Thesis No: CSER-M-19-09

An Efficient Compression Scheme for Large Natural Language Text

by

Md. Ashiq Mahmood

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Khulna 9203, Bangladesh

September 2019

An Efficient Compression Scheme for Large Natural Language Text

by

Md. Ashiq Mahmood
Roll No: 1707507

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

Master of Science in Computer Science & Engineering

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Khulna 9203, Bangladesh

September 2019

Declaration

This is to certify that the thesis work entitled "An Efficient Compression Scheme for Large Natural Language Text" has been carried out by Md. Ashiq Mahmood in the Department of Computer Science and Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh. The above thesis work or any part of this work has not been submitted anywhere for the award of any degree or diploma.

[illegible]

Signature of Supervisor

[illegible]

Signature of Candidate

Approval

This is to certify that the thesis work submitted by Md. Ashiq Mahmood entitled " An Efficient Compression Scheme for Large Natural Language Text" has been approved by the board of examiners for the partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering in the Department of Computer Science and Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh in September, 2019.

BOARD OF EXAMINERS

Acknowledgment

All the praise to the almighty Allah, whose blessing and mercy succeeded me to complete this thesis work fairly. I gratefully acknowledge the valuable suggestions, advice and sincere co-operation of Dr. K. M. Azharul Hasan, Professor, Department of Computer Science and Engineering, Khulna University of Engineering & Technology, under whose supervision this work was carried out. His open-minded way of thinking, encouragement and trust makes me feel confident to go through different research ideas. From him, I have learned that scientific endeavor means much more than conceiving nice algorithms and to have a much broader view at problems from different perspectives. I would like to convey my heartily ovation to all the faculty members, officials and staffs of the Department of Computer Science and Engineering as they have always extended their co-operation to complete this work. I am extremely indebted to the members of my examination committee for their constructive comments on this manuscript. Last but not the least, I wish to thank my friends and my family for their constant support.

Author

Abstract

Data compression is the route towards adjusting, encoding or changing the bit structure of information so that it requires less space. Data compression is a decrease in the quantity of bits expected to demonstrate the data. Compacting data can spare stockpiling limit, accelerate record exchange, and lessening costs for capacity equipment and system transfer speed. Data compression covers a huge space of jobs including data correspondence, data putting away and database improvement. In the same way, Text compression can be as straightforward as expelling every unneeded character, embedding a solitary recurrent character to demonstrate a string of rehashed characters and substituting a little piece string for a habitually happening bit string. The fundamental standard behind compression is to build up a strategy or convention for utilizing less bits to express the actual data. Character encoding is fairly identified with data compression which represents a character by a type of encoding system. In this thesis, an efficient and simple compression algorithm for large natural text named *n-Sequence* based *m* Bit Compression (*nSmBC*) is proposed which can able to beat WinZip and WinRAR in terms of compression ratio. WinZip and WinRAR are two well-known compression techniques used for text compression in the industry. The scheme provides an efficient encoding algorithm that converts an 8 bit character by 5 bits utilizing a look up table. The look up table is produced by using Zipf's distribution which is a discrete distribution of commonly used characters in different languages. 8 bit characters are converted to 5 bits by partitioning the characters into 7 sets. After converting the characters into 5 bit, an *n-sequence* scheme is developed to logically calculate the location number of a particular combination of characters. The reverse algorithm to recover the actual input is further demonstrated. The algorithm is finally compared with the well-known WinZip, WinRAR, Huffman and LZW techniques. Promising performance is demonstrated both by theoretical and experimental analysis.

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABREVIATIONS

LZW	Lempel-Ziv-Welch
nSmBC	*n-Sequence* based *m* Bit Compression Scheme
6BC	6 Bit Compression Technique

CHAPTER I

Introduction

1.1 Introduction

Data compression defines a method to represent data or can be called to encode data using shorter form of bit [1]. The primary purpose of compaction comprises decreasing extra room needed storing data, diminishing transfer speed necessity so as to transmit it, along these lines lessening all out expense [2]. Despite the fact that a vast extra room is accessible for putting away data however it might cross the limit of transmission. Techniques for compressing data is characterized into two classifications [3]. Lossless technique and Lossy technique for compression are the 2 classification of data compression. Lossless compression technique is more often abuse factual repetition so that data of sender's can be illustrated briefly. Lossless technique for compression is conceivable on the grounds that a large portion of this present reality data has factual repetition. In terms of lossless compression system, data misfortune is inadmissible. Actual content must remade through the packed content [4]. The second compression procedure, called lossy technique for data compression might conceivable if a small data misfortune is allowed. For this situation, actual content might not be remade from the packed data because of evacuation of some excess data during compressing. Compression of data may be utilized progressively proficient accumulating hybridization with various techniques [5]. Fundamental favorable position of this kind of method is actually it may pack yield document that might delivered in the wake of utilizing some compression systems. This demonstrates a superior outcome. Zipf’s dispersion is utilized in this paper to build the Look up table of characters. It originated from Zipf's law [6]. Zipf's law can be said to a test law detailed using numerical insights that insinuates the manner in which that various sorts of data considered in the physical and sociologies can be approximated with a Zipf’s conveyance. Zipf's law communicates with the given corpus comprising natural language articulations, recurrence of a certain word might be then again comparing against its situation by the help of table of recurrence. Thusly the most nonstop word will happen generally twofold as typically as the second most progressive word, on numerous occasions as routinely as the third most unending word.

1.2 Problem Statement

Numerous strategies have been proposed in the literature for compressing large natural text data. These strategies can be additionally grouped into four noteworthy sorts, that is, substitution, statistical, dictionary, and context-based method [7]. The substitution data compression procedures supplant a specific longer reiteration of characters with a shorter one. A system that is a delegate of these strategies is run-length encoding [8]. The statistical methods more often than not compute the likelihood of characters to create the briefest normal code length, for example, Shannon-Fano coding [9, 10], Huffman coding [11] and arithmetic coding [12,13]. The following kind comprises of dictionary based strategies, which include substitution of a substring of content by a file or a pointer code. They identify with a situation in the word reference of the substring. Delegates of these methods are LZW [14], LZ77 [15], and LZ78 [16]. The last kind is context based systems, which include the utilization of negligible earlier presumptions about the measurements of the content. Typically, they utilize the setting of the content being encoded and the historical backdrop of the content to give increasingly proficient compression. Delegates of this sort are Prediction by Partial Matching (PPM) [17] and Burrow– Wheeler change (BWT) [18].

Most of the techniques mentioned above uses a dictionary in the physical memory. So a huge amount of space in memory is needed to store this dictionary. Another fact is that all the techniques available in recent times applies only this 3 techniques including Run length encoding or Huffman encoding or LZW technique.

In this thesis, we introduced a new idea namely *n-sequence* based *m* bit compression Scheme. In this technique, the dictionary is logically implemented based on a hash function. Because of the logical implementation of the dictionary, it does not take any physical space in memory. So a huge space is saved in the physical memory.

1.3 Objectives

The main objectives of this thesis are to–

- Develop an efficient encoding algorithm which can work with the natural characters by converting 8 bit character to 5 bit.
- Construct a powerful compression algorithm which can compress the large natural dataset to a promising efficiency.
- Propose a decompression algorithm which can decompress the compressed dataset to the original dataset without any loss of the original dataset.
- Compare the efficiency of proposed technique with traditional and some industrial techniques to justify the effectiveness of the technique.

1.4 Scope

The important scopes under this thesis are as follows:

- Data can be compressed by using logical calculation without using any third party software or database.
- Standard dataset is used to compress and uncompress.
- NetBeans platform is used for developing Java desktop applications.
- Java programming language is used to implement the prototype system.

1.5 Contribution

The contribution of this thesis can be summarized as follows:

- A powerful compression technique will be available which can compress any amount of text by a promising efficiency rate.
- Theoretical analysis is verified with experimental results.
- Providing details of theoretical and experimental analysis for Compression of file Size, Compression Ratio, with operations on stored data.

1.6 Organization of the Thesis

- **Chapter I** presents the introductory part of the thesis which includes the introduction, problem statement, objectives, scope and contribution of the thesis.
- **Chapter II** presents Literature Review that describes some of the traditional and prominent Data Compression scheme that are already exists. Some of these data compression methods will be described.
- **Chapter III** proposes a new scheme for data compression called “An Efficient Compression Scheme for Natural Language Text”. It also provides an example to demonstrate how the algorithm actually works.
- **Chapter IV** illustrates the performance analysis by deriving a theory to find the efficiency rate. As well as the experimental results of proposed scheme and its evaluation are discussed which shows the technical soundness of the technique.
- The future direction of work on the proposed model and the conclusive words about the model are outlined in **Chapter V.**

CHAPTER II

Literature Review

2.1 Introduction

Data compression is useful in the modern computing world and it is generally utilized by numerous applications [28]. The fundamental standards of data compression are embarked to accomplish a decrease in document measure by encoding data more proficiently. The vast majority of the data compression procedures are lossless [29]. This implies the compacted document will be reestablished precisely to its unique state with no loss of data amid the decompression procedure. The significance of this is central as the record would be ruined and unusable should data be lost. Lossless compression algorithms use measurement displaying methods to decrease redundant data in a document [30]. A portion of the strategies may incorporate expulsion of dividing characters, speaking to a string of rehashed characters with a solitary character or supplanting repeating characters with littler piece arrangements. Another compression classification which is frequently utilized in interactive media records for music and pictures (for example JPEG documents) and where data is disposed of is alluded to as "lossy" compression [31]. In this class of data encoding techniques, inaccurate approximations (or fractional data disposing of) are utilized to speak to the substance. These strategies are essentially used to lessen data estimate for capacity, dealing with and transmitting content. At the point when there are a substantial number of documents included, compression can be a scientifically extreme and tedious procedure [32].. Another important concept in this thesis is Zipf’s distribution by which the dictionary was made. So Zipf distribution is also discussed here.

2.2 Types of Data Compression

Compression of data is significant to this modern era in view of measuring of the data which is exchanged inside a specific system. It constructs the exchange within data generally simple. This part clarifies and distinguishes lossless and lossy technique of compression.

2.2.1 Lossless Data Compression

A Lossless technique [33] for compression is a type of compression which can have the ability to reconstruct the original data perfectly from the compact data. This might be separated to lossy technique, which may not empower unmistakable special data changing from compacted form data. It is utilized in various applications. This type of compaction is utilized at the time of the fact that first and decoded data must be undefined.

2.2.2 Lossy Data Compression

A lossy technique [34] for compression framework is where packing and unpacking recoups data might be not exactly equivalent to the first, anyway is "close enough" to be useful all over. There exists two primary lossy pressure plans:

First one might be the lossy change codecs. Trial of image and noise will be taken, sliced within little pieces, changed within another reason quantized. The other one might be lossy perceptive codecs. Past or possibly coming about decompressed data might be used to foresee the present noise precedent or image layout.

2.2.3 Substitutional Data Compression

Substitutional Data Compression [35] improves the letters in order with super images, permitting encoding various events of individual images. This technique is important in circumstances where one image is locally ruling over all the others. It is as yet used to pack exceptional sorts of data (PC created designs for example), or as an extra handling venture in increasingly complex compression techniques. A substitutional strategy for the easiest sort is run-length encoding (RLE) [36].

2.2.4 Statistical Data Compression:

Lossless statistical data compression [37] calculations have produced a ton of enthusiasm in the course of the most recent fifteen years. Such calculations are regularly consecutive: they process a data stream from start to finish, gradually developing and refining a model dependent on data that has been handled, without the need to get to images further upstream than the first unencoded one. Developing a blower along these lines has various significant points of interest. Initially, all data that the blower uses is accessible to the decoder also, so there is no compelling reason to yield additional data for the decoder to have the option to discover how the data are encoded. Furthermore, the size of the record that should be packed shouldn't be known ahead of time and might be self-assertively enormous, as one may envision to be the situation for a data channel between two PCs on a system (the compression serving to improve transmission capacity). A third reason is that it extraordinarily disentangles the procedure thoughtfully: images from the data source are in every case either encoded or not encoded, there will never be any relationship between's the yield and up 'til now unencoded input images. Shannon-Fano coding [38], Huffman coding [39] and arithmetic coding are the examples of this method.

2.2.5 Dictionary-based Data Compression:

In dictionary compression [40], variable length substrings are replaced by short, possibly even fixed length codewords. Compression is achieved by replacing long strings with shorter codewords.

The general scheme is as follows:

• The dictionary D is a collection of strings, often called phrases. For completeness, the dictionary includes all single symbols.

• The text T is parsed into a sequence of phrases:

$$T = T1T2 \ldots Tz, Ti \in D.$$

The sequence is called a parsing or a factorization of T with respect to D.

• The text is encoded by replacing each phrase *Ti* with a code that acts as a pointer to the dictionary.

A dictionary-based method of the simplest sort is LZW [41].

2.2.6 Context-based Data Compression:

Context-based arithmetic coding [42] is a universal compression technique usually applied to encode multimedia content in combination with other compression methods in order to achieve high compression ratios. It is performed in two separate phases. The first phase aims to efficiently estimate the source statistics. The second phase utilizes arithmetic coding to represent the symbols with high probability of the occurrence with fewer bits than the symbols with low probability of occurrence. Example of this sort is Prediction by Partial Matching (PPM) [43].

2.3 Well- known Data Compression Techniques

This section explains the basic principles of some data compression techniques
The data compression techniques actually available in modern scenario are:

1. Run Length Encoding (RLE)
2. Huffman Coding
3. LZW Compression
4. Arithmetic Coding.

2.3.1 Run Length Encoding (RLE)

Run-length encoding [44] is a data pressure algorithm that is reinforced by most bitmap archive positions, for instance, TIFF, BMP, and PCX. RLE is fitting for compacting any sort of data paying little regard to its data content, anyway the substance of the data will impact the pressure extent achieved by RLE.
RLE works by diminishing the physical size of a proceeding with arrangement of characters. This keeping string, called a run, is generally encoded into two bytes. The vital byte addresses the amount of characters in the run and is known as the run count. Before long, an encoded run may contain 1 to 128 or 256 characters; the run consider generally contains the amount of characters short one. The second byte is the estimation of the character in the run, which is in the extent of 0 to 255, and is known as the run regard.

AAAAAAAAAAAAAAA

Uncompressed, a character continue running of 15 A characters would usually require 15 bytes to store:

15A

A comparative string after RLE encoding would require only two bytes.
The 15A code delivered to address the character string is known as a RLE group. Here, the primary byte, 15, is the run check and contains the amount of redundancies. The second byte, An, is the run regard and contains the authentic reiterated a motivating force in the run.

Another bundle is made each time the run character changes, or each time the amount of characters in the run outperforms the most outrageous check. Expect that our 15-character string by and by contains four differing character runs:

AAAAAAbbbXXXXXt

Utilizing run-length encoding this could be compacted into four 2-byte bundles:

6A3b5X1t

Thus, after run-length encoding, the 15-byte string would require only eight bytes of data to address the string, as opposed to the initial 15 bytes. For this circumstance, run-length encoding yielded a pressure extent of practically 2 to 1.

2.3.2 Huffman Coding

This Huffman coding is an entropy encoding procedure used for lossless data pressure. It was made by David A. Huffman while he was a Ph.D. understudy at MIT, and circulated in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes" [45]. The methodology in every practical sense starts with the leaf centers containing the probabilities of the picture they address, and after that another center whose youths are the 2 center points with most diminutive probability is made, to such a degree, that the new center's probability is comparable to the total of the children's probability. With the 2 centers solidified into one center point (consequently not considering them any more), and with the new center being presently considered, the strategy is reiterated until only a solitary center remains, the Huffman tree.

1. Scan text to be compacted and count event all things considered.
2. Sort or organize characters dependent on number of events in text.
3. Build Huffman code tree dependent on organized rundown.
4. Perform a traversal of tree to decide all code words.
5. Scan text again and make new document utilizing the Huffman codes.

Figure 2.1 Algorithm for Huffman

Considering the following short text:

Eerie eyes seen near lake.

Step 1: Count up the occurrences of all characters in the text.

Step 2: Find which characters are present: **E e r i space y s n a r l k.**

TABLE 2. 1 Character Frequency

Character	Frequency
E	9
Space	4
R	2
S	2
N	2
A	2
E	1
I	1
Y	1
L	1
K	1
.	1

Step 3: Build a tree by this steps

- Single node is dequeued in the left of the queue.
- New code words for each character is contained in the tree.
- Number of characters in text must be equal to the Frequency of root node

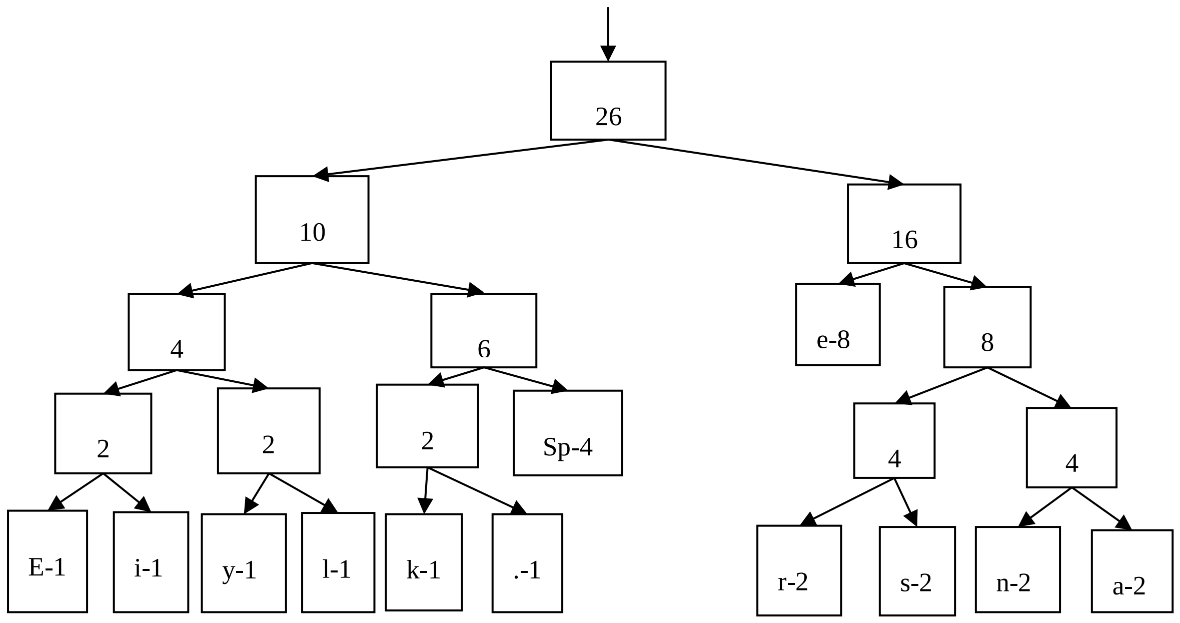


Figure 2.2 Huffman Tree

Encoding:

- Traverse the tree to obtain new word code
- Moving on to left is a 0 or right is a 1
- When a leaf node is reached code word is completed then.

In the example, the below result will be found:

TABLE 2. 2 Code of Huffman

Character	Code
E	10
Space	011
R	1100
S	1101
N	1110
A	1111
E	0000
I	0001
Y	0010
L	0011
K	0100
.	0101

Let us suppose for word “Eye” code will be

E= 0000, y= 0010, e= 10

So the encoded string is 0000001010

2.3.3 Lempel-Ziv-Welch (LZW) Compression

Lempel-Ziv-Welch (LZW) is a data pressure algorithm made by Abraham Lempel, Jacob Ziv, and Terry Welch [46]. The algorithm is expected to rush to execute anyway isn't commonly perfect since it performs simply compelled examination of the data. LZW can in like manner be known as a substitutional or lexicon based encoding algorithm. The algorithm routinely creates a data word reference (similarly called an understanding table or string table) of data occurring in an uncompressed data stream. Instances of data (substrings) are perceived in the data stream and are composed to sections in the word reference.

In case the substring is missing in the word reference, a code articulation is made subject to the data substance of the substring, and it is secured in the lexicon. The articulation is then stayed in contact with the pressed yield stream. Exactly when a reoccurrence of a substring is found in the data, the outflow of the substring recently secured in the lexicon is stayed in contact with the yield. Since the articulation regard has a physical size that is tinier than the substring it addresses, data pressure is practiced.

```
Input: BABAABAAA
Output: 66 65 256 257 65 260
1   Start table with character strings which are single
2   M = 1st character
3   Unless finishing the stream of data
4   N = Next character
5   IF M + N is found in the string table
6   M = M + N
7   Else
8   yield M’s Code
9   Adding M+ N to the string table
10  M = N
11  Finish while
12  yield M’s code
```

Figure 2.3 Algorithm for LZW

Example:

Using the LZW algorithm to compress the string

BABAABAAA

TABLE 2. 3 How LZW compress string

ENCODER	OUTPUT	STRING	TABLE
Output code	representing	codeword	string
66	B	256	BA
65	A	257	AB
256	BA	258	BAA
257	AB	259	ABA
65	A	260	AA
260	AA		

2.3.4 Arithmetic Encoding

Arithmetic coding [47] is a lossless coding procedure which does not encounter the evil impacts of the recently referenced hindrances and which will when all is said in done achieve a higher compression extent than Huffman coding. Arithmetic coding is a run of the mill calculation used in both lossless and lossy information compression calculations [48]. It is an entropy encoding methodology, in which the frequently watched pictures are encoded with less bits than lesser seen pictures. It has a couple of focal points over got frameworks, for instance, Huffman coding.

Encoding with Floating-Point Math

The term arithmetic coding covers two separate systems: encoding messages and deciphering them. The hypothetical idea of an arithmetic coding model is that each picture will have its very own exceptional segment of the number line of real numbers some place in the scope of 0 and 1.

For example, it might be started with an encoder that can encode only a letters arranged by 100 particular characters. In an essential static model, resulting to starting with capital letters, by then move to the lower case letters. This infers the chief picture, 'A', will guarantee the number line from 0 to .01, 'B' will have .01 to .02, and so on. With this model, the encoder can address the single letter 'B' by yielding a floating point number that is under .02 and more important than or proportional to .01. So for example, an arithmetic encoder that expected to make that single letter could yield .15 and be done.

To encode a progression of pictures incorporates a hardly progressively confounded system. For the character 'B', that suggests the message is some place in the scope of .01 and .02. The accompanying character in the message by then further parcels that present range proportionate to its present duty regarding number line. So some other letter that claims the completion of the number line, from .99 to 1.0, would change the range from [.01,.02) to [.0199, .020). After the entire message has been readied, it has the last range, [low,high). The encoder yields a skimming point number right in the point of convergence of that run.

Decoding With Floating-Point Math

The math in the decoder on a very basic level pivots the math from the encode side. To decipher a character, the probability model essentially needs to find the character whose broaden covers the present estimation of the message. Exactly when the decoder initially starts up with the model estimation of 0.22232425, the model sees that the regard falls between the break asserted by 'W': [0.22,0.23); so the model returns W.

2.4 Industrial Schemes

Actually two kind of data compression techniques named WinZip and WinRAR are used for industrial purposes. They are the most popular and widely used file formats to archive compressed data. They are undoubtedly the undisputed kings of compressed files. While both use the superfast LZ77 compression algorithm to compress and decompress content [47], surely one has a little edge over the other, may be in terms of speed and efficiency. Both are compression algorithms that efficiently compress the files to reduce their size without affecting the content of the files. While a WinRAR file is an archival file created with the WinRAR program, Zip file is a common file extension associated with several programs such as WinZip, WinRAR, and Freebyte Zip.

2.4.1 WinZip Compression

The WinZip file format was actually created by Phil Katz and Gary Conway following a lawsuit against PKWARE filed by System Enhancement Associates (SEA). The lawsuit claimed that the archiving products of PKWARE were taken from the SEA’s proprietary ARC archiving system. However, the lawsuit was dropped followed by a legal settlement with the SEA. Katz released his first compression program to use the new WinZip file format called PKZIP and subsequently released it into the public domain in 1989. Today, Zip is a widely used format for lossless data compression and is supported by several software utilities including the built-in WinZip support provided by Microsoft Windows and Mac OS X. The best part, WinZip files can be opened with any program that creates WinZip files.

Like other archive formats, WinZip files are data containers that contain one or more files together in a compressed or zipped format using WinZip compression. Well, WinZip archives are capable of more than just compressing files; they can encrypt files (password protected) and split archives with just a few clicks. Multiple files can be compressed or zipped using several methods such as LZMA, WavPack, PPMd, BZIP2, DEFLATE, etc. Each file can be stored separately, so that they can be accessed randomly and because they are archived individually, it makes it easy to extract them, or add new ones without even zipping in the entire archive. WinZip archives can also contain additional content which are not related to the archive thereby making it a self-extracting archive.

2.4.2 WinRAR Compression

WinRAR stands for Roshal Archive Compressed file, which is a proprietary archive file format named after its Russian-origin creator Eugene Roshal. Like other archives, WinRAR contains one or more files or folders together. Think of WinRAR as a folder just like a normal folder containing several programs or files, however, unlike a normal folder on your hard drive, WinRAR files require third-party software to open and extract the contents of the archive. It’s a native file format of WinRAR archiver which stores multiple files in the compressed form only needed to do is unpack its contents to access the files. It uses a higher compression ratio than regular WinZip compression and incorporates a proprietary compression algorithm that handles lossless data compression, file spanning, error recovery, and more. The archives files normally have the standard “.WinRAR” file extension.

2.4.3 A comparison between WinZip and WinRAR

Basics of WinZip and WinRAR

WinZip is an archive file format created by Phil Katz as a standard format for lossless data compression which incorporates several compression algorithms to compress/decompress one or more files. WinRAR is a proprietary archive file format developed by a Russian software engineer Eugene Roshal.

Efficiency of WinZip and WinRAR

RAR format can compress a file much better than the same when done with WinZip format, meaning the rate of compression of RAR is better than that of the WinZip format. Also WinRAR archived smaller sizes as compared to WinZip archives, which makes WinRAR a better alternative than WinZip.

Popularity of WinZip and WinRAR

The main advantage of using a WinZip format is its popularity. As WinZip file format was developed a long time ago, it has a little edge over the WinRAR format and is still the most widely used archive type, which still accounts for a significant number of archive files on the internet.

Proprietary Software for WinZip and WinRAR

A third-party software program called WinRAR is required to open and extract the contents of the WinRAR archive, whereas WinZip is a widely used format supported by various commercial as well as open source tools, and libraries.

Compression Speed in WinZip and WinRAR

WinZip uses a less complex structured format to store files. It uses the older yet popular DEFLATE compression algorithm to compress data which is less efficient than the newer compression methods which are not supported by any operating system by default. WinRAR uses a compression algorithm which is substantially better and efficient than the DEFLATE compression method.

Security in WinZip and WinRAR

WinRAR uses a proprietary program called WinRAR archiver to compress/decompress contents of a file that comes with built-in support for password encryption which is great for security. However, the default support in Windows and Macintosh operating systems does not have password protection feature.

2.5 Zipf’s Distribution

Zipf's law is an experimental law planned utilizing scientific measurements. The law is named after the language specialist George Kingsley Zipf, who initially proposed it [6].

Zipf's law expresses that given a vast example of words utilized, the recurrence of any word is conversely corresponding to its position in the recurrence table. So word number *n* has a recurrence corresponding to 1/n [27].

Accordingly the most regular word will happen about twice as frequently as the second most continuous word, multiple times as frequently as the third most successive word, and so forth. For instance, in one example of words in the English language, the most much of the time happening word, "the", represents almost 7% of the considerable number of words (69,971 out of marginally more than 1 million). Consistent with Zipf's Law, the second-place word "of" represents somewhat over 3.5% of words (36,411 events), trailed by "and" (28,852). Just around 135 words are expected to represent a large portion of the example of words in a vast sample.

A similar relationship happens in numerous different rankings, disconnected to language, for example, the populace positions of urban communities in different nations, enterprise sizes, pay rankings, and so on. The presence of the appropriation in rankings of urban areas by populace was first seen by Felix Auerbach in 1913.

The Most Common Words in English

Zipf’s law is a curious relation that connects distributions of words and populations of cities to inverse relations. The American linguist George Kingsley Zipf noticed it when looking at the relative frequencies of words in a large text, like the book Moby Dick.

Here are the most frequent words in the English language, along with the rough percentages of how often that word occurs in written texts. For example, the most common word, ‘the’, appears roughly 6.8% of the time. Of the 92 words in the two paragraphs of the book, he counted 9 uses of the word ‘the’. That is therefore somewhat above average.

TABLE 2. 4 Zipf’s Distribution

Rank	Word	Percentage
1	the	6.8
2	of	3.1
4	to	2.7
4	and	2.6
5	in	1.8

6	is	1.2
7	for	1.0
8	that	0.8

Zipf saw that the second most normal word 'of' happens about half as regularly as the most widely recognized word 'the'. While the third most normal word 'to' happens about a third as regularly as 'the'. Etc. The seventh most normal word 'for' happens around one seventh as regularly as 'the'. All the more by and large, the recurrence of the nth most regular word is around 1/n times the recurrence of the most well-known word.

So a graph of the frequencies of the most common words looks roughly like this:

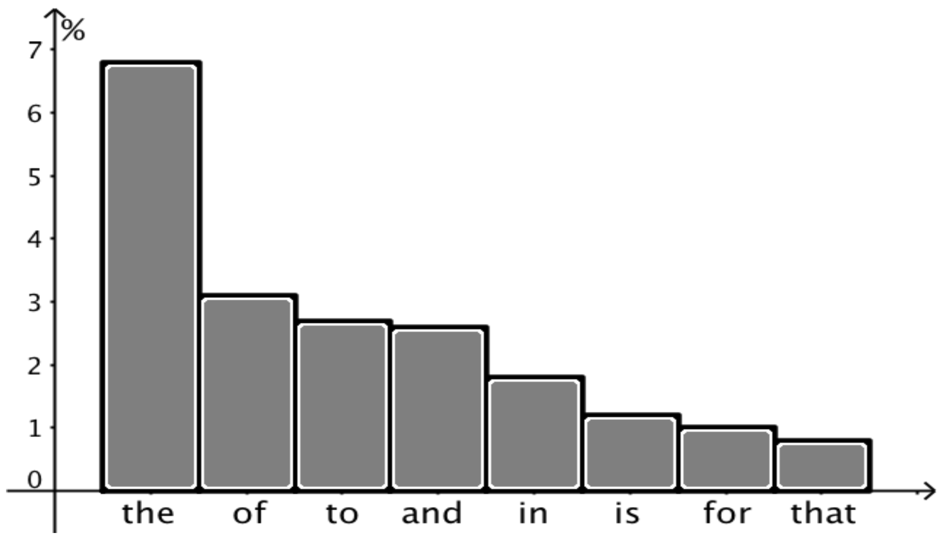


Figure 2.4 Zipf's Distribution

This distribution, remarkably, is quite stable over many different publications. Furthermore it turns out that less than 200 words account for more than half of all the written words in English.

2.6 Some Other Important Works

There are some important works on text compression is proposed in recent time. We summarizes a few of the important works.

Optimal Compressed Sensing and Reconstruction of Unstructured Mesh Datasets [1]

Compressed detecting (CS) is examined in [1] as an in situ technique to lessen the measure of the information as it is being created amid a huge scale reenactment. CS works by testing the information on the computational group inside an elective capacity space, for example, wavelet bases and after that reproducing back to the first space on representation stages. While much work has gone into investigating CS on organized datasets, for example, picture information, we explore its convenience for point mists, for example, unstructured work datasets frequently found in limited component reproductions. An example strategy is utilized that shows low lucidness with tree wavelets observed to be appropriate for point mists. It is recreated utilizing the stagewise symmetrical coordinating interest algorithm that was improved to encourage mechanized use in clump occupations.

Lightweight natural language text compression [2]

Variations of Huffman codes where words are taken as the source images are at present the most appealing decisions to pack regular language text databases [2]. Specifically, Tagged Huffman Code by Moura et al. offers quick direct looking on the packed text and irregular access capacities, in return for creating around 11% bigger compacted records. End-Tagged Dense Code and (s, c)- Dense Code, two new semi static measurable techniques for compressing common language texts. These strategies license more straightforward and quicker encoding and get preferable compression proportions over Tagged Huffman Code, while keeping up its quick immediate inquiry and arbitrary access capacities.

A Bit-level Text Compression Scheme Based on the ACW Algorithm [3]

A bit level, lossless, versatile, and asymmetric data compression conspire that depends on the versatile character word length (ACW(n)) algorithm is proposed in [3]. The proposed plan improves the compression proportion of the ACW(n) algorithm by isolating the parallel grouping into various subsequences (s), every one of them fulfilling the condition that the quantity of decimal qualities (d) of the n-bit length characters is equivalent to or under 256.

Evaluate Database Compression Performance and Parallel Backup [4]

This exploration tends to the challenges of the above issues and gives the answer for how to streamline and improve the procedure for pack the continuous database and execute reinforcement of the database in different gadgets utilizing parallel way. The proposed productive calculations gives the answer for pack the constant databases all the more viably and improve the speed of reinforcement and reestablish tasks [4].

n-Gram-Based Text Compression [19]

A content compression using n-gram word references is proposed in [19] which accumulates the content corpus of the Vietnamese language from the Internet and amasses five n-gram dictionaries with very nearly 500,000,000 n-grams, and a test set of 10 assorted content reports with different sizes to survey.

Efficient 6 bit Encoding Scheme for Printable Characters by table look up [26]

The scheme portray in [26] manages an encoding system by 6 bits for printable characters utilizing table turn upward. It changes over the 8 bit characters to 6 bits by partitioning the characters into 5 sets and utilizing them in a solitary table. The area of character is then utilized remarkably to encode by 6 bits.

Fast text compression using multiple static dictionaries [28]

A quick text compression technique dependent on numerous static word references is build up in [28]. This algorithm is language subordinate as a result of its static structure; in any case, it owes its speed to that structure. It performs compression with utilizing most every now and again utilized outlines and trigrams rather than ASCII codes that are not utilized or rarely utilized in text documents. In outline coding, at each coding venture the following two characters are examined to check whether they compare to a chart in the word reference. Provided that this is true, the relating record in the word reference is encoded; generally just the principal character is encoded. The coding position is then moved by a couple of characters as proper. On the off chance that three characters are utilized rather than two, the coding algorithm is named as trigram coding.

Text Database Compression Using Replacement and Bit Reduction [30]

A compression technique which depends on redundancy of words and number framework hypothesis is proposed in [30]. It utilizes a method wherein as often as possible happening words are supplanted by extraordinary characters and the altered document is considered as n-base number framework, where *n* is the quantity of various characters in the record. Further, compression process is completed by changing over this n-base number framework to paired number framework. The principle thought behind utilizing this algorithm is to speak to the entire data into lower number framework accordingly sparing bits prerequisite

On parsing optimality for dictionary-based text compression [40]

An overview is displayed [40] on the parsing issue for dictionary-based content pressure, distinguishing discernible consequences of both a hypothetical and pragmatic nature, which have showed up over the most recent three decades. We pursue the authentic strides of the Zip conspire indicating how the first ideal parsing issue of finding a parse shaped by the base number of expressions has been supplanted by the bit-ideal parsing issue where the objective is to limit the length in bits of the encoded content.

2.7 Discussion

Every compression techniques described in this chapter have some pros and cons. LZW is still the best data compression technique in term of compression ratio with time. Run length encoding and arithmetic coding is mainly used for data communication and soft error purpose. Huffman algorithm is an average algorithm process which has average results.

Though, there are a lot of research has been done on those different compression technique, but only a few researches have been made on competing with the well-known WinZip or WinRAR algorithm. Hence the proposed generalized representation scheme *nSmBC* will outperform over WinZip and WinRAR scheme. The detail of the proposed scheme is presented in the next chapter.

CHAPTER III

Compression Scheme for Large Natural Language Text

3.1 Introduction

We develop a compression method for natural text using *n-sequence* dictionaries.

Definition 3.1 (*n-Sequence*): *n-Sequence* is a sequence of words where *n* characters are arranged side by side. For example, if a character set contains {A,B} then 1 Sequence is <A,B>, 2 Sequence is <AA,BB,AB,BA>, 3 Sequence is <AAA,BBB,AAB,ABA,……>. It is the possible all combinations taking *n* characters from a given character set. Each of the member in the *n-sequence* is identified by an index number. If there are *k* members in an *n-sequence* set we index the members from 0 to (*k*-1). The index of "AB" in the 2-sequence is 2.

The proposed method is a scheme of 5 bit character encoding algorithm that represents a character by 5 bits rather than 8 bits. The characters we can represent by the decimal values is shown in Table 3.1 and 3.2. Each of the decimal value is in the range (0-31) which can be represented by 5 bits. Hence we represent 8 bit characters by 5 bits.

Following are the steps which are used to construct the proposed method:

1) Construction of the Look up table for encoding:

In this section we will construct a Look up table for representing the characters by 5 bit by randomly distributing the characters. The look up table is optimized by using a Zipf's distribution which is a discrete distribution of commonly used characters in different languages [6].

2) Construction of the *n-Sequence* dictionary:

A logical *n-Sequence* dictionary will be constructed in this section which is mathematically calculated and proved by an algorithm.

3) Compression and Decompression Algorithm:

We will describe the compression and decompression algorithm by providing suitable examples in this section.

3.2 Construction of the Look up table for encoding

Definition 3.2(Set tag): *Set tag* is defined as a group of characters tagged to a particular set number. For example, the capital letters, small letters, digits and symbols are tagged to particular set number as shown in TABLE 3.1 and TABLE 3.2. We call it *set tag* representation.

TABLE 3. 1 Random lookup table for *nSmBC*

Decimal value	**Binary value**	**Set- 1**	**Set- 2**	**Set- 3**	**Set- 4**	**Set- 5**	**Set- 6**	**Set- 7**
0	00000	অ	ণ	◌া	Space	Y	x	!
1	00001	আ	ত	ি◌	A	Z	y	"
2	00010	ই	থ	◌ী	B	a	z	#
3	00011	ঈ	দ	◌ু	C	b	1	$
4	00100	উ	ধ	◌ূ	D	c	2	%
5	00101	ঊ	ন	◌ৃ	E	d	3	&
6	00110	ঋ	প	ে◌	F	e	4	'
7	00111	এ	ফ	ৈ◌	G	f	5	(
8	01000	ঐ	ব	ে◌া	H	g	6	)
9	01001	ও	ভ	ে◌ৗ	I	h	7	*
10	01010	ঔ	ম	◌্	J	i	8	+
11	01011	ক	য	।	K	j	9	,
12	01100	খ	র	০	L	k	0	-
13	01101	গ	ল	১	M	l	_	.
14	01110	ঘ	শ	২	N	m	`	/
15	01111	ঙ	ষ	৩	O	n	{	:
16	10000	চ	স	৪	P	o	|	;
17	10001	ছ	হ	৫	Q	p	}	<
18	10010	জ	ড়	৬	R	q	~	=
29	10011	ঝ	ঢ়	৭	S	r	>	
20	10100	ঞ	য়	৮	T	s	?	
21	10101	ট	ৎ	৯	U	t	[	
22	10110	ঠ	◌ং		V	u	\	
23	10111	ড	◌ঃ		W	v	]	

24	11000	ট	◌ঁ		X	w	^	
25	11001	Set- 1						
26	11010	Set- 2						
27	11011	Set- 3						
28	11100	Set- 4						
39	11101	Set- 5						
30	11110	Set- 6						
31	11111	Set- 7						

After constructing this table, it was observed that there are some character available in the English literature that were used so little. Since the *n-SmBC* algorithm somehow depends on the set value, so if there happens a rapid change in set, the algorithm might not produce the desired result. Therefore, to solve this issue, the dictionary is optimized by using a distribution named Zipf's [27] which is a discrete distribution of commonly used characters in different languages.

The table is demonstrated in TABLE 3.2

TABLE 3. 2 Optimized look up table for *nSmBC*

Decimal value	**Binary value**	**Set- 1**	**Set- 2**	**Set- 3**	**Set- 4**	**Set- 5**	**Set- 6**	**Set- 7**
0	00000	অ	ণ	◌ৃ	E	e	1	Q
1	00001	আ	ত	◌্	T	t	2	X
2	00010	ই	থ	◌ঁ	A	a	3	Z
3	00011	এ	দ	◌ঃ	O	o	4	J
4	00100	◌া	ধ	ৎ	R	r	5	,
5	00101	◌ি	ঘ	ঢ়	I	i	6	?
6	00110	◌ী	প	ঐ	N	n	7	‘
7	00111	◌ু	ফ	ঊ	S	s	8	!
8	01000	◌ে	য়	ঔ	H	h	9	"
9	01001	◌ো	ভ	ঈ	D	d	0	#
10	01010	◌ৌ	ছ	।	L	l	+	\
11	01011	ক	ট	০	C	c	-	~
12	01100	খ	ঞ	১	U	u	*	^
13	01101	গ	ঠ	২	P	p	/	|
14	01110	ন	ঙ	৩	M	m	=	$
15	01111	শ	ষ	৪	W	w	(	:
16	10000	স	চ	৫	F	f	)	;
17	10001	ম	য	৬	G	g	{	_
18	10010	জ	ড়	৭	Y	y	}	`

29	10011	ব	ঝ	৮	B	b	<	
20	10100	র	ঊ	৯	V	v	>	
21	10101	ট	ও		K	k	[	
22	10110	ল	ঋ		z	x	]	
23	10111	ড	◌্		j	.	%	
24	11000	হ	◌ং		q	space	&	
25	11001			Set- 1				
26	11010			Set- 2				
27	11011			Set- 3				
28	11100			Set- 4				
39	11101			Set- 5				
30	11110			Set- 6				
31	11111			Set- 7				

Our target characters are all the characters of an English standard key board. We also consider the Bangla characters as well. We divide the characters into 7 sets namely Set-1, Set-2,…, Set-7. Each of the set contains 25 characters. The characters are placed in a lookup table as shown in Table 3.2. The entry in Table 3.2 is organized as follows:

1) Characters of the Bangla alphabet are placed in Set-1, Set-2 and Set-3.

2) Characters of the English alphabet are placed in Set-4, Set-5, Set-6 and Set-7. Position 21-24 of Set-3 and 19-24 of Set-7 is empty and can be filled with any missing characters.

3) The rest of the 7 combinations are filled with the 7 sets as shown in table 3.2.

Therefore, the table contains 32 characters (serial from 0 to 31). These 32 (2^5=32) combinations can be represented by 5 bits. Within the 32 combinations 25 combinations are utilized for converting the original 8 bit character to 5 bit and the rest of the 7combinations are utilized for representation of the sets. Therefore, we can use $(2^5 - 7) \times 7$ =175 characters in the Table. If we can take 6 bits then there can be $(2^6 - 7) \times 7$ =399 characters can be handled. We call it m bit representation of the scheme. In the following we represent m=5 bits to explain our proposed method. We represent any character using the encoding scheme (see Table 3.2). We call it *set representation*. For example, if we have a character stream "ABCDabcd956" then the *set representation* is "Set4 ABCDSet5abcdSet6956". Since "A" is located in Set4 so start with Set4 followed by "A", "a" is represented in Set5 we put Set5 before "a" and so on. When a set change occurs, we insert a Set number to distinguish it with others.

The placement of characters in the look up table is optimized using Zipf's distribution which is a discrete distribution of commonly used characters in different languages [6]. Zipf's law is an empirical law formulated using mathematical statistics. It states that given a large sample of words used, the frequency of any word is inversely proportional to its rank in the frequency table. So word number *n* has a frequency proportional to 1/n. Thus the most frequent word will occur about twice as often as the second most frequent word, three times as often as the third most frequent word.

We placed the characters in Table 3.2 such that the minimum number of set change occurs to handle the input string.

3.3 Construction of the *n-Sequence* Dictionary

After converting the 8 bit characters into 5 bits, we have a bit stream constructed from the 5 bits of each character. For any input text T, we create a bit stream (5 bits for each character) and from this bit stream we divide it take 4 bits each. We put trailing the last set number to make it mod 4 equal to zero if the length of the bit stream is not mod 4 equal to zero. From this 4 bits, we have $2^4 = 16$ different combinations of bits. Since each of the characters is represented by 8 bits, we add a fixed bit pattern in front of each of the 4 bits. Figure 3.1 shows an example. The fixed bit pattern is 0100.

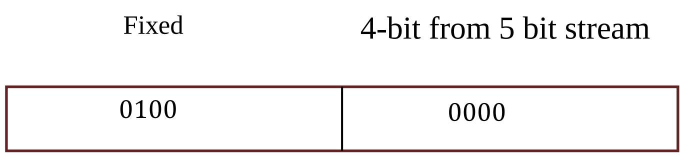


Figure 3.1: Adding Fixed bit pattern.

After adding fixed 4 bit pattern (0100) we set ASCII value range 64-79. Table 3.3 shows the characters along with it's decimal and ASCII values. Hence any of the characters shown in Table 3.2 becomes a character (@-O) shown in Table 3.3.

TABLE 3. 3 List of characters used for creating *n-sequence* algorithm

Serial No.	Characters	Decimal Value	Binary value
1	@	64	01000000
2	A	65	01000001
3	B	66	01000010
4	C	67	01000011
5	D	68	01000100
6	E	69	01000101
7	F	70	01000110

8	G	71	01000111
9	H	72	01001000
10	I	73	01001001
11	J	74	01001010
12	K	75	01001011
13	L	76	01001100
14	M	77	01001101
15	N	78	01001110
16	O	79	01001111

Example 1:

Original Text: "*Test Text* "

Set Representation:

Set4 T Set5 est space Set4 T Set5 ext

Decimal Representation:

28 1 29 0 7 1 24 28 1 29 0 22 1

5 bit representation:

11100 00001 11101 00000 00111 00001 11000 11100 00001 11101 00000 10110 00001

After Dividing by 4:

1110 0000 0111 1010 0000 0011 1000 0111 0001 1100 0000 1111 0100 0001 0110 0000 1111

Adding 0100 to every combination:

01001110 01000000 01000111 01001010 01000000 01000011 01001000 01000111 01000001 01001100 01000000 01001111 01000100 01000001 01000110 01000000 01001111

Corresponding ASCII Character:

N@GJ@CHGAL@ODAF@O

Dictionary Construction

Using the characters of Table 3.3, we generate a dictionary of *n-sequence* (See definition 3.1) of different values of *n*. Figure 3.2 shows the *n-sequence* for *n* =1, 2 and 3

@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

index 0 1 2 3 4 5 … 15

n-Sequence for n=1

@@	@A	@B	@C	@D	@E	…	…	…	O O

index 0 1 2 3 4 5 6 … 255

n-Sequence for n=2

@@@	@@A	@@B	@@C	@@D	…	…	…	…	O OO

index 0 1 2 3 4 5… 4095

n-Sequence for n=3

Figure 3.2: *n-sequence* for n=1, 2 and 3

Key generation

From those above *n-sequence* dictionary we generate a key of the form < k, ($v_{1,}$ $v_{2,}$ $v_{3,}$ $v_{4,}$ ...)> where k is the number of the *n-sequence* and *v* is the index value of the corresponding *n-sequence* dictionary (see Figure 3.2).

Logical Dictionary

The dictionary we generate using *n-sequence* generation is not stored in the physical memory. We implement the dictionary using a hash function *h(s)*. The function *h()* takes a string which is member of the *n-sequence* dictionary as input and returns the corresponding index of the dictionary. Hence the dictionary becomes a logical one and does not take any physical memory.

Hash function development

Forward Hash Function

Firstly, we assign all the 16 characters (see table 3.3) a value as follows $V_0 = @, V_1 = A, V_2 = B, V_3 = C, V_4 = D, V_5 = E, V_6 = F, V_7 = G, V_8 = H, V_9 = I, V_{10} = J, V_{11} = K, V_{12} = L, V_{13} = M, V_{14} = N, V_{15} = O.$

We store V_1, V_2... into the secondary stage and *n* as a single value in front end where $V_1 = 1, V_2 = 2, \dots, V_i = i$ ($1 \le i \le 15$)

The index value for *n-sequence* dictionary for different values of *n* is calculated as follows.

For n=1,

$h(s)$=$V_i + 1$

If s = “A” then $h("A") = V_1$+1 = 1+1= 2 [where i = 1]

For n=2,

$h(s)$= $(V_i * 16) + V_j$ + 1

If s = “AM” then $h(\text{AM}) = (V_1 * 16) + V_j + 1$ = (1*16) + 13+1= 30 [where i = 1 and j = 13]

For n =3,

$h(s)$= $(((V_i * 16) + V_j)$*16) + $V_k + 1$

If s = “BAG” then $h("BAG") = (((V_i * 16) + V_j)$ **16)* + $V_k + 1$= (((2*16) + 1) * 16) +7 + 1= 536 [where i = 2, j = 1 and k = 7]

Finally we generalize *h(s)* as

$h(s)$= $(((V_i$ * 16) +V_j) * 16 +V_k) * 16 +V_l)……+1

Where

$h(s)$= a string which is a member of *n-sequence* dictionary

V_i = Assigned number of the 1st character

V_j = Assigned number of the 2nd character

V_k = Assigned number of the 3rd character

V_l = Assigned number of the 4th character, and so on.

After getting the indexes using the above hash function we represent each index with 1 byte by using the Java OutputStreamWriter() function in Java platform which is used to convert the written characters to the bytes written to the underlying OutputStream. Here we convert the written index to ASCII which defines 1 byte.

We use the idea of *n-Sequence* for*m* bit representation hence call the scheme *nSmBC* (*n-Sequence* based *m* bit Compression).

Backward hash function

If the values of the index (i.e., the value of hash function) is known, then the corresponding values of the characters can be found by the backward hash function. The procedure is described as follows.

For n =1,

h= $V_i + 1$

Y = V_1 [h-1=Y]

$V_1 = Y$ mod 16

For n =2,

h= $(V_i * 16) + V_j$ + 1

Y = $(V_1 * 16) + V_2$ + 1 [h-1= Y]

$V_2 = Y$ mod 16

V_1 = Y / 16

For n =3,

h= $(((V_i * 16) + V_j)$*16) + $V_k + 1$

Y = $((V_1 * 16) + V_2)$*16) + $V_3 + 1$ [h-1= Y]

V_3 = Y mod 16

V_1 = [Y /16] / 16

V_2= [Y /16] mod 16

Hence the general equations becomes

V_n = Y mod 16

V_i= [[[Y /16] / 16]/16 …] mod 16 $[2 < i < (n-1) >]$

V_1 = [[[Y / 16] / 16] / 16 …] / 16

3.4 Compression and Decompression algorithm

In this section, the compression and decompression algorithms for *nSmBC* is briefly described in different steps. After the compression and decompression technique, an example is provided to show the working procedure of the algorithms.

3.4.1 Forward Mapping

Input**:** A string S to be compressed,

Output**:** An encoded compressed string Sc

Step1: Represent S to S′ as *set representation* adding Set tag.

Step2: Using the look up table, convert the string S′ by 5 bit stream. Let, in this stage the bit stream contains *k* bits.

Step3: d=*k%*4; if (d≠0) add trailing bits to the last set number to make d=0.

Step4: Store every 4 bit combinations in *k*.

Step5: Add 0100 in front of to every 4 bit combination of *k* to make the binary combination only limited to the characters Table 3.3.

Step6: Divide k by 8 to find the corresponding ASCII characters.

Step7: Create the logical *n-sequence* dictionary using forward hash function *h()* and store < *n*, index>.

Example 1:

Original Text (Input): "Test Text "

Step1: Set Representation: Set4 T Set5 est space Set4 T Set5 ext

Step2: Decimal Representation: 28 1 29 0 7 1 24 28 1 29 0 22 1

Step3: 5 bit representation: 11100 00001 11101 00000 00111 00001 11000 11100 00001 11101 00000 10110 00001

Step4: After Dividing by 4: 1110 0000 0111 1010 0000 0011 1000 0111 0001 1100 0000 1111 0100 0001 0110 0000 1111

Step5: Using Adding 0010 to every combination: 01001110 01000000 01000111 01001010 01000000 01000011 01001000 01000111 01000001 01001100 01000000 01001111 01000100 01000001 01000110 01000000 01001111

Step6: ASCII Representation :N@GJ@CHGAL@ODAF@O

Step7: Generate *n-Sequence* to get the *<n, index>* (*n*=4 used here): < 4, (57467, 904, 7184, 16737, 63422) >

3.4.2 Summarization of the Compression Algorithm

Algorithm 3.1: Summarization of the Compression Algorithm

// Sc is the integer value of the *n-sequence*

// S is the normal string

// Cs is the character set of corresponding index

// Sb is the normal string binary

Input: S

Output: Sc

```
for i = 1 to length of S do
    if S[i]= Dictioary then
        Sb = Binary(S[i])
        len = length of Sb / 4
    end if
while i=0 to len then
     Sb= 0100+ len(0:4)
end loop
while i=0 to Sb then
     T = Sb/8;
     Cs = T(Ascii)
      i = i + l
end loop
while i=0 to ch then
      x = Cs/n-Sequence
      for i=0 to value of n-sequence then
             y= Assigned value of the character
             x = (x * 16) + y; x = x + 1;
              Sc= Sc+  x*1 byte;
      end loop
end loop
```

3.4.3 Backward mapping

Input: Compressed String, Sc

Output: Uncompressed original string, S

Step1: Representing the string Sc by its corresponding <*n*, *index*> pair using backward hash function.

Step2: From the location of the pair <*n*, *index*> find the exact *n-sequence* character combination and store it in S_c'.

Step3: From S_c', find its corresponding binary combination from the ASCII Table (Table 3.3) and store the resultant binary bits in *k*.

Step4: Remove 0100 from every 8 bit binary combinations.

Step5: From the remaining bits stream, take 5 bits and representing it by the character set of the look up table (Table 3.2).

Step6: Remove the set number to get the original string *S*.

Example 2:

Compressed String: <4, (57467, 904, 7184, 16737, 63422 >

Step1: Corresponding string in *n-sequence* dictionary: N@GJ@CHGAL@ODAF@O

Step2: From 8 bit Representation: 01001110 01000000 01000111 01001010 01000000 01000011 01001000 01000111 01000001 01001100 01000000 01001111 01000100 01000001 01000110 01000000 01001111

Step3: Removing 0100 from every 8 bit combination: 1110 0000 0111 1010 0000 0011 1000 0111 0001 1100 0000 1111 0100 0001 0110 0000 1111

Step4: From 5 bit representation: 11100 00001 11101 00000 00111 00001 11000 11100 00001 11101 00000 10110 00001

Step5: Decimal Number corresponding to 5 bits: 28 1 29 0 7 1 24 28 1 29 0 22 1

Step6: Corresponding Set Representation: Set4 T Set5 est space Set4 T Set5 ext

Step7: Original Text: Test Text

3.4.4 Summarization of the Decompression Algorithm

Algorithm 3.2: Summarization of the De-Compression Algorithm

// Sc is the integer value of the *n-sequence*

// S is the normal string

// Cs is the character set of corresponding index

// Cb is the binary value of character set

// Sb is the normal string binary

// PSet is the previous set value

Input: Sc

Output: S

```
for i = 1 to length of Sc do
  index=Sc[i]
  Cs = Cs + get character set from index
end loop
Cb = Convert Cs to its corresponding binary
Cb = Discard first 4 bit (0100) from every binary
for i = 1 to length of Cb do
  if  (Cb[i] == Set ) then
     if  (Cb[i] == PSet ) then
        Break loop
     end if
     PSet = Cb[i]
  end if
  else then
       S = S+ PSet + Cb[i]
  end else
end loop
```

3.5 Conclusion

This chapter explains the proposed idea briefly. Firstly, it describes the construction of the encoding table. Secondly, it explains how n- sequence algorithm can be developed. Finally it describes the compression and decompression technique with a particular example to show the truthfulness of the proposed algorithm. In the next chapter the theoretical prove of the algorithm will be provided.

CHAPTER IV

Performance Analysis

4.1 Introduction

In this chapter, the theoretical and experimental analysis of this algorithm is derived. As well as, how the efficiency rate will be calculated precisely is also discussed. Firstly, the algorithm to calculate the rate of efficiency is derived by using some parameters. Finally, after using this algorithm, how the efficiency rate will be variate is discussed briefly.

Then in the experimental analysis section, it simulates the algorithms for compression ratio, compression of file size as described in chapter III. It also provides some applications of this algorithm in Database. As well as it will discuss some critical issues regarding implementing this technique.

4.2 Theoretical Analysis

In this section, the analytical evaluation of the proposed scheme is done. Table 4.1 shows the parameters for analytical evaluation. Some parameters are provided as input while others are derived from the input parameters. All lengths and sizes are in bits.

TABLE 4.1 Parameters for analytical evaluation

Parameter	**Description**
N	Total number of characters in the input string
S_1	Size of the input string, $S_1 = N \times 8$ bit
m	Number of bits used to compress the input character using lookup table (Table 3.2)
ε	Number of bits required to store *Sets* for set representation

β	Number of bits used to create the converted characters of Table 3.3
S_2	Size of input string using m bit representation, $S_2 = N \times m + \varepsilon$ bit
v	Number of characters generated from S_2 by taking β bits, $v = \frac{S_2}{\beta}$
q	Number of indices to store, $q = \frac{v}{n}$
α	Size of one index
S_3	Size of q, $S_3 = q \times$ α (Compressed file size)
η	Compressed ratio, $\eta = \frac{S_3}{S_1}$
σ	Savings of space σ=(1- η)×100%

$$\eta = \frac{q*\alpha}{N*8}$$

$$= \frac{v * q*\alpha}{n* N*8}$$

$$= \frac{S_2 * \alpha}{\beta *n* N*8}$$

$$=\frac{N*m+ \varepsilon \ * \alpha}{\beta *n* N*8},$$ [We assume the size of the *Sets* are negligible $\varepsilon \approx 0$]

$$= \frac{N*m* \alpha}{\beta *n* N*8} = \frac{m* \alpha}{\beta *n*8}$$

$$=\frac{m* 8}{\beta *n*8},$$ [α = 1 byte = 8 bit]

$$\eta = \frac{m}{\beta *n}$$

Using the above equation, we evaluate the trend of η with varying values of *n* (6 to 15). Figure 4.1 shows the evaluated result. From the Figure 4.1 we can say that the performance of the proposed *nSmBC* depends on the value of *n* i.e. if the length of *n-sequence* is large the performance will be better. The performance also depends on the value of *m* and *β.* If m is large then performance will be lower if m is very small then small number of characters will be accommodated in the lookup table. If the value of β increases then performance will also be increased but when β increase then the no. of characters also increase in Table 3.3.

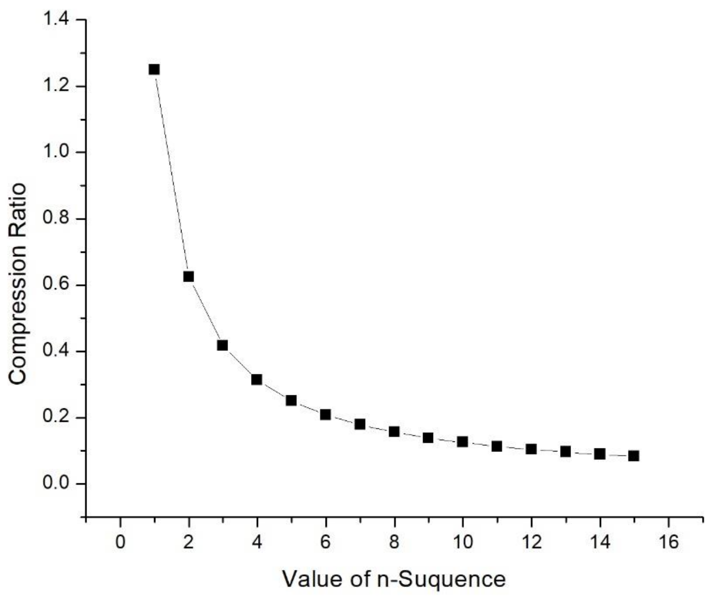


Figure 4.1 Compression Ratio for different *n-Sequence* value

4.3 Experimental Analysis

At that point in the experimental investigation segment, it mimics the algorithms for compression ratio, compression of file size as depicted in part III. It likewise gives a few utilizations of this algorithm in Database. Just as it will examine some basic issues in regards to executing this method.

4.3.1 Experimental Setup

The construction of prototype system for efficient compression scheme for large natural text dataset are done on a machine having the following specification. The experimental setup is visualized in Table 4.2

TABLE 4. 2 Experimental Setup

Parameter	**Specification**
Processor	Intel Core i7 6700
No. of Processor	1
Clock Speed	3.40 GHz
Cache Memory	1406 MB
RAM	8 GB
HDD	1.0 TB
Operating System	Windows 10 Pro 64-bit
Compiler	Java
Compiler Optimization	None

We have implemented our proposed algorithm in Java NetBeans IDE 8.2 with the parameter values shown in Table 4.3. In this Section we present the experimental results. In our experiment, we used raw data set collected from Microsoft Research (MSR) Abstractive Text Compression Dataset [49]. The details of the dataset can be found in [50]. Table 4.4 shows the description of the dataset.

This is a manually-created, multi-reference dataset for abstractive sentence and short paragraph compression [50]. The impact of single- and multi-sentence level editing operations on human compression quality as found in this corpus. The correlations between automatic evaluation metrics and human judgments of meaning preservation and grammaticality in the compression task, and analyze the impact of the linguistic units used and precision versus recall measures on the quality of the metrics has been explored in it.

TABLE 4. 3 Parameters for experimental evaluation

n	*S1(MB)*	*m*	β	α
6-15	0.5,1.05,2.0,3.07,5.03	5	4	1

TABLE 4. 4 Dataset description

Sl. No.	Description
Dataset 1	Size: .5MB , No. of characters : 514,055
Dataset 2	Size: 1.05MB , No. of characters : 1,102,371
Dataset 3	Size: 2.0MB , No. of characters : 2,103,453
Dataset 4	Size: 3.07 MB , No. of characters : 3,228,053
Dataset 5	Size: 5.03 MB , No. of characters : 5,283,107

Figure 4.2 shows the experimental results for compression ratio with varying values of n for *nSmBC*. It demonstrates that when the value on *n* increases the value of η decreases. For *n* = 15, η reduces to 0.08 which means the σ is 92% as shown in Figure 4.3

When *n* increases, η reduces because, η is depends mainly on *n, m* and β. For increasing the value of *n,* more characters can be increased to include to a single index.

Hence the η will reduce and the σ will increase as shown in Figure 4.3. This is what we shown in our analytical evaluation in Section 4.2 (see Figure 4.1). Hence we validate our analytical model.

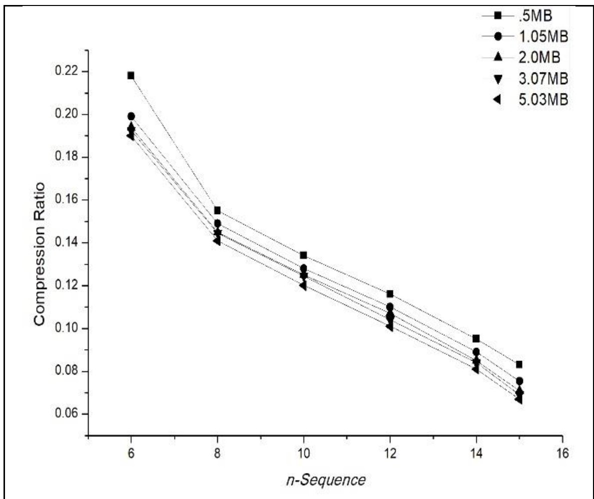


Figure 4.2: Compression Ratio for different *n-Sequence* value

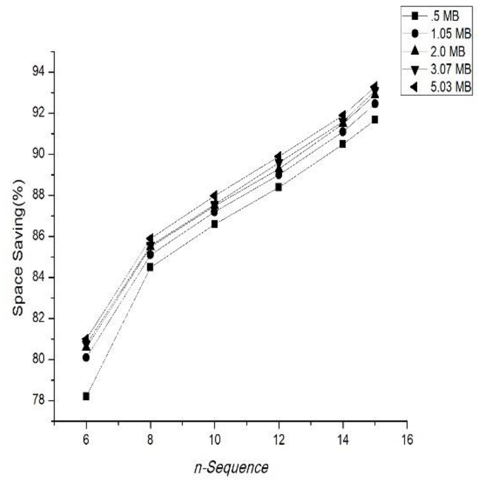


Figure 4.3: Space savings for different *n-Sequence* value

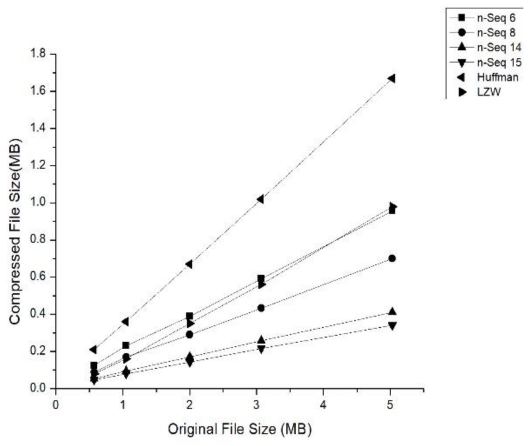
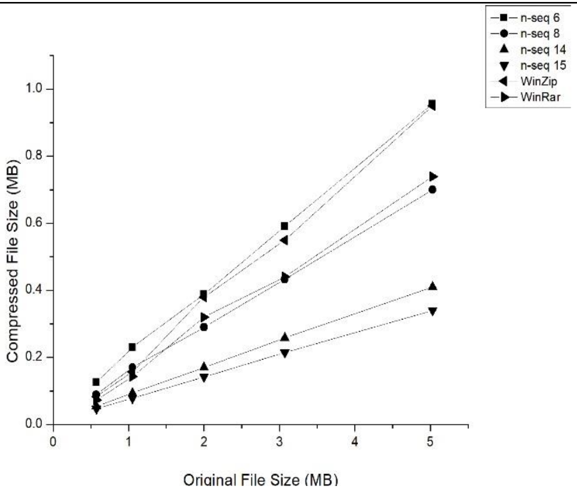
4.3.2 Experimental Test

The different version of the *nSmBC* technique i.e. different *n-sequence* is compared with WinZip, WinRAR, Huffman and Lzw algorithm. Actually 2 kind of test is experimented in this section comprising the Compression of files and Compression ratio.

- **Compression of files:** Files of different size is applied to all the algorithms and the compressed size of the original file after compression is observed.
- **Compression ratio:** Files of different size is applied to all the algorithms and the compression ratio is observed.

4.3.2.1 Compression of files

We compare our proposed technique with well-known LZW, Huffman, WinZip and WinRar techniques. The experimental result is shown in the following Figure 4.4 and Figure 4.5.

 Figure 4.4: Comparison with compressed file size for LZW and Huffman	 Figure 4.5: Comparison with compressed file size for WinZip and WinRAR

The *nSmBC* outperforms Huffman technique for *n* = 6, 8, 14 and 15. LZW shows good results but the *nSmBC* scheme outperforms LZW for *n* = 14 and 15. For all the cases Huffman shows worst result.

We also compare our technique with two industrial softwares WinZip and WinRAR. Figure 4.5 shows the comparison with WinZip and WinRAR for compressed file size. The *nSmBC* performs well than WinZip and WinRAR for *n* = 14 and 15. The WinZip performs well for small S_1, when S_1 increases the *nSmBC* performs well even for *n* = 8.

4.3.2.2 Compression Ratio

We compare our proposed technique with well-known WinRar and WinZip Techniques. The experimental result is shown in the following Figure 4.7 and Figure 4.8.

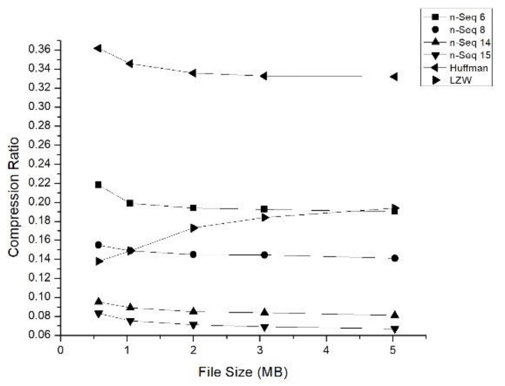


Figure 4.7: Comparison with compression ratio for LZW and Huffman technique

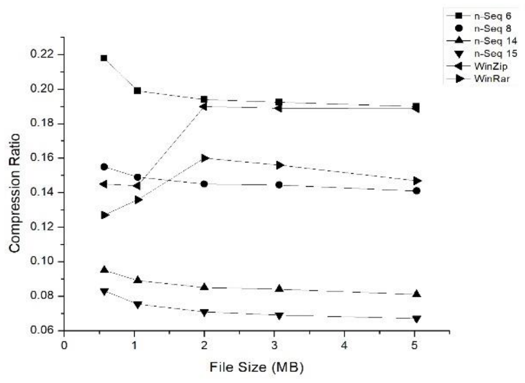


Figure 4.8: Comparison with compression ratio for WinZip and WinRAR

The comparison for η with LZW and Huffman is shown in Figure 4.7. The result for *nSmBC* is shown for *n* =6, 8, 14 and 15. The Huffman shows poor result among the schemes. The reason behind Huffman technique to be poor is that the data is derived by Huffman from the frequency of occurrence of the possible values in the source symbol. So if the size of data is quite large then a large number of individual symbols will be created. As a result, it shows poor η comparing to others. It's η ranges from 0.35 to 0.4 leading to σ = 60%-65%. The result demonstrates that *nSmBC* provides the best compression ratio. It reaches to η = 0.08 (σ = 92%) for *n* = 15. The other values *n* also provides good performance.

Figure 4.8 shows the comparison for η with WinZip and WinRAR. The WinRAR shows η = 0.10 (σ = 90%) at initial level. But at the increasing S_1, η degrades to 0.15 (σ = 85%). In case of WinZip, it also shows same type of values for η as WinRAR at the initial stage but not as good as WinRAR. At the increasing S_1, the compression ratio fluctuates between 0.16 to 2.0 (σ = 80-85%). The *nSmBC* shows better performance and it outperforms WinZip and WinRAR for *n*=14 and 15. Finally, we conclude that the *nSmBC* outperforms other techniques.

4.4 Dictionary Size

If the *nSmBc* dictionary is not developed logically, we have to store the dictionary in the physical memory. So a database must be created for each of the n-sequence value. So a fatal flaw might be created by doing this. A graphical representation is shown in Figure 4.10 to discuss this fatal flaw.

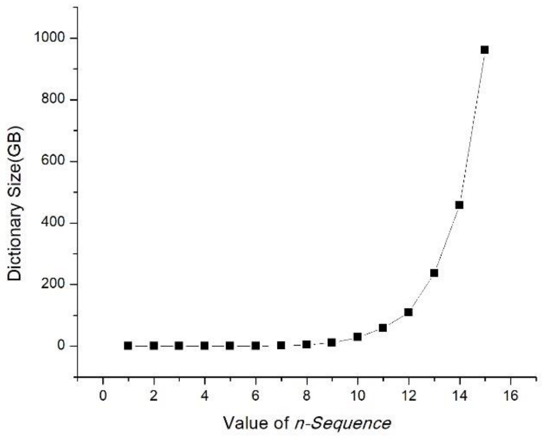


Figure 4.10 Dictionary size of the Database for different *n-Sequence* value

From Figure 4.10, it can be seen that by the increasing value of *n-Sequence* the dictionary size in the database in increasing gradually. And at some point it might touch 100 Gb. So a huge space in physical memory can be needed to save this database. By developing this *nSmBc* technique using the logical dictionary without using the physical memory, we can save this huge amount of space and time.

4.5 Applications In Database

To show the applicability of the *nSmBc* scheme, we apply the compression scheme in database application with the parameter values shown in Table 4.5.

TABLE 4. 5 Parameters for experimental evaluation

n	*S1(MB)*	*m*	β	α
6-15	0.112,.310,.594,1.01	6	4	1

We compare with the following algorithms:

1) 6BC

2) Huffman

3) LZW

4) 6BC+Huffman and

5) 6BC+LZW.

The comparison is based on the following database operations:

1. Selection Operation:

   Selection is applied to table of different sizes.

$$\sigma C1 = V1\ (T1)$$

Where, C_1 is the column name and V_1 is the value to retrieve.

2. Join Operation:

   Join is applied to table of different sizes.

$$\Delta C1 = V1\ (T1 \bowtie T2)$$

3. Selection with Projection (SP) Operation:

   Selection with Projection (SP) is applied to table of different sizes.

$$\pi C1, C2 \ldots Ck\ (\pi)$$

4.5.1 Compression Ratio

Figure 4.11 exhibits the ratio of compression for Selection. Excellent proficiency is appeared by Huffman and LZW. An average compression proportion is accommodated by 6BC strategy. In any case, the huge fact is that on the off chance that after uniting these two methodologies with 6BC, it shows awesome viability much superior to 6BC. The purpose for this is, the table of Huffman is gotten from assessed likelihood or recurrence of event (weight) for every conceivable estimation of the input. Furthermore, what's more In LZW, characters series are being supplanted by the single codes. It incorporates each recent characters series which consequently is used to establish a strings table. Compression occurs during the time of a single code is yield instead of a character series.

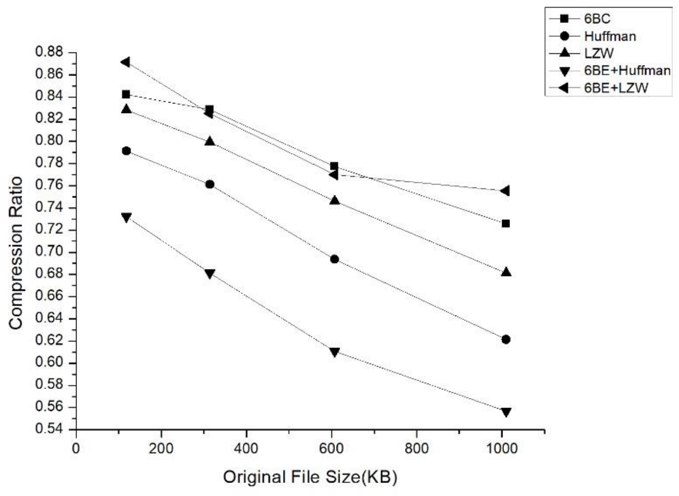


Figure 4.11 Compression ratio for selection

Figure 4.12 exhibits the ratio of compression for the join task. In this chart it was perceptible that Huffman and LZW algorithm gives very awing compression proportion. The 6BC procedure gives an ordinary compression proportion. Be that as it may, in the wake of consolidating Huffman and 6BC improves the exhibition.

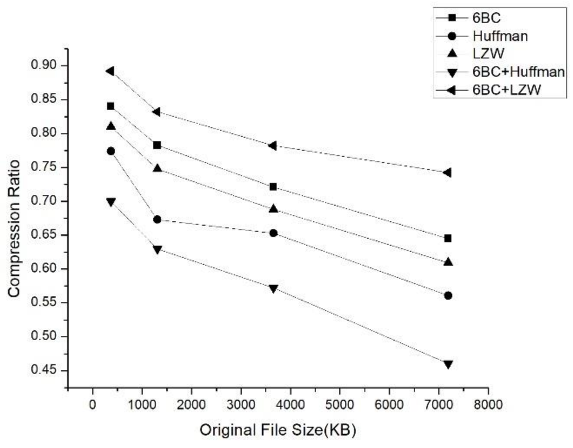


Figure 4.12 Compression ratio for Join

Figure 4.13 exhibits the compression proportion for selection with projection operation. In this diagram, the 6BC procedure gives a low compression proportion toward the start of the compression. In any case, it furnishes a normal compression as for the expanding information measure which furnishes a similar outcome with other compression procedure for the huge informational collection. Likewise consolidating with Huffman and 6BC the best execution as same as all the past database activity appears.

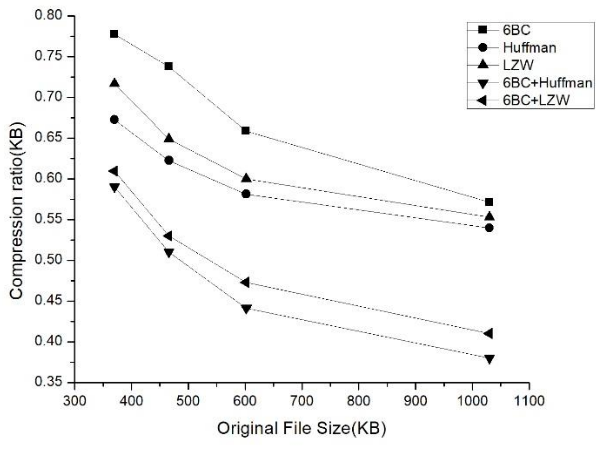


Figure 4.13 Compression ratio for Selection with Projection

4.5.2 Compression time:

Figure 4.14, 4.15 and 4.16 exhibits the compression time for all the above 5 methods for selection, join and projection separately.

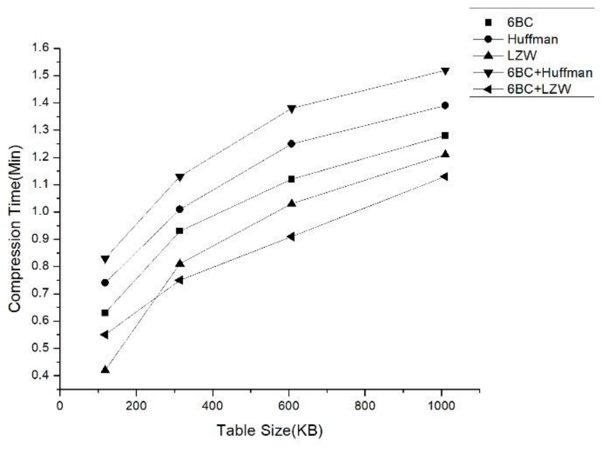


Figure 4.14 Compression time for selection

Figure 4.14 exhibits that 6BC exhibits promising performance. 6BC+LZW gives the best result. Figure 4.15 and Figure 4.16 provide that 6BC and LZW give noteworthy execution.

The reason is, actually 6BC creates packed content commonly littler in size than the original content size. Furthermore, that compacted content is furthermore packed by 6BC+Huffman and 6BC+LZW methods. So execution provides better performance gradually.

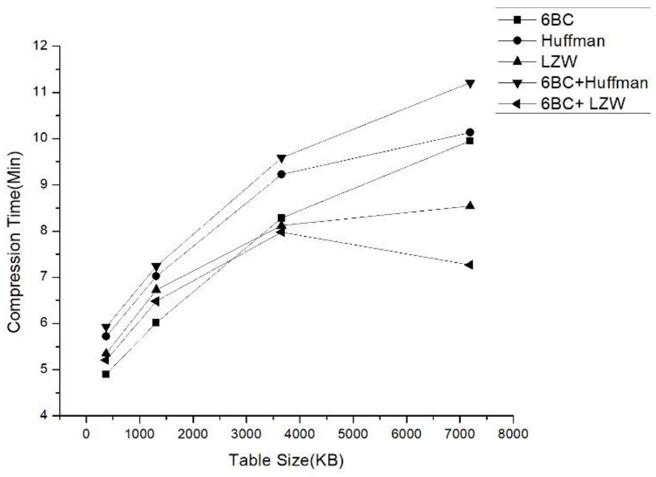


Figure 4.15 Compression time for join

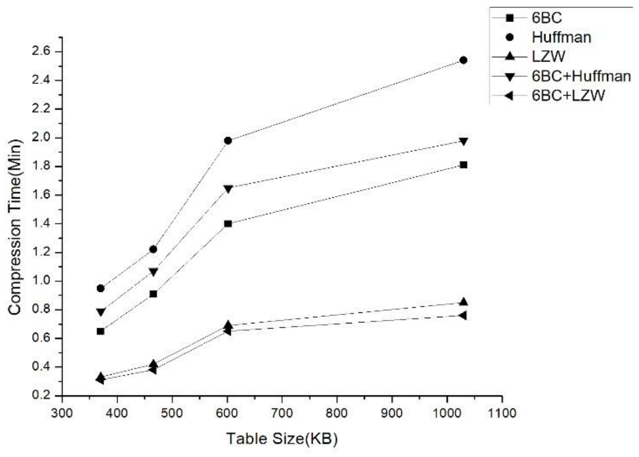


Figure 4.16 Compression time for selection with projection

4.5.3 Retrieval performance

The following range key query is executed for the performance of retrieval,

$$SELECT\ *\ FROM\ T1\ WHERE$$

$$DepId\ <\ V1\ AND\ Id\ <\ V2$$

This range query is applied to a 466 kb size table. The retrieval time of the 5 methods is calculated. Figure 4.17 exhibits the performance of retrieval. We see that 6BC method provide an average result. Huffman and LZW demonstrates a decent exhibition.

6BC+Huffman and 6BC+LZW gives the best outcome. The purpose for this is after compacting the original content by 6BC the compact content was very shorter than the first content. So during the time of compacting again it by LZW and Huffman, performance enhances.

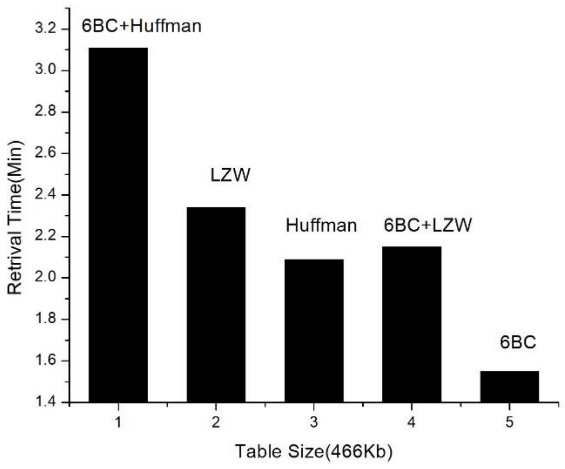


Figure 4.17 Retrieval performance

In the same way *nSmBC* can be applied to database. Though it is quite obsevable that

nSmBC provides far far better result than 6BC so it will provide some best besult for sure.

4.6 Critical Issue

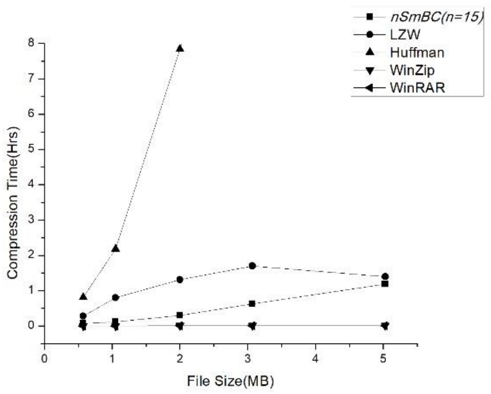


Figure 4.18 Compression time of *nSmBC* with the others techniques

There are some minor critical issue or we can say some minor limitation is available in this *nSmBC* technique. Since we know, Compression techniques are quite time consuming issue.

So time is the main critical issue in this algorithm. Figure 4.18 provides the compression time of different algorithms. So time is the main critical issue in this algorithm. We use Java currentTimeMillis() function to calculate the time of the *nSmBC*, LZW and Huffman method. Since WinZip and WinRAR are industrial techniques so we take their time manually. From Figure 9, it shows that *nSmBC* provides better result than Huffman. Initially LZW provides bad result but at the increasing size of the file it will show the similar range as *nSmBC*. But WinZip and WinRAR provides best result. But this time issue can be resolved by utilizing this algorithm in high configuration computer.

4.7 Discussion

Firstly, this chapter describes the theoretical proof of the compression algorithm and the efficiency rate. It also provides some logical calculation of the compression ratio. And also the best case as well as the worst case. The experimental result complying with the theoretical analysis which is shown in the experimental analysis part.

In the next section, it shows the experimental result of *nSmBC* technique. The experimental results comply with the theoretical analysis. *nSmBC* works quite impressively in large file size data. It beat the compression ratio of WinZip, WinRAR, LZW and Huffman in a quite long margin. Also an application of this algorithm to database operation is also provided.

CHAPTER V

Conclusions

5.1 Concluding Remarks

In this thesis, we present a novel method for text compression. The thesis proposes the idea on *n-sequence* and construction logical dictionary. The large dictionary is implemented of a hash function. The proposed *nSmBC* takes 5 bits for each character using a lookup table. Analytical and experimental results are presented to show the superiority of the scheme. The scheme can be able to compress up to 92% for web based diverse data set. The scheme shows superior performance to existing schemes namely LZW, Huffman and also for WinZip and WinRAR. The technique can easily be utilized to compress large amount of natural language text. Both the forward and backward mapping algorithms are presented. Since the algorithm use the logical dictionary for compression so that a particular text pattern retrieval is not possible. This technique can also be utilized to parallel processing environment as well as load balancing technique to achieve promising encoding time. We believe, the *nSmBC* is an efficient algorithm for compression that has the potential to compete with the existing text compression techniques.

5.2 Future Scope

The future direction of this research may be summarized as below:

- This technique can be further applied to database technology. Compressed dataset can be applied to database technology by implementing database operation like Selection, Projection, Join and Group by on the dataset.
- An innovative and efficient software can be developed which can be applied in data communication and data storage.
- It is quite possible by making this technique commercially usable, it can provide excellent result in data compression arena.

- Dependency in existing WinZip and WinRAR compression can be reduced by using this compression technique.
- By using supercomputer or other high configuration machines, the time issue can be easily resolved.
- The idea can be implemented in parallel and multiprocessor environment easily.

REFERENCES

[1] S. Maher, N. D. Fabian, D. M. Hensinger, J. Lee, E. M. Allendorf, A. Bhagatwala, M. L. Blaylock, J. H. Chen, J. A. Templeton, and I. Tezaur. "Optimal Compressed Sensing and Reconstruction of Unstructured Mesh Datasets."*Data Science and Engineering,* vol. 3, no. 1, pp. 1-23, 2018.

[2] Nieves, R. Brisaboa, A. Fariña, and G. Navarro "Lightweight natural language text compression", *Information retrieval*, vol. 10, no. 1, pp. 1-33, 2007.

[3] H. A. Bahadili, and S. M. Hussain, "A Bit-level Text Compression Scheme Based on the ACW Algorithm," *International Journal of Automation and Computing*, vol. 7, no. 1, pp.123-131, 2010.

[4] M. Murugesan and T. Ravichandran, "Evaluate Database Compression Performance and Parallel Backup," *International Journal of Database Management Systems (IJDMS)*, vol. 5, no. 4, pp. 17-25, 2013.

[5] S. S. Sundaram and R. Lourdusamy, "A Comparative Study Of Text Compression Algorithms," *International Journal of Wisdom Based Computing*, vol. 1, no. 3, pp. 68-76, 2011.

[6] W. Li "Random Texts Exhibit Zipfs-Law-Like Word", *IEEE Transactions On Information theory*, vol. 38, No. 6, November 1992.

[7] V. H. Nguyen, H. T. Nguyen, H. N. Duong, and V. Snasel, "Trigram-based Vietnamese text compression," in *Recent Developments in Intelligent Information and Database Systems*, vol. 642 of Studies in *Computational Intelligence*, pp. 297–307, Springer, 2016.

[8] B. Žalik and N. oLukač, "An Chain code lossless compression using move-to-front transform and adaptive run-length encoding," *Signal Processing: Image Communication*" vol. 29, no. 1, pp. 96–106, Elsevier, 2014.

[9] C.E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[10] R.M. Fano, "Transmission of information" *Tech. Rep., Massachusetts Institute of Technology, Research Laboratory of Electronics*, Cambridge, Mass, USA, 1949.

[11] J. Wu, Y. Wang, L. Ding, and X. Liao, “Improving performance of network covert timing channel through Huffman coding,” *Mathematical and Computer Modeling*” vol. 25, no. 1-2, pp. 69–79, Elsevier, 2012.

[12] G. Howard and J. S. Vitter, “Arithmetic coding for data compression,” *Proceedings of the IEEE*, vol. 82, no. 6, pp. 857–865, 1994.

[13] I. H. Witten, R. M. Neal, and J. G. Cleary, “Arithmetic coding for data compression,” *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[14] T. A. Welch, “Technique for high-performance data compression,” *IEEE Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[15] J. T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, and Simon J. Puglisi, “LZ77-Based Self-indexing with Faster Pattern Matching,” *A. Pardo and A. Viola (Eds.): LATIN 2014, LNCS 8392*, pp. 731–742, Springer-Verlag Berlin Heidelberg 2014.

[16] H. Bannai, S. Inenaga, and M. Takeda, “Efficient LZ78 Factorization of Grammar Compressed Text,” *L Caldr´on-Benavides et al. (Eds.): SPIRE 2012, LNCS 7608*, pp. 86—98, Springer-Verlag Berlin Heidelberg 2012.

[17] J. Cleary and I. Witten, “Data compression using adaptive coding and partial string matching,” *IEEE Transactions on Communications*, vol. 32, no. 4, pp. 396–402, 1984.

[18] M. Burrows and D. Wheeler, “A block-sorting lossless data compression algorithm,” *Digital SRC Research Report*, 1994.

[19] Nguyen, H. T. Nguyen, H. N. Duong and V. Snasel ” n-Gram-Based Text Compression, ” *Computational Intelligence and Neuroscience*, vol. 2016, no. 9483646, pp. 1-11, 2016.

[20] Mishra, S. Prakash, Col G. Singh, and R. Prasad. "A review on compressed pattern matching." *Perspectives in Science*, vol. 8, pp. 727-729.

[21] Anisimov, Anatoly V., and Igor O. Zavadskyi. "Variable length prefix (Δ, k)-codes." In *2015 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pp. 43-47, 2015.

[22] Peng, Min, W. Gao, H. Wang, Y. Zhang, J. Huang, Q. Xie, G. Hu, and G. Tian. "Parallelization of massive text stream compression based on compressed sensing." *ACM Transactions on Information Systems (TOIS)*, vol. 36, no. 2, pp. 17, 2017.

[23] J. Dvorsk, J. Pokorn, and J. Sna´sel, “Word-based compression methods and indexing for text retrieval systems,” in *Proceedings of the 3rd East European Conference on Advances in Databases and Information Systems (ADBIS ’99)*, pp. 75–84, Maribor, Slovenia, 1999.

[24] B.A. Al-hmeary, “Role of Run Length Encoding on Increasing Huffman Effect in Text Compression”, *Journal of Kerbala University*, vol. 6, no.2, 2008.

[25] B. Stankić, D. Kojić, M. Cvetanović, M. Dukić, S. Stojanović, and Zaharije, “ERLE: Embedded run length image encoding,” *2014 22nd Telecommunications Forum Telfor (TELFOR)*, pp. 975-978, 2014.

[26] A. Mahmood, T. Latif, and K. M. A. Hasan, “An Efficient 6 bit Encoding Scheme for Printable Characters by table look up”, *International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pp. 468-472, 2017.

[27] Fagan, Stephen, Gençay, and Ramazan, "An introduction to textual econometrics", *Handbook of Empirical Economics and Finance*, pp. 133–153, 2010.

[28] A Carus and A Mesut, “Fast text compression using multiple static dictionaries,” *Information Technology Journal*, 1013-1021, 2010.

[29] Md. A. K. Azad, R. Sharmeen, S. Ahmad, and S. M. Kamruzzaman, “An Efficient Technique for Text Compression” *The 1st International Conference on Information Management and Business*, pp. 467-473, 2005.

[30] A. Kumar, Sk S. Ali, and D. Chakraborty “Text Database Compression Using Replacement and Bit Reductionn”, *SIPM, FCST, ITCA, WSE, ACSIT, CS & IT 06*, pp. 407–416, 2012.

[31] C. Y. Lin, Y. C. Chung, and J. S. Liu, "Efficient data compression methods for multidimensional sparse array operations based on the EKMR scheme", *IEEE Transactions on Computers*, vol. 52, no. 12, pp. 1640-1646, 2003.

[32] J. Lansky and M. Zemlicka, “Compression of small text files using syllables,” in *Proceedings of the Data Compression Conference, Snowbird, Utah, USA*, March 2006.

[33] W. Huang, W. Wang, and H. Xu, “A Lossless Data Compression Algorithm for Real-time Database,” *2006 6th World Congress on Intelligent Control and Automation*, pp. 6645-6648, 2006.

[34] Bonfield, James K., Shane A. McCarthy, and R. Durbin. "Crumble: reference free lossy compression of sequence quality values." *Bioinformatics,* vol. 35, no. 2, pp. 337-339, 2018.

[35] S. Khalid, “Introduction to data compression”, *Morgan Kaufmann*, 2017.

[36] J. Kim, J. Lee, and J. Lee, “Performance of Low-Density Parity Check Codes with Parity Encoded by (1, 7) Run-Length Limited Code for Perpendicular Magnetic Recording,” *IEEE Transactions on Magnetics*, vol. 48, no. 11, pp. 4610 - 4613, 2013.

[37] Azeez, N. A. and A. A. Lasisi. "Empirical and Statistical Evaluation of the Effectiveness of Four Lossless Data Compression Algorithms." *Nigerian Journal of Technological Development*, vol. 13, no. 2, pp. 64-73, 2016.

[38] M. Vaidya, E. S. Walia, and A. Gupta, “An Data compression using Shannon-fano algorithm implemented by VHDL, *IEEE International Conference on Advances in Engineering & Technology Research*, pp. 1–5, 2014.

[39] M. M. Kodabagi, M. V. Jerabandi, and N. Gadagin, “Multilevel security and compression of text data using bit stuffing and huffman coding”, *2015 International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT),* pp. 800 - 804, 2015.

[40] A. Langiu. “On parsing optimality for dictionary-based text compression,” *Journal of Discrete Algorithms*, pp. 65-70, 2013.

[41] M. Sangeetha, P. Betty, and G.S. Nanda Kumar, “A biometrie iris image compression using LZW and hybrid LZW coding algorithm,” *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pp. 1-6, 2013.

[42] Z. Tang, “One adaptive binary arithmetic coding system based on context,” *2011 International Conference on Computer Science and Service System (CSSS)*, pp. 1440 - 1443, 2011.

[43] N. Francisco, D. Nobre, C. de Souza, Baptista, and C. EC Campelo, "Combining Markov model and prediction by partial matching compression technique for route and destination prediction." *Knowledge-Based Systems*, vol. 154, pp. 81-92, 2018.

[44] F. Luo, Z. Huang, F. Yan, and D. Sun, “Route memorization in real-time data processing using Run-Length Encoding”, *2009 IEEE Intelligent Vehicles Symposium*, pp. 1354 - 1358, 2010.

[45] A. Huffman, “A method for the construction of minimum redundancy codes,” *Proceedings of the IRE,* vol. 40, no. 9, pp.1098–1101, 1952.

[46] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[47] W. L. Tai, C. S. Chan, and C. A. Chu, “Apply Run-length encoding on pixel differences to do image hiding,” *2013 9th International Conference on Information, Communications & Signal Processing*, pp. 1-5, 2013.

[48] Sze, V. and Budagavi, Madhukar, “Reducing context coded and bypass coded bins to improve context adaptive binary arithmetic coding (CABAC) throughput”, *U.S. Patent 9*, pp. 584,802, 2017.

[49] https://www.microsoft.com/enus/download/details.aspx?id=54262

[50] C. Toutanova, C. Brockett, Ke M. Tran, and S. Amershi, “A Dataset and Evaluation Metrics for Abstractive Compression of Sentences and Short Paragraph” *Empirical Methods in Natural Language Processing, EMNLP*, pp. 340-350, 2016.