essex postgraduate student hand book	2127888 bits		23%	
Huffman coding	pghandbook0506	Essex postgraduate student hand book	2127888 bits		62.69%	
Proposed Compression	Assembler.txt	2117 lines of java code, "old SP Assigmnet"	643088 bits	421808	65.591%	Expression = 54
WinZip	Assembler.txt	2117 lines of java code, "old SP Assigmnet"	643088 bits		15%	
WinRar	Assembler.txt	2117 lines of java code, "old SP Assigmnet"	643088 bits		15%	
Huffman coding	Assembler.txt	2117 lines of java code, "old SP Assigmnet"	643088 bits		60.7%	
Proposed Compression	design doc.txt	Design Document from group project course	1585872 bits	1039180	65.5272%	Expression = 59
WinZip	design doc.txt	Design Document from group project course	1585872 bits		19%	
WinRar	design doc.txt	Design Document from group project course	1585872 bits		17%	
Huffman coding	design doc.txt	Design Document from group project course	1585872 bits		60.2%	
Proposed Compression	website_URL.txt	text file that contain a URL	400 Bytes	178 Bytes	44.5%	Expression = 30
WinZip	website_URL.txt	text file that contain a URL	400 Bytes		100%	No compression
WinRar	website_URL.txt	text file that contain a URL	400 Bytes		100%	No compression
Huffman coding	website_URL.txt	text file that contain a URL	400 Bytes		99%	

Table 8: Empirical experiment on English text files

A bar chart has been extracted from the previous table in order to provide a clear observation on the ratio differences between the algorithms, and analyse the results. This chart is illustrated in (Figure 18).

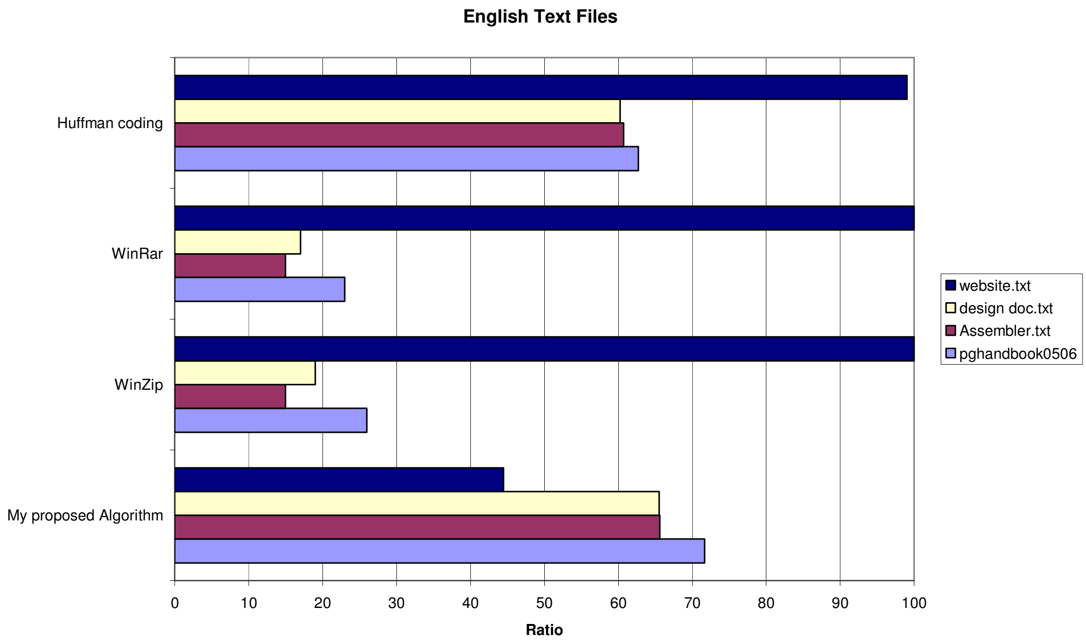


Figure 18: Empirical experiment on English text files "Bar chart"

The x-axis shows the ratio of the file after applying the compression algorithm, while the y-axis shows the compression technique that has been used. Each bar has a unique colour in order to identify one file. For example the light blue bar represents the file which has the name "pghandbook0506". The intersection of the bars with the x-axis shows the ratio of the file after its compression. For example the compression ratio for "pghandbook0506" by the proposed compression technique is 71.6646%, meaning that it saved 28.3354% of the original file size, while the same file has a shorter bar in each of the WinZip, WinRar and Huffman tree, which means that the compression ratios there is better than the proposed compression technique for this file.

It is obvious from the previous chart that the proposed compression technique has a bad compression ratio in comparison with WinZip, WinRar, and the Huffman algorithm. For example the first file in (Table 8), which has the name "pghandbook0506", was compressed until 26% and 23% by both WinZip and WinRar, and 62.69% by the Huffman tree algorithm. However the ratio of compression by the proposed compression technique was 71.6646% which is less than WinZip, WinRar, and the Huffman tree by 45.6646%, 48.6646% and 8.9746%. The proposed compression technique saved 28.3354% from the original file size, which is considered as a relatively acceptable saving. However this ratio is low in comparison with other techniques that are designed specially to compress text.

In order to analyse these results a snapshot of the files was taken, to observe the binary streams density. The snapshots are illustrated in (Appendix B). We conclude that the reason for the low compression ratio for the English text files is that the ASCII code occupies 8-bits for each character of the computer memory. Therefore each k-map will process two characters as it receives 16 bits. Owing to the fact that English text is stored as a ASCII representation, each 8-bit represents one unit, and each 4-variable k-map will process 16-bit (two units). This issue puzzles the density of the binary streams of the data. Another reason for the low ratio is that the number of possible characters in an English text file is 26 as per the English alphabet plus special characters like: commas, and brackets; estimating that the total number of possible characters that might appear in an English text file is 60 characters. The proposed compression technique unifies the binary data to 83 expressions (see section 3.2). Therefore the maximum size the

Huffman tree could be is 83 leaf nodes. While the Huffman tree algorithm encodes only the characters that appeared in the input file, therefore the maximum size of the Huffman tree there might be is 60 leaf nodes. Consequently the compression ratio performance of a small Huffman tree is better than a large one.

The file which has the name "website.txt" and size 400 bytes shows a different behaviour, than the other three files. After applying the proposed compression technique it results in a compression ratio of 44.5% which means that it saved 55.5% of the original size of the file which is better than WinZip, WinRar and the Huffman tree. Linking that to the results of compressing 1KB in the analytical experiments (see section 4.6.2), we can conclude that the proposed compression technique results in a high compression ratio in the small files. Therefore the behaviour of the proposed compression technique is different for the small files.

4.6.2.2 Unicode files

The second category that has been applied on this experiment was Unicode. Unicode text files have been used to investigate the performance of the compression ratio for different coding systems other than the ASCII. A text files that contained Arabic language text have been used. The experiment results are illustrated in (Table 9). Three files were included in this category. Note that all of these files are included in (Appendix B).

Algorithm	File name	Description	size before compression	Size after compression	Ratio	Note
Proposed Compression	Arabic_text.txt	arabic text, letter	17424 bit	9840 bit	56.4738 %	Expressions =58
WinZip	Arabic_text.txt	arabic text, letter	17424 bit		39 %	
WinRar	Arabic_text.txt	arabic text, letter	17424 bit		35 %	
Huffman coding	Arabic_text.txt	arabic text, letter	17424 bit		60.46 %	
Proposed Compression	Arabic_news.txt	Arabic text from news website	3436560 bit	4838900 bit	58.8287 %	Expressions =52
WinZip	Arabic_news.txt	Arabic text from news website	3436560 bit		1 %	
WinRar	Arabic_news.txt	Arabic text from news website	3436560 bit		1 %	
Huffman coding	Arabic_news.txt	Arabic text from news website	3436560 bit		49.5 %	
Proposed Compression	Arabic_book.txt	Arabic e-book	4618256 bit	2741140 bit	59.3545 %	Expressions =28
WinZip	Arabic_book.txt	Arabic e-book	4618256 bit		29 %	
WinRar	Arabic_book.txt	Arabic e-book	4618256 bit		3 %	
Huffman coding	Arabic_book.txt	Arabic e-book	4618256 bit		48.4	

Table 9: Empirical experiment on Unicode text files "Arabic language"

The structure of the table is the same as (Table 8). A bar chart has been extracted from the previous table in order to provide a clear observation on the ratio differences between the algorithms, and analyse the results. This chart is illustrated in (Figure 19).

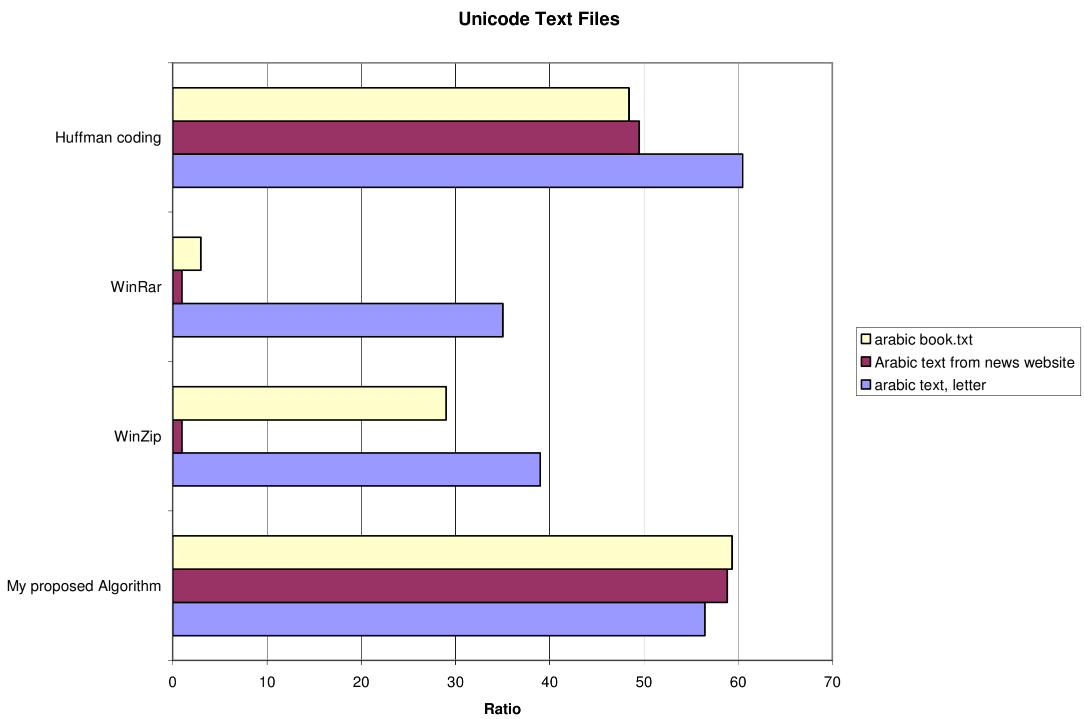


Figure 19: Empirical experiment on Unicode text files "Bar chart"

Generally the compression ratio of the Unicode files are better than the ASCII code or English text files, as the number of expressions that were found in the IR representation for the Unicode files were less than the expressions that were found in the ASCII files. However the performance of the compression is considered to be low in comparison to WinZip, WinRar, and the Huffman tree. For example both WinZip and WinRar saved 99% of the file size which has the name "Arabic_news.txt", while the proposed compression reduced its size until 43.5262%, which makes a 55.4738% difference between WinZip, WinRar and the proposed compression. Saving 43.5262% of a file size is consider to be a satisfactory achievement, however this is not the best result in the text compression business. In order to analyse these results a snapshot of the files was taken, to observe the binary streams density. The snapshots are illustrated in (Appendix B). Once again the low compression ratio is due to the puzzled binary streams. However the Unicode unites are occupying 16-bit of the computer memory and each k-map process a block of 16-bit. This factor supports a better compression ratio than the files which were stored in the ASCII code representation.

4.6.2.3 JPG files

The final category that has been applied to this experiment was JPG images. This type of data has been used to investigate the performance of the compression ratio on the images. The experiment results are illustrated in (Table 10). Five files were included in this category, two of them are grey images, and the remaining three are coloured images, in order to study the proposed compression behaviour under different images which have different outward appearance. Note that all the images are included in (Appendix B).

Algorithm	**File name**	**Description**	**size before compression**	**Size after compression**	**Ratio**	**Note**
Proposed Compression	ahmed.jpg	Face picture	40976	25703	62.727%	Expressions =83
WinZip	ahmed.jpg	Face picture	40976		88%	
WinRar	ahmed.jpg	Face picture	40976		83.34%	
Proposed Compression	mar17a6ey.th.jpg	mix color picture	33808	23332	69.0133%	Expressions =82
WinZip	mar17a6ey.th.jpg	mix color picture	33808		99%	
WinRar	mar17a6ey.th.jpg	mix color picture	33808		99%	
Proposed Compression	Rhinozeros.jpg	gray picture	710672	656004	92.3076%	Expressions =82
WinZip	Rhinozeros.jpg	gray picture	710672		98%	
WinRar	Rhinozeros.jpg	gray picture	710672		98.16%	
Proposed Compression	Black White 10.jpg	gray picture	186384	161236	86.5074%	Expressions =82
WinZip	Black White 10.jpg	gray picture	186384		89%	
WinRar	Black White 10.jpg	gray picture	186384		89.42%	
Proposed Compression	Fractal.jpg	mixed color image	1399824	524011	37.4341%	Expressions =65
WinZip	Fractal.jpg	mixed color image	1399824		97%	
WinRar	Fractal.jpg	mixed color image	1399824		97%	

Table 10: Empirical experiment on JPG images

The structure of (Table 10) is not much different from the last two tables. Once again a bar chart has been extracted from the previous table. In order to observe the change in the compression ratio, this chart is illustrated in (Figure 20).

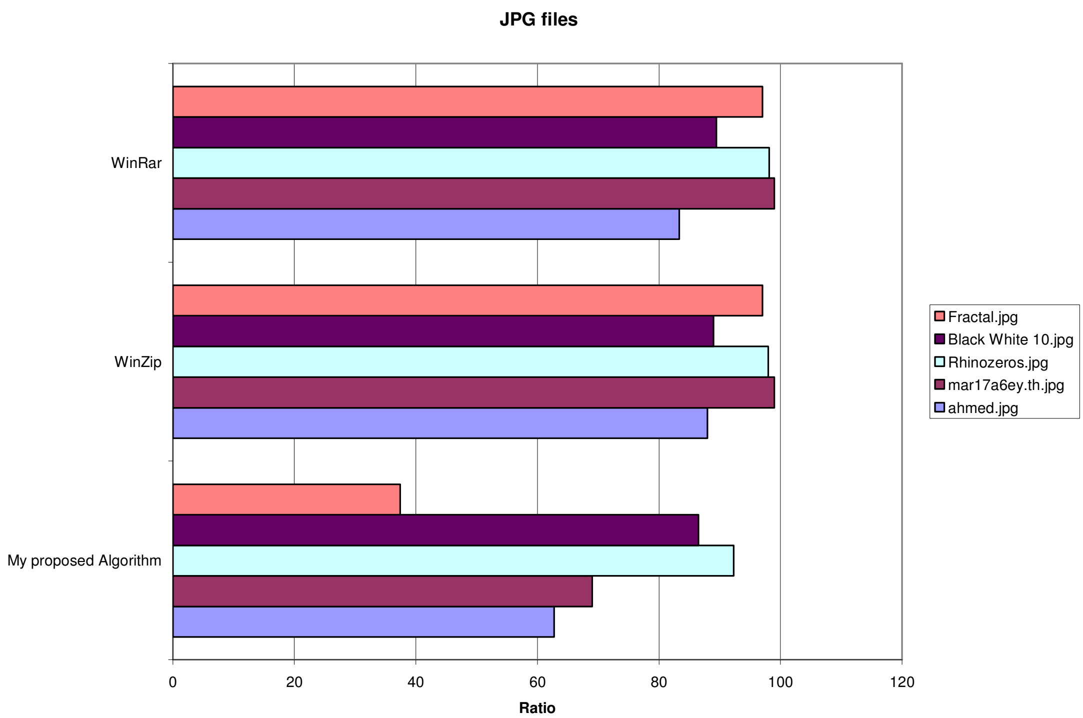


Figure 20: Empirical experiment on JPG images "Bar chart"

It is observable from the previous chart that the proposed compression technique has superior performance on the images in comparison with both WinZip and WinRar. After analysing the snapshots of the binary streams of the data, it is observed that the streams density was clustered. The reason for that is any image would have some logic in its appearance. Therefore the pixels that describe the image would take some gradation to draw it. Consequently the binary stream that represents these pixels would have the same gradation, and thus would reflect on the compression ratio performance.

For example let us consider the first file in (Table 10) which has the name "ahmed.jpg". Note that it is a coloured image. WinZip saved 12% of its original size, WinRar saved 16.66%, while the proposed compression saved 37.273%. Generally the bar chart in (Figure 20) shows shorter bars in the proposed compression technique section, than the other two sections. Which mean that the proposed compression technique provides a higher compression ratio than both WinZip and WinRar for the images.

We can conclude that the new compression technique provides a good compression ratio when applied to images, due to the high density of its binary streams.

Chapter 5 Conclusion

This chapter will conclude with some conclusive remarks regarding the proposed algorithm, and show where it can be expanded in the future.

5.1 Conclusion

Writing the last chapter is much harder than writing the first one. In the first chapter the objective was clear, as it paints a rosy picture of what the thesis intends to do. However when the readers finishes reading and reaches the last chapter, there is no scope for lily gilding. They have formed an opinion to validate the usefulness of this work.

We proposed a new universal lossless compression technique with a built-in encryption by combining each of the static Huffman tree algorithm which can be used for text compression and 4-variables k-map technique which is used for logic digital minimisation. Then we changed the structure of the Huffman tree to encrypt the compressed file. The proposed algorithm consists of four steps: Firstly compress a file, secondly encrypt it, thirdly decrypt it, and finally decompress it back identically to the original file.

This thesis suggested a definition for the compression as the process of observing the regularities in a sequence of data and trying to reduce it. A problem of frailer blocks compression arose with regard to the compression process. However this problem was amended.

This thesis suggested a new data set to measure the performance of the proposed compression technique which was called "*probability corps*". Experiments have been applied in order to investigate the truth of what was proposed regarding the lossless compression. The experiments were divided into two groups. The first was analytical, which aimed to investigate the factors that affect the compression performance, and the second was empirical, which explored the compression performance on three types of data, ASCII files, Unicode files and JPG images. The results were compared with other techniques. It has been concluded from the experiment's results that the major factor which affects the performance of the proposed compression technique ratio is the density of the binary streams for the data. The experiments showed an acceptable compression ratio for both of the ASCII and the Unicode files, however it was not as effective as the other existing techniques. The experiments showed encouraging results for the JPG files which was better than many other techniques.

The encryption strength and its measurement criteria were considered in this thesis. We concluded that the key size and the randomness of the encrypted plaintext are two major factors to compare the cryptosystems. This thesis proposed a symmetric encryption key technique. The key size of the proposed encryption is;

$$\sum_{i=2}^{83} \binom{i}{83} * 2^{i-1}$$

Where "i" define the number of leaf nodes in the tree.

The permutation of $\binom{i}{83} = \frac{83!}{i!(83-i!)}$.

5.2 Future Works

Many interesting questions were raised during this thesis, and have been answered by the discussions and the experiments that were reported. Some of these discussions have raised further new challenging question which need additional efforts to investigate their answers. The research on this algorithm can be expanded on the following points:

- Investigate different categories of k-maps, such as 2-variables and 3-variables k-map.
- Expand the experiments to the multimedia data format such as AVI, Mpeg, and WAV. As the binary streams density of these types of data is assumed to be high.
- Study the 2^{16} possibilities, and let the k-map decide whether to cover the 0's or the 1's depending on the number of 1's in each block (Adaptive decision).
- Investigate the issue of recompressing the data.
- Study the compression/decompression speed.
- Investigate the *bad maps* problem and improve its solution.
- Apply different compression algorithms on the proposed data set "*probability corps*"

Appendix

A

This appendix will include different snapshots of the probability corps data set showing how the density of the binary streams varies, as the probability of 1's change.

- **10% of 1's**

0000000000100000100000000000000001000000000110000000000000000000000000000000000000010000001000
0000000000000110100100101000000010000000011000000000000000000000001000010000001000000001001
0000001000000000000000000001000000000000000000100110100000000001000100000000000000000000000
0001000000001000000001001000000000100000000000000000000000000000001000000000000001000000100
0001010000001000000000000000101000010100000000000010011000000001100000000000000001110100000000
0100010000000100101000000010000000000000000001010010100100000000000001000000001000000000000
0000000000010000010000001010000000100000000000000000001110000000000000000000000100000000000
0000000001001000000000000000000000000000000010001000000000100000000000010000000001000010000
0000000000000001000010100101100000100000000000000000000000000000001000000000000000000001
0000010000000000000000000001100000000100001000000000000100000000000000001000000000000000
1010000001000000100010000000001000000000000000010010000000100010000000000000000010010000000
0000000000000001000010000110000000000000000000000000000100000000100000000100010000000000
0001100000000000001000000000000000000000010000000000010000000000100000000000000010000000000
0000000000100010001000000000000100000000000000010100000000000100001000000000011000100000000
00000000000010000000010001001000000000000100000100000000000000000000100001000000000100000001

- **20% of 1's**

0000000000100000101000010000000001000000001111000100000100111000010000000000000010000011010
1000000100010111100100101000111010001001011010000000100000000000010000100000010000000001001
0001011000000000000000001001100010000000000001001101000000010010001000000000000000010000000
0001111000010011000011010000000011000000000000100000000010001110000000000000011000001100
0011110001010000000000000000101010010100110100000100110000000110000000001000010111010000000
0100010000000100101000001010001000010000000110101101001000000001000110000000110000000000000
0000000001010000001000000101000000010000001001000001001111000000000010000001000100000000000
0110000001001000000010000000100001000000000100010001000001000000100000110000000010000100000
0010000001000000100001011010110000010000000000000000000000100000000011000010000001001111000001
0010110000000101000000000011100000101100001010000000000110000000000000000101000000000000011
1010100010000001000100000000010000000000100000100110010001000100000000001000000101100010101
000000000000000100001000011000001000000000100000000000001010000000100000010010001000000100000
00111000000000000110000000000010110001001001000000000100000000001000000000000000100000000010
0000000000100010011000000001000010100000000000001110000000000010000100000000001100011000010
0000110000000100000000100110010000000110011100010000000100001000000000110001000100010000000001
0010000110000011000000000000000100001000000001101010000010000000000001000000101010101000000000

- **30% of 1's**

0000000100100000101000010000000001100000001111100110010101111000010000000001100011000011110
101100010001011110010010100011101000100111101100000010000000001000100001000101010001001001
0001011000000000000000000110110001001000000000110110100000001001000100000000011000010000000
1001111000010111000011010000001001100000000001000100000000101000111001000001000001100000110 0
0011110001010010000000100000101010010100110100000100110000000110000000000100101011101000010 1
0100010000000010010100000010100010000100000000110101101001100001001110110000000110010000100 00
0001000000101010011000000010100001001000100110100010100111101000100100100000001000101000000 000
0110000001100110010001000000001000010000000001100010001100001000000100000110000000001000010 000
1010000110000000100101011010110000010001000100100001000001000110110111000100000100111100011
0010110000100101100000000001111001010110000101100000000001100001000000000000101100000000000 11
1011100010010001000101001010110000010000100000100110010101000101001000011000000010110001111
0000000000000001000010000110000010011000010000100000011111000000100000010010001000101100 00
0011101000000000101100001000000111110010010010000000001010000000001000000010000111000000000 10
1001000001010001001100000000100001010010000001010011100000100000101011010010000011000110001 10
0000110100001000000001001100100000000110011100010001110100001000011001100010101000100000001

- **40% of 1's**

0100010110100000101000010000000001110000000111110011001010111100001000010000110001100001111 0
1011000100010111100100101000111110001101111011000100101000000001000100001000101110001101001
0001011000000000000000000110110011011100000010011011011001000100100010000100001100001000110 0
1001111000010111010011010100011111000000000010001010101001010101110010000110010011000001100
0011110101110010000000101001101010110100110100000010011010000001110000000101001010111011000 111
0100010000001110010100000010100011000100010001101111010011010010011101110000000110010010111 00
1001000000111010011000100101001110010011001101001101001111010001001001000000010011010000000 00
0110001011101111010011011000010000100000000011000100111000010000000100010110000000011101010 000
1010000110000000110101011010111000010001000101110001000000100011011011110010000001001111000 11
0010111000100101100100000101111001010110011101101010100011001010110000000010110110010110011
1011110010010001000111001110110000010010100101100110010101000101011001011000010101110111 11
0100000100000000101011001011000000100110000100001000100111110000000100000001001000100010110 000
1011111011000000110110010100000001111110110110010000100011100000000001000000010000111100000 1010
1001000001111101011100000000100101010010001010110111000000110100111011010010000001110011000 110
1100110101101000000000100110110000001011101111001100111110100100001100110001110100010000000 1

- **50% of 1's**

0110010110100000101000010100010011100000001111101110010101111000010100100101100011100011111 1
1011000100011111100100101011111111000110111101110010010100000001100010000101110111000110110 1
0011111010110100000000001110110011011100000010011111011101110100101110001100001100001000110 0
1101111000010111010011010100011111010000000010001010101001010101110111100110010111100001100
0011111101110011001001010011011101111110110101001101110100000111100001101001010111011010111
0100010010011100101000101010001110010001010111111111111110101100111011100001011001001011111
1001000000111010011000101101001111010011001101001101001111010001011001000000010011011010011 0
0111001111101110100110110010100101100110001100010011100101000000010001011000011111101011000
1010000110110001101010110101110101100110011011100011000010001101101111001000011001111010 11
0110111000100101101110101011110110101100111011010101101110010101100000010010111110010110011
1011111010010101000111101110111000010110100101100110010101001101011011011001010111110111 11

- **60% of 1's**

011101011011001011100011010001001111001101111101110010101111100010100100101100011100011111
101100010001111110010010111111111000110111101110010010100110011101110001011101110001101101
011111101011010000000101110111111011110010101111101111111010110111000110000110001100011110
111111100001111101001101010001111101001000110001010101101110111110111110110011111100001100
011111111111001110100111001101110111110111101001101110100000111100001111011111111111010111
110001011001110010100110101000111001000111111111111111010110011101110010101100100101111
110110010111010011000101101011111010011001101001101001111010011011001000001100111110100110
011100111110111010011011001010011110011000110001001110010100000010001111000011111101011000
101010011011010110101011110111010110111001111110001101001000110110111100110001100111101111
111011100010110110111010101111011110110111101101011110111001010110000110010111110110110011
111111010010101001111111110111010010110101101110010101101101011110111011101111011111
010000111100010101111001011010001011100011010010111101111111001010011001001100100010111100
101111101101000110110010100110111111111011001010011001111000000110001000100001111010011011

- **70% of 1's**

111101011011001011100011011001001111001111111101111011101111101010101101101110011111011111
101110010011111110100111111111101011011110111011111010011001110111001101110111100101101101
011111111011110000110111110111111011111110101111110111111010110111001110101110001100011110
111111110001111101111110101001111110101110011001101010110111011111011111011111111100001100
011111111111011101011110011011111111011110100110111010000111110000111101111111111111010111
110111111011110010100110101000111111001111111111111111111110110011101110110101110100111111
110110010111110011000101101111111010011011101001101001111010011011001000001110111110100110
011100111110111010111110010110111100111011100010111100111000001111011110000111111011111000
101010111011111110101011110111010111111011111111011101001100110110111101111011101111111111
111011110010110111111110101111111110110111101111011110111101010110010110010111111110111011
111111010010101011111111111110101101101101011011100101011011110111101110111011111111111
010000111100011101111001011110001011110011010010111101111111101111011001001101100010111100

- **80% of 1's**

111111111111001011100011011011011111101111111101111111101111111010111101101110011111011111
101110111011111111100111111111111111011110111011111011011001111110111011101111001111101
111111111011111110111111101111110111111101111111111111110111111110011101011100011010110
111111110001111101111110101001111110111111011111101011110111111111011111111111111100001100
011111111111011101111110011011111111011111100110111011000111110000111101111111111111010111
110111111011110011100111101100111111111111111111111111111011001110111011010111011011111111
111110011111111111101011111111110110111111010111011111101101101101110010111011111110110
011100111110111010111110110101101111001110111000101111001110000011110111100001111101111000
101011111011111110101111111111011111110111111111110100110011011011111111110111011111111
111011110110111111111101011111111101111111011111111101111110101100101100111111111111011
111111010010111011111111111111010110111110101111110111101101111011111011101110111111111
010000111101011101111110101111000101111001111011111111111111101111011011001101101010111110
111111011111101101110111011111111111111111101011111001111000101111011011111101111111011111

- **90% of 1's**

111111111111111111100011011111011111101111111011111110111111111111111111011110111111011111
101110111011111111111011111111111111110111111101111101101101111111101110111111111111111101
111111111011111111111111110111111011111110111111111111111111111111111001111111111111110101111
111111111111111111111101010011111111111111111111010111111111111110111111111111111111111101
011111111111111111111111111101111111101111110110111011110111101001111111111111111011111
110111111111111011110111101110111111111111111111111111111011101111111111010111011111111111
111110111111111111111111011111111110110111111110111011111111110110111111011111111111111110111
011111111110111010111111111111111110111111111111011110111010101111011110000111111111111101
101011111011111110101111111111101111111111111111111111101111111011011111111110111111111111
1111111111111111111111111111111111111111111111111011111111111011011011111111111111111011
111111101001011101111111111111111101101111101011111101111011111111111110111111111111111111
010010111111011111111110111111101111111011111011111111111111111111111011011001101111011111111
111111111111111111111111111111111111111111111101011111011111011101111101111111110111111011111

Appendix

B

This appendix will include snapshots for the files that were used in the empirical experiments showing how the density of the binary streams varies, as the data format changes.

1. ASCII files.

File Name	pghandbook0506.txt
Size	260 KB
Source	http://cswww.essex.ac.uk/intranet/students/handbook/pghandbook0506.PDF

Snapshot for the file:

0100001101101111011011100111010001100101011011100111010001110011000011010000101000110001000100
0011010000101001000011010011110100111001010100010001010100111001010100010100110000110100000
1010010101110100010101001100010000110100111101001101010001010010000000100011001010010010011
11010011010010000000101010001001000010001010010000000100100001000101010000010100010000100000
01001111010001100010000000100010001000101010100000100000101010010010101000100110101000101 01
0011100101010000101110001011100010111000101110001011100010111000101110001011100010111000101110 0010
111000101110001011100010111000101110001011100010111000101110001011100010111000101110001011
1000101110001011100010111000101110001011100010111000101110001011100010111000101110001011 10
0010111000101110001011100010111000101110001011100010111000101110001011100010111000101110 00
1011100010111000101110001011100010111000101110001011100010111000101110001011100010111000 10
1110001000000011010100001101000010100100100101001110010101000101001001001111010001000101 01
0101000011010101000100100101001111010011100010000000101110001011100010111000101110001011 10
0010111000101110001011100010111000101110001011100010111000101110001011100010111000101110 00
1011100010111000101110001011100010111000101110001011100010111000101110001011100010111000 10
111000101110001011100010111000101110001011100010111000101110001011100010111000101110001011
1000101110001011100010111000101110001011100010111000101110001011100010111000101110001011 10
0010111000101110001011100010111000101110001011100010111000101110001011100010111000101110 00
1011100010111000101110001011100010111000101110001011100010111000101110001011100010111000 10
111000101110001011100010111000101110001011100010111000101110001011100010111000101110001011
1000101110001011100010111000101110001011100010111000101110001011100010111000101110001011 10

File Name	Assembler.txt
Size	79 KB
Source	Self writing

Snapshot for the file:

0010111100101010000011010000101000100000001010100010000001000001011100110111001101100101011011010110001001101100011001010111001000101110011010100110000101110110011000010000110100001010001000000001010100000110100001010001000000010101000100000010000110111001001100101011000
010111010001100101011001000010000001101111011011100010000000110001001100100010000011000011
1100100011010001111011011110000100101100001000000011001000110000001100000011010000101100001000000011000000110010001110100011000100110100001000001101010100001101000010100010000000101010001011110000110100001010000011010000101000000110100001010000011010000101000101111001010
1000101010000011010000101000100000001010100000110100001010001000000010101000100000010000000110000101110101011101000110100001101111011100100010000000100000001001010011000010110110101
1001010110110000100000010010110110000101110100011101000110000101101110000011010000101000100000000101010001000000100000000111011001100101011100100111001101101001011011110110111000001101000010100010000000101010001011110000110100001010000011010000101001101001011011010111000001101111011100100111010000100000011010100110000101110110011000010111100000010111001110011011101110110100101101110011001110010111000101010001110110000110100001010011010010110110101110000011011110111001001110100001000000110101001100001011101100110000101111000000101110011100110111011101101001011011100110011100101110011001100110100101101100011001010110001101101000011011110110111101110011011001010111001000101110001010100011101100001101000010100110100101101011100000110111101110010011101000010000001101010011000010111011001100001001011100110

File Name	website_URL.txt
Size	400 byte
Source	Self writing

Snapshot for the file:

000000000110100000000000000111010000000000000111010000000000000111000000000000000111010000000000
101111000000000001011110000000000111011100000000001110111000000000011101110000000000010111000000
000001100011000000000110111100000000001100100000000000011001010000000000111000000000000000011100
100000000001101111000000000011010100000000000011001010000000000110001100000000001110100000000000
001011100000000000110001100000000001101111000000000011011010000000000001011110000000000110001100
000000011100000000000000111000000000000000101111000000000010010000000000000111010100000000000110
011000000000011001100000000000110110100000000001100001000000000110111000000000001011111000000
000110001100000000001101111000000000011001000000000000110100100000000001101110000000000001100111
0000000000101110000000000011000010000000000111001100000000001110000000000000000000000

File Name	design doc.txt
Size	191 KB
Source	Self writing

Snapshot for the file:

0000110100001010010001000110010101110011011010010110011101101110001000000100010001101111011000110111010101101101011001010110111001110100000001101000010100100011101110010011011110111010101110000000100000001101100010000001101001011011100010000000100001101000011001101000011000000110011001000000001010000001100100011000000110000001101100010100100001101000010100100001101101111011101010111001001110011011001010011101000000100101000111011100100110111101110101011100000010000000101000000111001001101111011010100110010101100011011101000010000000101000010000110100001100110100001100000011001100101001000011010000101001000011011011110111010101110010011100110110010100100000001010011011101010111000000110010101110010011101100110100101110011011011110111001000111010000001001010001100110111101110010011001000010110000100000001001010010111101101000011011100000011010000101001010000011100100110111101101010011001010110001101110100001000000010100110111010101110000001100101011100100111011001101001011100110110111101110010001110100000010010100011001101111011100100110010000101100001000000010010100110111110110100001101110000011010000101001010011011110100011101010110010001100101011011100111010001110011001110100000010010100000101111001011011111011001000110010101101010011010010010110000100000001001111101101101011011111011011000110111110110110001110101001000000010000010110011001101111011011000110000101100010011010010011101100001101000010100100001001100001011011000110110001101001001011000010000000101010001110101011001110110001101100101001110110000110100001010010010110110000101110100011101000110000101101110001011000010000000100000010110100001101101011000010110

2. Unicode files

File Name	Arabic_text.txt
Size	3 K
Source	Self writing

Snapshot for the file:

11101111101110111011111111101100010101000110110001011001111011001100001010010000001101100010
1001111110110011000010011011001100001001101100110000111001000000110110001010011111011001100
0100110110001011000111011000101011011101100110000101110110011000011000100000110110001010011
1111011001100001001101100010110001110110001010110111011001100010101101100110000101000011101
0000101011011000101100111101100010111001110110001010011111011000101011111101100010101001001
1000000110110011000010111011000101011111101100110001010110110001011000100100000011011000101
1100111011000101001111110110011000010100100000011011000101001111110110011000010011011000101001
0111011000101011111101100010100111111011000101100011101100010101001001000000110110001010011
11101100110000100110110001011100111011000101001111101100110000101110110001010100100100000011
0110011000010011011001100001001101100010101000110110001011100111011000101010111101100010101
0111110110001010101000100000000101110001000000001000000001000000001000000001000000001000000001000
0000100000000100000000100000000100000000100000000100000000100000000100000000100000000100000000100000
0010000000010000011011000101001111101100110000100110110011000010111011001100010001101100110
0000101101100010110001000011010000101000100000000100000000100000000100000000100000000100000001101
1000101001111110110011000010011011000101100111101100110000100110110001010011111011001100001

File Name	Arabic_news.txt
Size	420 KB
Source	http://www.alarabiya.net

Snapshot for the file:

11101111101110111011111100100000000001101000010101101100010101111110110001010100011011001110
0010100010000000010110100100000011011000101001111101100110000100110110001011100111011000101
1000111011000101010001101100110001010110110001010100100101110110110011000011011011000101010
1000100000000001101000010100000011010000101011011000101100111101100110001010110110011000011
1101100110001000110110011000011000100000110110001010100011011001100010001101100010110011111
0110001011100100100000110110011000010111011000101101001101100010100111110110001011100111011
1001100001000010000001101100010100111110110011000010011011000101101001101100110000101110110
0110001010110110011000010111011000101100011101100110001010001000000110110001010001111011001
1000011000100000011011000101010101101100010110100110110001011100111011000101100010010000011
0110001010100011011000101001111101100110000100110110011000000111011000101011101101100010111
0001001000000110110011000010011011001100000011110110011000100011011001100001101101100110000
1111011000101001110010000001101100010100011110110011000100011011001100001000010000001101100
0101100111101100010111001110110011000100011011000101011111101100110001010110110001010100100
1000001101100010101010110110011000010011011000101010101101100010101101110110011000000100010
0000110110001010100011011000101001111101100110000100110110001011100111011001100001011101100

File Name	Arabic_book.txt
Size	564 KB
Source	http://www.saaid.net/book/8/1634.doc

Snapshot for the file:

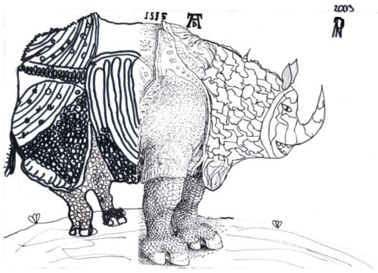
1111111111111111000000110100000000000000101000000000000000110100000000000000101000000000000010011100
0001100100010000000011001000010000000110010010000000001100011001100000011000100000000000000000010
0111000000110010001000000001100011100100000011000110000000001100011000100000011000100111000001
1000100000100000011000000110100000000000000101000000000000000110100000000000000101000000000001000101
0000001100010110100000011001000101000000110010010000000011000101111000000110001000000000000000001
0001010000001100010110100000011001000101000000110001011110000001100010000000000000000001101000000
0110001001110000001100100001100000011000110001000000110000001101000000000000010100000000000000011
0100000000000001010000000000001000000000000000000001101000000000000010100000000000010010100000110
0100010000000110010010010000001100010000000000000000110101000000110010011100000011000101111100
0001100101000000000011001001010000001100100001000000110010011010000001100010000000000000000100
0100000000110001001110000001100010000000000000000101010000000110010011100000011000101000000001
1001010010000000110010001000000011001010000000000110010010010000001100010000000000000000001000101
0000001100100100000000011000101111000000110010100010000001100100111000000110001010100000011001
0011110000001100100011100000011000111010000000000000110100000000000001010000000000001000110000
0110010001010000001100101000100000011000100111000000110001000000000000000010100000000110010011
1000000110001110010000001100101001000000110001011110000001100100111110000001100000110000000110
0010000000000000001000001000000110001001010000001100100011000000110010010100000011000100000000
0000000100010000000011001000101000000110001000000000000000100011000000110010000110000001100100
0110000000110010100100000011000100000000000000000100011000000110001010100000011001001000000001
1001000001000000110010100010000001100100111000000110010100010000001100100111000000110001110 01
0000001100100111100000011001001111000000110001000000000000000100101000000011001001000000011001
0001010000001100010011000000110001100000000011000100000000000000001000110000001100100011000000

3.JPG files

	**File name:** ahmed.jpg
	Size: 5.14 KB
	Dimension: 120 X 150
	Source: self shooting

Snapshot for the file:

111111111110110001111111111110000000000000000001000001001010010001100100100101000110000000000
000001000000010000000100000000101101000000000010110100000000000000000001111111111000010000
010111001100010001010111100001101001011001100000000000000000001001001010010010010101000000
000000100000000000000000000000000000001110000000000001111000000010000001000000000000000110
000000000000000000000000001100010000000000000000000000000000100000000000100000010000000000
010100000000000000000000000000001101000000000000000000000000000000110010000000010000001000000
000000010100000000000000000000000000011111000000000000000000000000000000001001100000010000000
110000000000000000100000000000000000000000000000001000000000000111000000000000000001000100000
000000110000000000000001000000000000000000000000000000000000001000000110000000000000000001000
010000000000110000000000000001000000000000000000000000000000000000001100000111000000010110
100110000111000000100000000000000000100000000000000000000000010010000000000000000000000000000
000000000000000000000000000000000010000110110000101101110011011110110111000000000001000011
011000010110111001101111011011100010000001010000011011110111011101100101011100100101001101
101000011011110111010000100000010000010011011100110000000000000110010001100000011000000011
010100111010001100000011000100111010001100000011001000100000001100010011010000111010001101
000011010100111010001101000011100100000000000111100000000010011010100000100000010100000000
000000010000000000000000000000111111100000001000000000000000100111011000001000000010100
000000000000100000000000000000000000000000110000000100000000000000000000000001001000000000
011100000000000001000000000000000000000000000001100000011001000110010001100000000000111001000
000000001000000000001010000000000000000000000000000111000000010000000000000000000000000100
100100000000001000000000000101000000000000000000000000000010001000000010000000000000000000
000001100100010000011100000000000001000000000000000000000000000000000001000000100000000110000
000000000010100100010000010100000000000000010000000000000000000000000000011011000000010000000
000000000000000001100100100000101000000000000000010000000000000000000000000000111110000000010
000000000000000000000001010010010000001010000000000000001000000000000000000000000000100011000
000010000000000000000000000100100100100000101000000000000000100000000000000000000000000000100
111000000000100000000000000000000000010110010010000001010000000000000000100000000000000000000
000101011000000010000000000000000000000011110010010000000110000000000000000100000000000000000
000000000000101000000000000000000000000000010011001001000000011000000000000000100000000000
000000000000000101100100000000000000000000000000000101010010010000000101000000000000000010000
000000000000000000000101111000000010000000000000000000011111001001001000000111000000000001111

<table>
<tr><td rowspan="4"></td><td>File name: Rhinozeros.jpg</td></tr>
<tr><td>Size: 86.8 KB</td></tr>
<tr><td>Dimension: 650 X 473</td></tr>
<tr><td>Source:http://www.vs-kitzbuehel.tsn.at/projekte/galerie/grafik/Rhinozeros.jpg</td></tr>
</table>

Snapshot for the file:

11111111110110001111111111100000000000000000010000010010100100011001001001010001100000000000
00000100000000100000000100000000100101100000000001001011000000000000000000011111111110110110000
00000100001100000000000000011000000100000000100000000110000001000000010000000110000001100000
11000000110000001000000001100000011000001000000010100001000000001010000010100000100000000100
00000101000010100000001110000011100000110000010000000110000000101000001100000011000000101100
00101000000101100001011000011010000111000010010000100000000110100001110000100010000111000000
1011000010110001000000010110000100000001000100010011000101000001010100010101000101010000011
00000011110001011100011000000101100001010000011000000100100001010000010101000101001111111
11011011000000000100001100000000100000011000001000000010000000101000001000000010100001001001
0001010000001010000100100010100000011010000101100001101000101000001010000010100000101000001
010000010100000101000001010000010100000101000001010000010100000101000001010000010100000101
0000010100000010100000010100000010100000010100000010100000010100000010100000010100000010100
000101000000101000000101000000101000000101000000101000000101000000101000000101000000101000001010000
01010000001010000001010000001010000001010000001010000001010000001010000001010000001010000001
01001111111111100000000000000000001000100001000000000011101100100000010100010100000000110000000
01001000100000000000000001000010001000000001000000110001000100000000111111111110001000000000
00011111000000000000000000000001000001010000000010000000100000001000000010000000100000000100
00000000000000000000000000000000000000000000000000000000000000000100000001000000001100000
0100000000101000000110000000111000010000000100100001010000001011111111111100010000000000101101
01000100000000000000000001000000000100000011000000110000001000000100000000110000010100000101
00000100000000100000000000000000000000000101111101000000001000000100000000110000000000000010000
010001000001010001001000100001001100010100000100000110000100110101000101100001000001110010
001001110001000101000011001010000001100100011010000100001000001000110100001010110001110000
0100010101010100101101000111110000000100100001100110110001001110010100000100000100100001010
000101100001011100011000000110010001101000100101001001100010011100101000001010010010101000
110100001101010011011000110111001110000011100100111010010000110100010001000101010001100100
011101001000010010010100101001010011010101000101010101010110010101110101100001011001010110
100110001101100100011001010110011001100111011010000110100101101010011100110111010001110101
0111011001110111011110000111100101111010100000011100001001000010110000110100001111000100010
001001100010101001001010010011100101001001010110010110100101111001100010011001100110101010
001010100011101001001010010110100110101001111010100010101001101010101011001010110011101101
001011010110110110101101111011100010111001101110101100001011000011110001001100010111000110
110001111100100011001001110010101101001011010011110101001101010111010110110101111101100011
011001110110101110000111100010111000111110010011100101111001101110011111101000111010011110
1010111100011111001011110011111101001111010111110110111101111111100011111001111110101111111
11110001000000000000000111110000000001000000000000000110000000100000001000000010000000100000001

	**File name:** Black White 10.jpg
	Size: 22.7 KB
	Dimension: 600 X 399
	Source: http://www.wmphotos.com/Gallery%20FILES/16-Fine%20Art%20B&W/thumbnails/Black%20White%2010.jpg

Snapshot for the file:

11111111110110001111111111100000000000000000010000010010100100011001001001010001100000000000
00000100000001000000001000000000100100000000000010010000000000000000000011111111111011100000
00000000111001000001011001000110111101100010011001010000000000110010010000000000000000000000
00000000000000000111111111110110110000000001000010000000000000001000000001100000011000000110
00000110000001100000100000000110000001100000100000000110000001000000001110000100000000110000
00111000000101000000100000000100000000101000000111000010000000001101000011010000111000000110100
00
11010001000000001000100001100000011000000110000001100000011000000110000010001000011000000011
00000011000000110000001100000011000000110000001100000011000000110000001100000011000000110000001100
00001100000001100000011000000110000001100000011000000110000001100000011000000110000000110000
00110000000110000001100000011000000000100001001000010000000100000000100100001010000010010000
10110000100100001001000010110000111000001011000011010000101100001110000100010000111000000011
10000001110000011100001000100010001000011000000110000001100000011000000110000010001000010001
00001100000011000000110000001100000011000000110000010001000011000000110000001100000001100000
00110000000110000001100000011000000110000001100000011000000110000001100000011000000110000000
11000000110000001100000011000000110000001100000011000000110000001100000011000000110000000011
00000011001111111111000000000000000000010001000010000000000110001111000000100101100000000011
00000001001000100000000000000000100001000100000000100000011000100010000000111111111101110100
00000000000100000000000001001101111111111000100000000011010001000000000000000000000000000000
01110000000100000001000000001000000001000000001000000000000000000000000000000000000000000000
00000000000000000000000100000000101000000110000001000000110000000010000000000000011100001000
00001001000010100000101100000000100000000000000010000000100000001100000000100000001000000100
00000100000000100000000000000000000000000000000000000000000000000000000000010000000000000
00100000000110000010000000010100000110000000111000010000000010010000101000000101100010000000000
00000000100000000010000001100000011000000100000001000000000100000011000000111000000110000000100
00000010000000110000000100111001100000000100000001000000011000100010000010000000000000000010100
10000100010010001100010100000010101000100000110000100110110000100100010011100011000000010001
01000011001010010001101000010000001110001010110110001010000100010001111000000101010010110100
01111000010011001100010110011000101111000000010010001110010100000010111100010010010101000011
00110100010100111001001010100010101100100110001101110011110000100011010101000100001001111100
01001110100011101100110011011000010111010101000110010001110100110000111101001011100010000
10000001001101000000110000100100001010000110000000110011000010010010100010001010100011010100
1
00101101000101011011010011010101010010100000011010111100101110001111110011110001001101010000
11100100111101000110010101110101100001011001010110100101101101011100010111010101111001011
1
11010101100110011101101000011010010110101001101011011011000110110101101110011011110110001
1
01110100011101010111011001110111011110000111100101111010011110110111110001111101011111100
1
11111101110011100001001000010110000110100001111000100010001001100010101000101110001100100
0

<table>
<tr><td rowspan="4">
</td><td>File name: mar17a6ey.th.jpg</td></tr>
<tr><td>Size: 4.15 KB</td></tr>
<tr><td>Dimension: 152 X 167</td></tr>
<tr><td>Source:
http://img185.imageshack.us/img185/8418/mar17a6ey.th.jpg</td></tr>
</table>

Snapshot for the file:

1111111111011000111111111110000000000000000000100000100101001000110010010010100011000000000000
0000010000000100000001000000000100100000000000001001000000000000000000001111111110110110000
0000010000110000000000000101000000011100000111000010000000011100000110000010100000100000000010
0000001000000010110000101000001010000010110000111000011000000100000000111000001101000011101
0000111000011101000101010001011000010001000110000010001100011111001001010010010000010001000
0111110010001000100001001001100010101100110111001011110010011000101001001101000010100100010
0001001000100011000001000001001100010011010000111001001110110011111000111110001111100010001001
0100101110010001000100100101000011001111000100100000110111001111010011111000111011111111111
1101101100000000010000110000000010000101000000101100001011000011100000110100001110000111000000
0100000001000000011100001110110010100000100010001010000011101100111011001110110011101100110011
1011001110110011101100111011001110110011101100111011001110110011101100111011001110110011101100111
1100111011001110110011101100111011001110110011101100111011001110110011101100111011001110110011101
0011101100111011001110110011101100111011001110110011101100111011001110110011101100111011001110110
1110110011101100111011001110110011101100111011001110110011101100111011001110110011101100110011
1011111111111100000000000000000010001000010000000000001010011100000000010011000000000011000000
0100100010000000000000000100001000100000000100000011000100010000000111111111110001000000000000
0001110000000000000000000000000100000101000000001000000001000000001000000000000000000000000000
000000000000000000000000000000000000000000000000000000000101000000000000000100000000110000
0100000000110000000111000000000100001000111111111110001000000000000011101000010000000000000000000
0100000001100000011000000100000000100000011100000110000001000000011000000001000000001100000000
00000000000000000000000000000001000000000000000100000000110000010000000101000100010001001000
1000010000001100011000100010011000101000010001001000000101010001011000011001000100100011001 1
0010010000100111000101110010101100010001010101000011010100111001001000000111010100100110000
1010000000110000010101000010001011000100100001100110110001111000000111111111110001000000000000
0001101000000000100000000000000010000000110000000100000000100000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000110000010000000000000000000100000
0101000000001000001101111111111100010000000000000101111000100010000000000000000100000000100000000
0100000100000000000000000100000100000000011000000011000000000000000000000000000000000000000000
0000000000000000000000010000000100000000110001000100000100000100100010000100110001010000000101
1000010000001010001001100010100001000100010001100110010010100011000000010011001110010001101 0
0001101100011101000111111000000111000111110000011111000111111111110110100000000000000001100000000
1100000001000000000000000100001000100000001100010001000000000001111110000000011100111000010001
1100011101001101110101001110000100100110100111100010001011100011000110110111001001110100000
0011001001001110000101010110100101000100000000111001010000110001100011111111000100001001 10

File name: fractal.jpg	
Size: 683 KB	
Dimension: 800 X 600	
Source: http://www.nonlinearthinking.com/images/fractal.jpg	

Snapshot for the file:

11111111110110001111111111100000000000000000010000010010100100011001001001010001100000000000
00000100000010000000010000000001001000000000000100100000000000000000001111111111000010010
01110000010001000101011110000110100101100110000000000000000001001101010011010000000001010
10000000000000000000000000000100000000000000001110000000100010010000000000000001100000000
00000000000000000000001000000000000000100000000000000000000000100011010000000000000010100
00000000000000000000000000001000000000000000000000000011000100000000100011011000000000000
01010000000000000000000000000000001000000000000000000000000110101000000001001010000000000
00000000110000000000000000000000000000010000000000000001000000000000000000000000100110001
00000000000000100000000000000000000000000000101000000000000000000000000000111001000000000100
11001000000000000000010000000000000000000000000000000101000000000000000000000000000100001101000
01110110100100000000000000100000000000000000000000000000000100000000000000000000000000100111
00000000000000000000000000110010000000000000000000000000000100100000000000000000000000000
00000001000000000000000000000000001001000000000000000000000000000000000101000000101100100001
10111101100010011001010010000000101000001101000011011110111010001101111011100110110100000110
11110111000000100000000110111001011100011000000000000000110010001100000011000000110100001110
10001100000011011000111010001100100011100000100000000110001001101010011101000110100000110001
00111010001101010011100100000000000000000000000000000000000001110100000000000001000000000
00001100000000000000000000000000000000111111111111111110000000000000000010100000000000100000
00000000010000000000000000000000000000000010000000000000000000000011001000000101000000000000
11000000000000010000000000000000000000000000000010000000000000000000000100101100000000000
00000000000000000000000000000000000000000000000110000000010000001100000000000000001100
00000000000000000000000000001000000000000011000000000000000000000000100011010000000000000
01010000000000000000000000000000001000000000000000000000000100010110000000010001101100000
00000001010000000000000000000000000000001000000000000000000000001000111100000000100101000
00000000000000110000000000000000000000000000000100000000000000010000000000000000000000001000
00000100000000000000100000000000000000000000000000000010000000000000000000000001001001100000
00100000000100000000000000010000000000000000000000000000000001000000000000000000000100101110101
10000000000000000000000000000000000000000000000000000001001000000000000000000000000000
00000001000000000000000000000000010010000000000000000000000000000000000001111111111101100011
11111111100000000000000000001000001001010010001100100100101000110000000000000000100000000100000
00010000000000100100000000000010010000000000000000000011111111110110100000000000001100010000
0101100100011011110110001001100101010111110100001101001101000000000000000011111111111101110
00000000000011100100000010110010001101111011000100110010100000000011001001000000000000000000
0000000000000000000011111111110110110000000001000010000000000000011000000100000000100000000
10000000100100001000000001100000010010000100100001100000100010000101100001010000010110001000
0100010101000011110000110000001100000011110001010100011000000100110001001100010101000100011
00010011000110000000100010000110000001100000011000000110000001100000011000000100010000110000
00110000000110000000110000000110000000110000000110000000110000000110000000110000000110000000
11000000011000000011000000011000000011000000011000000011000000011000000011000000011000000011

Bibliography

1. A Listair Moffat, A.T., *Compression and coding algorithms*. 2002: Kluwer Academic Publisher. Pages 1-6.

2. Mandal, J., *An Approach towards Development of Efficient Data Compression Algorithms and Correction Techniques*. PhD. 2000, Jadavpur University, India.

3. Sebastian, D., *Universal losselss data compression algorithmes*. PhD. 2003, Silesian University.

4. Huffman, D.H. *A Method for the Construction of Minimum-Redundancy Codes*. in *IRE*. September 1952.

5. Ziv, J.a.L., A. *A Compression of individual sequences via variable-rate coding*. in *IEEE Transaction on Information Theory*.

6. Ziviani, N., Moura, E., Navarro, G., and Baeza-Yates, R. *Compression: A Key for Next-Generation Text Retrieval Systems*. in *IEEE Computer*. Novmber 2000.

7. Brisaboa, N., Iglesias, E., Navarro, G., and Paramá, J. *An Efficient Compression Code for Text databases*. in *25th European Conference on IR Research, ECIR* Apirl 2003. Pisa, Italy.

8. Wayne, N.J., *ARC File Archive Utility*. 1986, Enhancement Associates.

9. Glendale, W., *PKARC FAST! File Archival Utility*, PKWARE, Inc.

10. Adler, M.a.M., M. . *Towards Compressing Web Graphs*. in *IEEE Data Compression Conference*. 2001. Utah, USA.

11. Wu, D., Hou, Y., Zhu, W., Zhang, Y., and Peha, J. *Streaming Video over the Internet: Approaches and Directions*. in *IEEE Transactions on Circuits and Systems for Video Technology*. March 2001.

12. Tanenbaum, A.S., *Computer Networks*. 2002: Prentice Hall PTR. Pages 724-736.

13. KERCKHOFF, A. *La Cryptographie Militaire*. in *J. des Sciences Militaries*. 1883.

14. *Cryptography FAQ (03/10: Basic Cryptology), "3.5. What are some properties satisfied by every strong cryptosystem?"* [cited 23 August 2006]; Available from: http://www.faqs.org/faqs/cryptography-faq/part03/index.html.

15. A. Hauter, M.V.C., R. Ramanathan. *Compression and Encryption. CSI 801 Project Fall 1995*. December 7, 1995 [cited 10 March2006]; Available from: http://www.science.gmu.edu/~mchacko/csi801/proj-ckv.html.

16. *ACT- Archiver Index, WinZip 7.0 SR-1*. [cited 31 August. 2006]; Available from: http://compression.ca/act/act-index.html.

17. *How does cryptography work*. 12 March 2006 [cited 12 March 2006]; Available from: http://www.pgpi.org/doc/pgpintro.

18. Mark Nelson, J.-L.G., *The Data compression book*. 1996: M&T Books. Pages 12-20.

19. Wagner, N.R. *The Laws of cryptography with java code chapter 9* [cited 12 August 2006.]; Available from: http://www.cs.utsa.edu/~wangner/lawbookcolor/laws.pdf.

20. Barker .E, B.W., Burr .W, Polk .W, Smid .M., *Recommendation for key management- Part 1*, **NIST.** National Institute of Standards and Technology, Editor. May 2006.

21. Augustine, J., Feng, W., and Jacob, J. *Logic Minimization Based Approach for Compressing Image Data*. in *Conf. VLSI Design*. 1995. New Delhi, India.

22. Agauan, S., Baran, T. and Panetta, K. *Transform-Based Image Compression by Noise Reduction and Spatial Modification Using Boolean Minimization*. in *IEEE workshop, Statistical Signal Processing*. 2003. USA.

23. *Introduction to Cryptography page 11*. 1999 [cited 12 August 2006]; Available from: http://ivan.tubert.org/doc/introtoCrypt.pdf.

24. E, W.D.a.M. *New Directions in Cryptography*. in *IEEE Transactions on Information Theory*. Novmber 1976.

25. Lonardi, S., *Off-line data compression by textual substitution*, in *Computer Sciences*. 31 December 1998, Msc. UNIVERSITA DEGLI STUDI DI PADOVA.

26. *Adaptive Huffman Coding*. Adaptive Huffman coding [cited 14 August 2006]; Available from: http://www.cs.cf.ac.uk/Dave/Multimedia/node212.html.

27. Boole, G., *An Investigation of the laws of thought*. 1954: New York:. Dover.

28. M. Morris Mano, C.R.K., *Logic and computer design fundamentals*. 2004: Prentice Hall. 33-40.

29. Ercegovac, M.D., Lang, T., Moreno, J, *Introduction to Digital Systems*. 1999: John Wiley and Sons Inc.

30. Mano, M.M., *Digital Design, third edition*. 2002: Prentice Hall. Pages 71-73.

31. wang, C.-E., *Simultaneous Data Compression and Encryption CA 95819-6021.*, University, Sacramento: California State.

32. Stinson, D.R., *Cryptography theory and practice*. 2006: Chapman & Hall/CRC. Page 393.

33. Bell, R.A.a.T.C. *A corpus for the evaluation of lossless compression algorithms*. in *IEEE Data Compression Conference (DCC'97)*. March 25 1997. Los Alamitos, California.: IEEE Computer Society.

34. T. C. Bell, J.G.C., and I. H.Witten, *Text Compression*. 1990, Englewood Cliffs: Prentice Hall.

35. *Large Canterbury corpus. "The Large Corpus"*. [cited 25 August 2006]; Available from: http://corpus.canterbury.ac.nz/descriptions/.

36. Press W., T.S., Vetterling W., Flannery B., *Numerical Recipes in C. Second Edition.* 1995: Reprinted with correction, Cambridge University Press.

37. *ACT- Archiver Index, RAR 3.00b5*. [cited 31 August. 2006]; Available from: http://compression.ca/act/act-index.html.

38. ElQawasmeh, E. and Kattan, A. *REVERSING K-MAP USAGE FOR DEVELOPING A NEW COMPRESSION ALGORITHM*. in *ACIT,The International Arab Conference on Information Technology* 2004. Algeria

39. Kattan, A." *Data Compression with built in encryption*". CC401 project proposal, March 2006.