Tribhuvan University

Institute of Science and Technology

A Comparative Study of Lossless Data Compression Algorithms

A Dissertation

Submitted To

Central Department of Computer Science and Information Technology

Tribhuvan University

Kirtipur, Kathmandu, Nepal

In Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computer Science and Information Technology

Submitted By

Suresh Thapa

CDCSIT, TU

(December, 2012)

Tribhuvan University

Institute of Science and Technology

A Comparative Study of Lossless Data Compression Algorithms

A Dissertation

Submitted to

Central Department of Computer Science and Information Technology

Tribhuvan University

Kirtipur, Kathmandu, Nepal

In Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computer Science and Information Technology

Submitted By

Suresh Thapa

(December, 2012)

Supervisor

Mr. Nawaraj Paudel

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Date :- …….

Recommendation

I hereby recommend that the dissertation prepared under my supervision by **Mr. Suresh Thapa** entitled "**A Comparative Study of Lossless Data Compression Algorithms**" be accepted as in fulfilling partial requirements for the degree of Master of Science in Computer Science and Information Technology.

Mr. Nawaraj Paudel
Asst. Professor
Central Department of Computer Science and Information Technology,
Tribhuvan University, Kritipur
(**Supervisor**)

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

We certify that we have read this dissertation work and in our opinion it is satisfactory on the scope and quality as a dissertation in the partial fulfillment for the requirement of Master of Science in Computer Science and Information Technology.

Evaluation Committee

Mr. NawarajPaudel	**Mr. NawarajPaudel**
Acting Head of Department	Central Department of Computer Science
Central Department of Computer Science	and Information Technology
and Information Technology	Tribhuvan University
Tribhuvan University	Kirtipur
Kirtipur	(Supervisor)

Date:

(External Examiner)	**(Internal Examiner)**

Acknowledgement

Let me take an opportunity to express my sincere gratitude to all the persons who supported and encouraged me to complete this thesis work entitled **"A Comparative Study of Lossless Data Compression Algorithms".**

First of all I would like to thank Tribhuvan University, Central Department of Computer Science and Information Technology for providing me this opportunity to perform this research work.

I must record my immense gratitude to my supervisor **Mr. Nawaraj Paudel** who patiently listens to many fragments of data and arguments and was able to make then decisions very stimulating. His guidance and conclusion remarks had remarkable impact on my thesis. I am greatly obliged to our Department Head, Assoc. Prof. **Dr. Tanka Nath Dhamla** for his constant support. He was the one who was always available to deal with every obstacle that I faced during my study with his insightful suggestions.

I am also highly thankful to all the teachers and staffs of CDCSIT for providing me such a broad knowledge and enlightment in two years of study period. Their motivation and support was really appreciable.

I am also indebted to all my friends who supported me during my Masters. Their cheerfulness and sense of humor would always brighten bad day; I would have never made it without their unfailing support. Special thanks to, Mr. Pravakar Ghimire, Mr. Amar Man Maharjan, Mr. Shiva Raj Panta and Mr. Rabindra Maharjan.

Last but not the least, I would like to acknowledge and appreciate the direct and indirect support of my friend Ms. Rukamanee Maharjan.

I have given my best effort to make this thesis work complete and error free but still if it contains some faults, suggestions regarding those mistakes will always be welcomed.

Suresh Thapa

December, 2012

Abstract

Data Compression is the method for minimizing the resources allocated by reducing size of the files. Data Compression is widely required in the era of information communication technology as it is useful for processing, storing and transferring data that requires lots of resources. There are lots of data compression algorithms which are available to compress files of different format. This dissertation is basically concerned with lossless data compression algorithms namely gzip and bzip2 and performance of these algorithms is analyzed and compared. The performance parameters are comparison ratio, comparison speed, saving percentage, decompression speed. For more reliability text data of different file format is considered for study. With the help of performance parameters, this dissertation is concluded by stating which algorithm performs well for text data.

Table of Contents

CHAPTER 5

List of Figures

List of Tables

Abbreviation

GIF	Graphics Interchange Format
JPG	Joint Photographic Group
KB	KiloByte
LZ	Lempel Ziv
LZH	Lempel Ziv Haruyasu
LZMA	Lempel Ziv Markov Algorithm
LZSS	Lempel Ziv Storer Szymanski
LZW	Lempel Ziv Welch
PDF	Portable Document Format
TIF	Tagged Image File
XML/ xml	eXtended Markup Language

CHAPTER 1

INTRODUCTION

1.1 General Background

Data Compression is the technique to reduce the size of particular file. Compressing file is very useful when processing, storing and transferring huge sized file which needs loads of resources. Choosing technique for data compression wisely can reduce the size of file and resources needed dramatically. Compressing data is the cost effective as it stores data relatively on small size and increase the data transfer rate. Reduction of size of file is achieved by excluding redundant patterns and by encoding the contents of file using symbols that require less storage space than was originally required. Basically data compression is taking a stream of symbols and transforming them into codes [8].

Data Compression = Modeling + Coding

The model is collection of data and rules used to process input symbols and determines which code to output and code is the produce the appropriate code. If the compression is effective, the resulting stream of codes will be smaller than original symbols.

Content of file is changed after compression to an encoded form and the file cannot be used until it is decompressed. The decompression process is the inverse of compression. It restores a file to its original form.

There are mainly two families of compressions: Lossy Compression and Lossless Compression

1.1.1 Lossy Compression

Lossy compression is the technique where to achieve effective compression result some of the original data can be discarded. This is effective on compressing graphics,

images and digital voices. For example, during compression of image file, human eye cannot detect difference between image generated from original file and image generated from decompressed file. Here data of some range which could not be detected by human eye are neglected.

1.1.2 Lossless Compression

Lossless Compression is the technique where discarding any of original data cannot be acceptable i.e. data obtained from decompressed file should be same as original data. For example, loss of data in text and data files would not be acceptable as it may contain words or numbers that are intended for further computing process.

There are many data compression algorithms that has been proposed and used. Some of main data compression techniques are Huffman Coding, Run Length Encoding, Shannon Fano Algorithm, Adaptive Huffman Encoding Algorithm, Arithmetic Encoding Algorithm, Limpel Zev Welch Algrithm, bzip2, gzip, LZMA.

Lossless data compression generally uses one of two different types of modeling techniques: statistical or dictionary based. Statistical modeling reads in and encodes a single symbol at a time using probability of appearance of that character. Dictionary based modeling uses a single code to replace strings of symbols.

1.2 Bzip2

Bzip2 compression files uses the Burrows-Wheeler Block Sorting text compression algorithm and Huffman coding. Bzip2 compression is considered better than LZ77/LZ78 based compression and approaches the performance of PPM family of statistical compression [1].

Bzip2 is flexible library for handling compressed data in the bzip2 format.

1.2.1 Burrows Wheeler Block Sorting Algorithm

Burrows-Wheeler is the block sorting algorithm that processes a block of text as a single unit. [5] explains that this algorithm transforms a string of N characters by

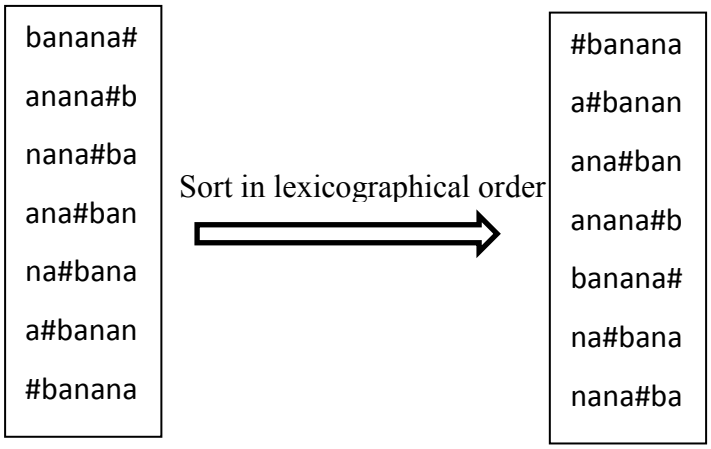
forming the n rotations (cyclic shifts) of S, sorting them lexicographically, and extracting the last character of each of the rotations. A string L is formed from these characters, where the ith character of L is the last character of the ith sorted rotation. In addition to L, the algorithm computes the index I of the original string S in sorted list of rotations. There is an efficient algorithm to compute the original string S given only L and I.

The important factor here is implementation of sorting the rotations of input block. So efficiency can be measured on how well one can sort the rotations of input block. Also, the selection of input block size plays a vital role.

Let us describe the algorithm with example as defined in [10].

We have taken string S = ‘banana#’ as example, N = 7 and the alphabet X = {‘#’, ‘a’, ‘b’, ‘n’}

C1: Sorting Rotation



C2: Finding Last Character of Rotation, take last character after rotation then

L = ‘annb#aa’, I = 4

M1: Using Move to Front Coding

Taking Y = {‘#’, ‘a’, ‘b’, ‘n’} and L = ‘annb#aa’, we compute vector R as (1 3 0 3 3 3 0)

M2: Encoding

Appling Huffman encoding to the elements of R where each element is treated as separate token to be encoded.

The output of algorithm C is pair of (OUT, I) where OUT is output of coding process and I is the value computed as in C1.

Here, the output is compressed file format and decompression is just reversed process.

W1: Decoding

Decode the stream OUT using the inverse of coding process used in M2. The result will be R as (1 3 0 3 3 3 0)

W2: Inverse Move to Front Coding

Taking Y ={'#', 'a', 'b', 'n'} initially as in algorithm M, compute L= 'annb#aa' and I=4

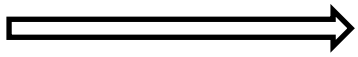
D1: First Character of Rotation

First character of rotation is computed by sorting the character L to from F.

F = '#aaabnn'

D2: Build List of Predecessor characters

Row	M
0	#banana
1	a#banan
2	ana#ban
3	anana#b
4	banana#
5	na#bana
6	nana#ba



Row	M'	T
0	a#banan	1
1	na#bana	5
2	nana#ba	6
3	banana#	4
4	#banana	0
5	ana#ban	2
6	anana#b	3

Using F and L, the first columns of M and M' respectively, we calculate a vector T that indicated the correspondence between two rows of the two matrices.

Here T as (1 5 6 4 0 2 3)

D3: Form Output S

For each i = 0,… , N-1 : S[N – 1 – i] = L[T^i [I]] where T^0[x] = x and T^{i+1}[x] = T[T^i [x]]

i	S[N – 1 – i]	L[T^i [I]]	Character
0	S[6]	L[4]	#
1	S[5]	L[0]	a
2	S[4]	L[1]	n
3	S[3]	L[5]	a
4	S[2]	L[2]	n
5	S[1]	L[6]	a
6	S[0]	L[3]	b

Final Output as S = ‘banana#’

1.3 Gzip

Gzip (also known as GNU zip) is lossless compression algorithm that compresses files. Gzip is based on an algorithm known as DEFLATE, which is also a lossless data compression algorithm. It uses both the LZ77 algorithm and Huffman Coding [12].

1.3.1 LZ77 Algorithm

LZ77 compression works by finding sequences of data that are repeated. The term ‘Sliding Windows’ is used; at any given point in the data, there is record of what character went before. For example, a 32K sliding windows means that the compressor (and decompressor) have a record of what a last 32768 (32*1024) characters were. When the next sequence of characters to be compressed is identical to one that can be found within the sliding windows, the sequence of characters is replaced by two numbers: a distance, representing how far back into the windows the sequence starts, and a length, representing the number of characters for which the sequence is identical.

For example, consider the sentence:

“spain_in_vain_with_rain_in_plain”

where the underscores “_” indicates spaces.

At first, LZ77 outcomes uncompressed characters as there is no repeated character.

spain_

The next chunk of message

in_

has occurred earlier in message and can be represented as pointer back to that earlier text, along with a length field.

spain_<3,3>

Here <3,3> means look back three characters and take three characters from that position.

After this comes

v

that has to be output uncompressed

spain_<3,3>v

then the characters “ain_” is encoded as

spain_<3,3>v<8,4>

Similarly doing this finally, the original message

“spain_in_vain_with_rain_in_plain”, has been compressed to message

spain_<3,3>v<8,4>with_r<9,4><3,3>pl<7,3>

Since in both compression algorithms’, bzip2 and gzip, huffman encoding is used and is equally important for better compression result. Huffman coding is often used as backend to these compression algorithms; Huffman coding uses a specific method for choosing the representation for each symbol, resulting in prefix-free code (that is, the bit string representing some particular symbol is not a prefix of the bit string representing any other symbol) that expresses the most common characters using shorter strings of bits than are used for less common source symbols. A simple example is used to illustrate the algorithm

Symbol	A	B	C	D	E
Count	15	7	6	6	5

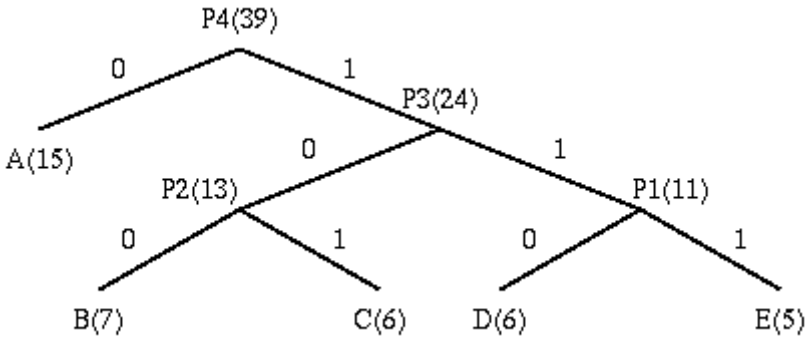


Figure 1.1 Huffman Coding

Symbol	Count	Code	Number of bits
A	15	0	15
B	7	100	21
C	6	101	18
D	6	110	18
E	5	111	15
		Total number of bits	87

CHAPTER 2

PROBLEM DEFINITION

2.1 Problem Definition

In today's business, information is the key assets and most of information is in digital format. This information need to transfer from one network to other. All the information saved in the document, regardless of document format, must be as it is while transfer. The time of data transmit also plays a vital role. In such a case, compression of data can help on reducing size. Smaller the size, less likely data integrity will be compromised in transmission. A good compression can also check data after transmission to ensure that data received is exactly with the data sent.

Many changes and improvements are seen in recent years in the field of data compression. Many data compression algorithms have been introduced where some of them work pretty well and some did not do well. Each algorithm claims that they give good compression result on the basis of either size or time. Some algorithms work fine on compression but take inconsiderable time to perform action and vice-versa. On the basis of popularity, there a few compression techniques nowadays, and most of them use the concepts of dictionary or statistics. The main idea that these algorithms use is the utilization of repetition of characters/string in the data to achieve compression. For lossless ness of data, further analysis is needed and must be verified after decompression.

It's ultimately the end user that makes choice of using which algorithm is best suits for his/her applications. Hence, confusion is created to user for choosing the right algorithm. Compression algorithm must be chosen wisely under which circumstance it is best suit to use. Data compression algorithm that is best for one format of file is not necessarily be best for other format. If time has to be taken under consideration than response time for compression must be evaluated.

2.2 Literature Review and Related Work

Various kinds of approaches are there for lossless data compression. Each approach tries to minimize the size of data as possible as it can. The basis principal of these approaches is almost same, eliminating the redundant patterns and coding the contents. But the only concern is how these approaches work for elimination redundant patterns. Algorithms like Run Length Coding, Shannon Coding, Huffman Coding, Adaptive Huffman Coding and Arithmetic Coding use statistical compression technique. Basically LZ Family use dictionary based compression technique. LZ77 algorithm uses the concept of sliding windows.

According to official website of bzip2 [1], bzip2 compresses the files to within 10% to 15% of the best available techniques. Bzip2 compression used the burrows-wheeler block sorting text compression algorithms and Huffman coding. [10] concluded that burrows-wheeler block sorting compression algorithm achieves compression comparable with good statistically modeler and is closer in speed to coders based on algorithms of LZ. Decompression is faster than compression like LZ.

Gzip (GUN zip) is based on DEFLATE algorithm, which is a combination of Lempel-Zip (LZ77) and Huffman Encoding [4]. According to [5] DEFLATE algorithm can be efficiency comparable to the best available general purpose compression methods.

Comparison of different compression algorithm on text data has been done in [15]. Here the method used for the compression is the standard methodology but the compression is carried out for only fundamental algorithms like Huffman Encoding, The Shannon Fano Algorithm, Arithmetic Encoding, LZW Algorithm. Since the methodology adapted and measuring compression done is under standardization but it lacks the comparison of today's popular compression algorithms LZ77, bzip and gzip. Here compression algorithms are tested on ten text files of different size and different content. According to study done on [15] for compression algorithm like Run Length, LZW, Adaptive Huffman, Huffman Encoding and Shannon Fano algorithm, it shows that considering the compression time, decompression time and saving percentages, Shannon Fano is considered as most effective algorithm but Shannon Fano algorithm

is quite a low compared to the Huffman Encoding algorithm. Huffman Encoding show similar performances except in the compression time.

Author of [14] has done a comparison of various Statistical compression technique (Run Length Encoding, Shannon Fano coding, Huffman coding, Adaptive Huffman coding and Arithmetic coding) and LZ family algorithm. Under study of statistical compression technique, it shows little different result than [15]. Here, compression ratio obtained by Huffman coding algorithm is better compared to Shannon Fano coding algorithm. But on overall, arithmetic coding shows one of best result on the basis of compression ratio. Under the study of LZ77 family algorithm, [14] concluded that LZB outperforms LZ77, LZSS and LZH to show marked compression amongst the LZ77 family.

[9] that focused on prominent data compression algorithms on various file format particularly .DOC, .TXT, .BMP, .TIF, .GIF and .JPG files. Studied has been carried out for Run Length Encoding, Huffman Coding, Arithmetic Coding, LZ77 Encoding and LZW Coding techniques. Through the result obtained using the algorithm, LZW and Huffman Coding has given nearly result with compression of document and text file. LZW works on replacing string of characters with single code whereas Huffman works by representing individual characters by bit sequences.

The importance of data compression in business perspective is defined in [11]. [11] gives the idea of how data compression increases the efficiencies and decreases the cost of storing and transferring important business information.

2.3 Methodology

In this dissertation, study will focus on evaluating the effectiveness of compression algorithms through Compression Ratio, Compression Time, Saving Percentage as parameters using different file sizes.

Furthermore, to evaluate parameters like compression ratio, compression speed, saving percentage etc., text files of various sizes will be processed through

implementation of code of different compression algorithms and parameters like compressed file size, compression time, decompression time will be recorded [15].

Compression Ratio is the ratio between the size of the compression file and the size of source file.

$$compression\ ratio = \frac{size\ after\ compression}{size\ before\ compression}$$

Thus a representation that compresses a 10MB file to 2MB has a compression ratio of 2/10 = 0.2, often notated as an explicit ratio, 1:5. This formulation applies equally for compression, where the uncompressed size is that of the original; and for decompression, where the uncompressed size is that of the reproduction.

Compression Factor is the inverse of the compression ratio, i.e. ratio between the size of the source file and the size of compressed file.

$$compression\ factor = \frac{size\ befor\ compression}{size\ after\ compression}$$

In this case, values greater than 1 indicate compression and values less than 1 expansion. Hence, bigger the compression factor, the better the compression.

Saving Percentage calculate the shrinkage of the source file as percentage.

$$saving\ percentage(\%) = \frac{size\ before\ compression - size\ after\ compression}{size\ before\ compression}$$

Above defined methods evaluate the effectiveness of compression algorithm using file sizes. Compression time and decompression time other methods to evaluate the performance of compression algorithms which will be used to measure effectiveness.

CHAPTER 3

IMPLEMENTATION

3.1 Implementation

Data that are used on analysis is primarily based on common windows desktop files. To understand the data compression, it is import to know the size, content and format of the file that is processed for compression. Here, the test data are files of different sizes and different content with different file format.

3.1.1 Data Collection

To implement the compression algorithms, namely bzip2 and gzip, different format text files are randomly collected. The collected files are of various file size. Some of collected files contain fake data and some contain actual user's text data. The file format that are collected for test data are Microsoft Word Document (doc) format file, Adobe Portable Document Format (pdf) File, Extended Markup Language (xml) Format File and Database Log file. The files are file with normal English language, computer programs, E-books which are also in normal English language, database log file generated with fake as well as with real data. Computers programs and xml files have more repeating set of words than that of E-books and normal text files. Collected files with file format and original file size are presented in Table 3.1.

File Name	File Format	Original File Size in KB
D60_en	Pdf	11816
data compression	Pdf	6015
Data Compression Book	Pdf	1689
GP-zip Family	Pdf	2113
isoiec 14496-3	Pdf	7538
Patel Thesis	Pdf	2934
PhD_Johns	Pdf	5984
Text_Mining_Infrastructure	Pdf	686
The Text Mining HandBook	Pdf	8108
Thesaurus	Pdf	3424
AscolCampus	Mdf	2048
DWPDatabase	Mdf	2048

DWPDatabase_log	Ldf	1024
MvcMusicStore	Mdf	2304
Pasa	Mdf	3072
PasaDB_backup	File	2581
pulse_backup	Bak	4309
Pulse_log	Ldf	1024
SutekiShop	Mdf	2304
GMAT	Doc	6334
25308-b00	Doc	3625
Compression	Docx	653
CustomerInformation	Xlsx	5115
Eat Pray Love	Doc	916
First Draft Report	Doc	842
Matthew MacDonald	Doc	4398
Matthew MacDonald	Docx	1305
Thesis_final_rukamanee	Doc	367
AdobeDreamweaver	Xml	6442
AdobeFireworks	Xml	6505
Construction	Xml	377
FontList	Xml	608
GlobalInstallOrder	Xml	1927
HealthcarePro	Xml	438
PropertyManagement	Xml	459
System.Net.Http	Xml	153
Calculation	Php	162
jquery.ui.theme	Css	19
Pasadb	Sql	396
SharpZipLib	Chm	1308

Table 3.1 Data For Implementation

3.1.2 Performance Evaluation

With these sample data and the result obtained after applying compression algorithms, both bzip2 and gzip, compression ratio, compression factor, saving percentile and compression time is calculated. Respective decompression algorithms are used for decompression the compressed data and decompression time is calculated. The compression results and the calculation results are tabulated. Source codes for both algorithms are opened source [1] and [4]. These source codes for compression algorithm are written in C# language. For the studied purpose the source codes are modified as according to need.

The performance measurements factors discussed above are based on file sizes and time. The performance measurements could be more than mentioned as performances could be based on different approaches. So, all of them cannot be applied for all the selected algorithms. Additionally, the quality difference between the original and decompressed file is not considered as a performance factor as the selected algorithms are lossless. The performances of the algorithms depend on the size of the source file and the organization of symbols in the source file. Therefore, a set of files including different types of texts such as English phrases, source codes, user manuals, etc and different file sizes are used as source files. Graphs are drawn in order to identify the relationship between the file sizes between original and compressed one, the compression and decompression time and other performance factors.

The performances of the selected algorithms may vary according to the measurements. Therefore, all these factors are considered for comparison in order to identify the best solution. An algorithm which gives an acceptable saving percentage within a reasonable time period is considered as the best algorithm.

CHAPTER 4

TESTING AND ANALYSIS

4.1 Testing Data

Two lossless algorithms, namely bzip2 and gzip, are tested for forty text files with different file sizes and different content.

4.1.1 Testing Data and Result using Bzip2 Algorithms

File Name	File Format	Original File Size in KB	Compres-sed File Size in KB	Compressio n Ratio	Compression Factor
D60_en	Pdf	11816	9220	78.02979012	128.1561822
data compression	Pdf	6015	5553	92.319202	108.3198271
Data Compression Book	Pdf	1689	1500	88.80994671	112.6
GP-zip Family	Pdf	2113	1612	76.28963559	131.0794045
isoiec 14496-3	Pdf	7538	7164	95.03847174	105.2205472
Patel Thesis	Pdf	2934	1612	54.94205862	182.0099256
PhD_Johns	Pdf	5984	4408	73.6631016	135.753176
Text_Mining_In frastructure	Pdf	686	537	78.27988338	127.7467412
The Text Mining HandBook	Pdf	8108	4487	55.34040454	180.6997994
Thesaurus	Pdf	3424	2559	74.73714953	133.8022665
AscolCampus	Mdf	2048	82	4.00390625	2497.560976
DWPDatabase	Mdf	2048	101	4.931640625	2027.722772
DWPDatabase_l og	Ldf	1024	57	5.56640625	1796.491228
MvcMusicStore	Mdf	2304	91	3.949652778	2531.868132
Pasa	Mdf	3072	190	6.184895833	1616.842105
PasaDB_backup	File	2581	140	5.424254165	1843.571429
pulse_backup	Bak	4309	339	7.867254583	1271.091445
Pulse_log	Ldf	1024	51	4.98046875	2007.843137
SutekiShop	Mdf	2304	103	4.470486111	2236.893204
GMAT	Doc	6334	1382	21.81875592	458.3212735

25308-b00	Doc	3625	876	24.16551724	413.8127854
compression	Docx	653	632	96.78407351	103.3227848
CustomerInformation	Xlsx	5115	659	12.88367546	776.1760243
Eat Pray Love	Doc	916	267	29.14847162	343.071161
First Draft Report	Doc	842	132	15.67695962	637.8787879
Matthew MacDonald	Doc	4398	922	20.96407458	477.0065076
Matthew MacDonald	Docx	1305	1309	100.3065134	99.69442322
Thesis_final_rukamanee	Doc	367	86	23.43324251	426.744186
AdobeDreamweaver	Xml	6442	1089	16.90468799	591.5518825
AdobeFireworks	Xml	6505	1132	17.40199846	574.6466431
Construction	Xml	377	16	4.24403183	2356.25
FontList	Xml	608	66	10.85526316	921.2121212
GlobalInstallOrder	Xml	1927	80	4.151530877	2408.75
HealthcarePro	Xml	438	17	3.881278539	2576.470588
PropertyManagement	Xml	459	18	3.921568627	2550
System.Net.Http	Xml	153	10	6.535947712	1530
Calculation	Php	162	20	12.34567901	810
jquery.ui.theme	Css	19	3	15.78947368	633.3333333
Pasadb	Sql	396	26	6.565656566	1523.076923
SharpZipLib	Chm	1308	1277	97.62996942	102.4275646

Table 4.1 Compression Result using bzip2 Algorithm

File Name	**File Format**	**Saving Percentage**	**Compression Time in millisecond**	**Decompression Time in millisecond**
D60_en	pdf	21.97020988	14144	7084
data compression	pdf	7.680798005	7312	3751
Data Compression Book	pdf	11.19005329	2168	1191
GP-zip Family	pdf	23.71036441	2572	1163
isoiec 14496-3	pdf	4.961528257	10038	5136
Patel Thesis	pdf	45.05794138	3040	1268
PhD_Johns	pdf	26.3368984	9030	3288
Text_Mining_Infrastructure	pdf	21.72011662	870	389
The Text Mining HandBook	pdf	44.65959546	18645	3501

Thesaurus	pdf	25.26285047	6283	1921
AscolCampus	mdf	95.99609375	1910	284
DWPDatabase	mdf	95.06835938	1274	291
DWPDatabase_log	ldf	94.43359375	522	163
MvcMusicStore	mdf	96.05034722	2109	355
Pasa	mdf	93.81510417	1297	458
PasaDB_backup	file	94.57574583	1030	370
pulse_backup	bak	92.13274542	2071	742
Pulse_log	ldf	95.01953125	832	177
SutekiShop	mdf	95.52951389	1310	316
GMAT	doc	78.18124408	8121	1278
25308-b00	doc	75.83448276	3000	939
Compression	docx	3.215926493	875	560
CustomerInformation	xlsx	87.11632454	4876	1045
Eat Pray Love	doc	70.85152838	1071	279
First Draft Report	doc	84.32304038	662	153
Matthew MacDonald	doc	79.03592542	5643	877
Matthew MacDonald	docx	-0.30651341	1702	1148
Thesis_final_rukamanee	doc	76.56675749	208	74
AdobeDreamweaver	xml	83.09531201	3694	1719
AdobeFireworks	xml	82.59800154	3802	1593
Construction	xml	95.75596817	438	77
FontList	xml	89.14473684	578	138
GlobalInstallOrder	xml	95.84846912	2110	313
HealthcarePro	xml	96.11872146	565	88
PropertyManagement	xml	96.07843137	528	87
System.Net.Http	xml	93.46405229	99	31
Calculation	php	87.65432099	127	46
jquery.ui.theme	css	84.21052632	31	5
Pasadb	sql	93.43434343	357	56
SharpZipLib	chm	2.370030581	1715	1214

Table 4.2 Compression Result using bzip2 Algorithm

4.1.2 Testing Data and Result using Gzip Algorithm

File Name	File Format	Original File Size in KB	Compres-sed File Size in KB	Compressio n Ratio	Compression Factor
D60_en	Pdf	11816	9175	77.64895058	128.7847411
data compression	Pdf	6015	5641	93.78221114	106.6300301
Data Compression Book	Pdf	1689	1499	88.75074008	112.6751167

GP-zip Family	Pdf	2113	1606	76.00567913	131.5691158
isoiec 14496-3	Pdf	7538	7255	96.24568851	103.9007581
Patel Thesis	Pdf	2934	1972	67.21199727	148.7829615
PhD_Johns	Pdf	5984	5098	85.19385027	117.3793645
Text_Mining_Infrastructure	Pdf	686	544	79.30029155	126.1029412
The Text Mining HandBook	Pdf	8108	4518	55.72274297	179.459938
Thesaurus	Pdf	3424	2678	78.21261682	127.8566094
AscolCampus	Mdf	2048	107	5.224609375	1914.018692
DWPDatabase	Mdf	2048	134	6.54296875	1528.358209
DWPDatabase_log	Ldf	1024	85	8.30078125	1204.705882
MvcMusicStore	Mdf	2304	115	4.991319444	2003.478261
Pasa	Mdf	3072	247	8.040364583	1243.724696
PasaDB_backup	File	2581	196	7.593955831	1316.836735
pulse_backup	Bak	4309	434	10.07194245	992.8571429
Pulse_log	Ldf	1024	85	8.30078125	1204.705882
SutekiShop	Mdf	2304	141	6.119791667	1634.042553
GMAT	Doc	6334	1691	26.69718977	374.5712596
25308-b00	Doc	3625	1061	29.26896552	341.6588124
compression	Docx	653	630	96.47779479	103.6507937
CustomerInformation	Xlsx	5115	1014	19.82404692	504.4378698
Eat Pray Love	Doc	916	325	35.48034934	281.8461538
First Draft Report	Doc	842	157	18.64608076	536.3057325
Matthew MacDonald	Doc	4398	1160	26.37562528	379.137931
Matthew MacDonald	Docx	1305	1303	99.8467433	100.1534919
Thesis_final_rukamanee	Doc	367	93	25.34059946	394.6236559
AdobeDreamweaver	Xml	6442	1610	24.99223844	400.1242236
AdobeFireworks	Xml	6505	1641	25.22674865	396.4046313
Construction	Xml	377	21	5.570291777	1795.238095
FontList	Xml	608	69	11.34868421	881.1594203
GlobalInstallOrder	Xml	1927	88	4.566683965	2189.772727
HealthcarePro	Xml	438	23	5.251141553	1904.347826
PropertyManagement	Xml	459	24	5.22875817	1912.5
System.Net.Http	Xml	153	12	7.843137255	1275

Calculation	Php	162	23	14.19753086	704.3478261
jquery.ui.theme	Css	19	4	21.05263158	475
Pasadb	Sql	396	36	9.090909091	1100
SharpZipLib	Chm	1308	1272	97.24770642	102.8301887

Table 4.3 Compression Result using gzip Algorithm

File Name	**File Format**	**Saving Percentage**	**Compression Time in millisecond**	**Decompression Time in millisecond**
D60_en	pdf	22.35104942	2835	645
data compression	pdf	6.217788861	1722	487
Data Compression Book	pdf	11.24925992	495	133
GP-zip Family	pdf	23.99432087	507	133
isoiec 14496-3	pdf	3.754311488	2066	776
Patel Thesis	pdf	32.78800273	677	188
PhD_Johns	pdf	14.80614973	1556	370
Text_Mining_Infrastructure	pdf	20.69970845	172	51
The Text Mining HandBook	pdf	44.27725703	1679	449
Thesaurus	pdf	21.78738318	742	104
AscolCampus	mdf	94.77539063	483	72
DWPDatabase	mdf	93.45703125	398	111
DWPDatabase_log	ldf	91.69921875	154	29
MvcMusicStore	mdf	95.00868056	468	70
Pasa	mdf	91.95963542	615	98
PasaDB_backup	file	92.40604417	508	74
pulse_backup	bak	89.92805755	821	134
Pulse_log	ldf	91.69921875	167	27
SutekiShop	mdf	93.88020833	494	74
GMAT	doc	73.30281023	1362	307
25308-b00	doc	70.73103448	697	163
Compression	docx	3.522205207	132	34
CustomerInformation	xlsx	80.17595308	671	221
Eat Pray Love	doc	64.51965066	234	55
First Draft Report	doc	81.35391924	133	28
Matthew MacDonald	doc	73.62437472	952	220
Matthew MacDonald	docx	0.153256705	254	41
Thesis_final_rukamanee	doc	74.65940054	73	18
AdobeDreamweaver	xml	75.00776156	1207	319
AdobeFireworks	xml	74.77325135	1183	323
Construction	xml	94.42970822	34	9
FontList	xml	88.65131579	61	19
GlobalInstallOrder	xml	95.43331604	158	38
HealthcarePro	xml	94.74885845	37	12

PropertyManagement	xml	94.77124183	38	12
System.Net.Http	xml	92.15686275	15	5
Calculation	php	85.80246914	42	71
jquery.ui.theme	css	78.94736842	4	5
Pasadb	sql	90.90909091	58	10
SharpZipLib	chm	2.752293578	266	36

Table 4.4 Compression Result using gzip Algorithm

4.2 Analysis and Interpretation

On the application of compression algorithms on forty test data, following result are obtained for the compression ratio.

4.2.1 Compression Ratio

Table 4.5 represents the average compression ratio observed for bzip2 and gzip algorithms.

Compression Algorithm	**Average Compression Ratio**
Bzip2	34.00592447
Gzip	36.8208785

Table 4.5 Average Compression Ratio

On the basis of data recorded on Table 4.5, it is clearly analyzed that bzip2 algorithm gives excellent compression ratio in comparison to gzip algorithm.

The graphical presentation of observed compression ratio for entire test data under study is given in Fig 4.1. It can be observed that compression ratio is independent on file size i.e. file size does not matter in increase or decrease of compression ratio.

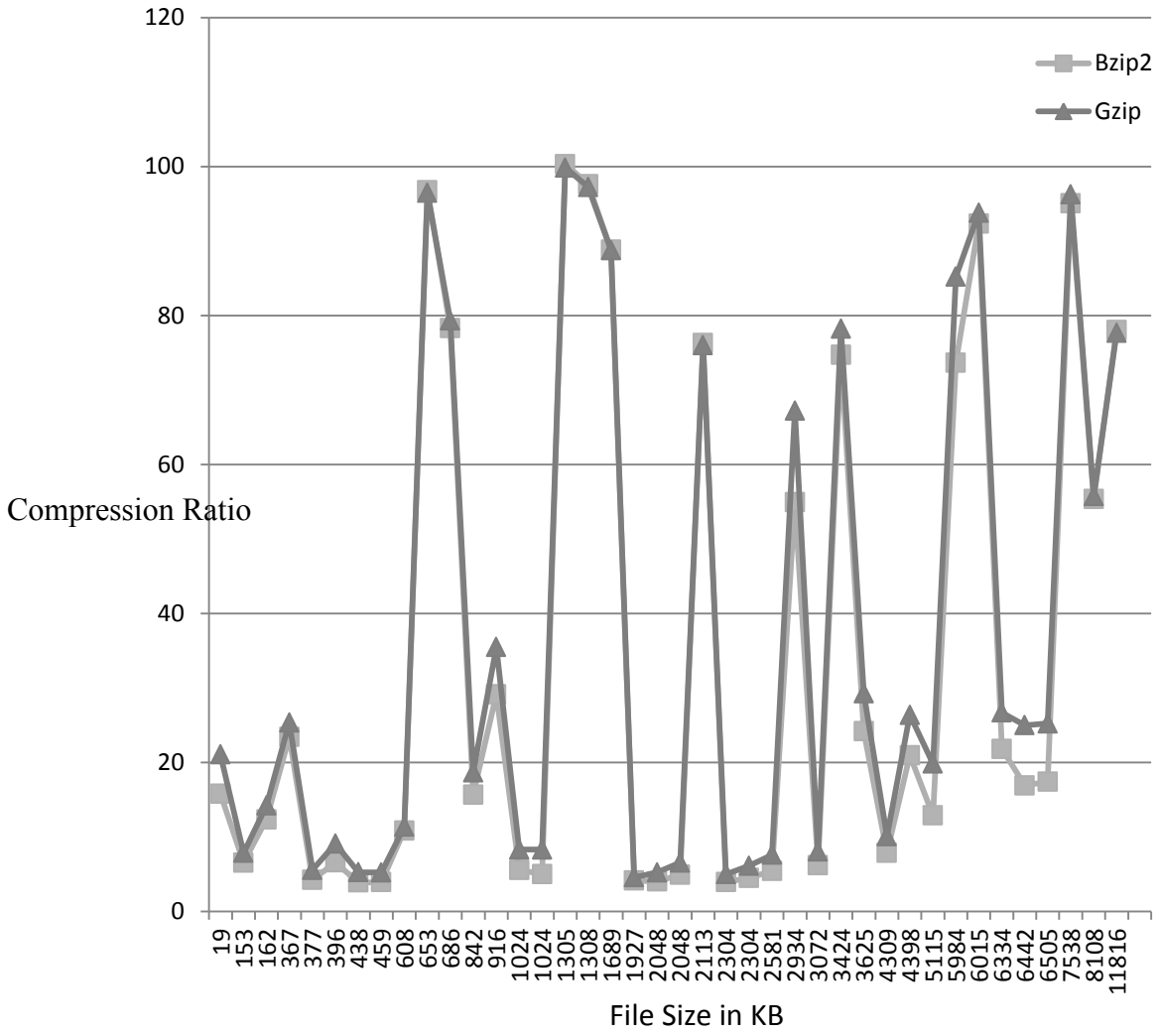


Fig 4.1 File Size Vs Compression Ratio

4.2.2 Compression Time

Table 4.6 represents the average compression time observed for bzip2 and gzip algorithms.

Compression Algorithm	Average Compression Time (millisecond)
Bzip2	3166.475
Gzip	604.25

Table 4.6 Average Compression Time

On the basis of data recorded on Table 4.5, gzip algorithm can be considered to be excellent as it can compress files much faster than bzip2.

The graphical representation of observed compression time for entire test data under study is given in Fig 4.2. It can be seen that the compression time increases as increase in size of file and vice versa. Some exception has been occurred while compressing some files. In some cases bzip2 has shown unusual behavior. For example for file "The Text Mining HandBook" has taken inconsiderable higher compression time. In other cases, it has shown almost same kind of behavior.

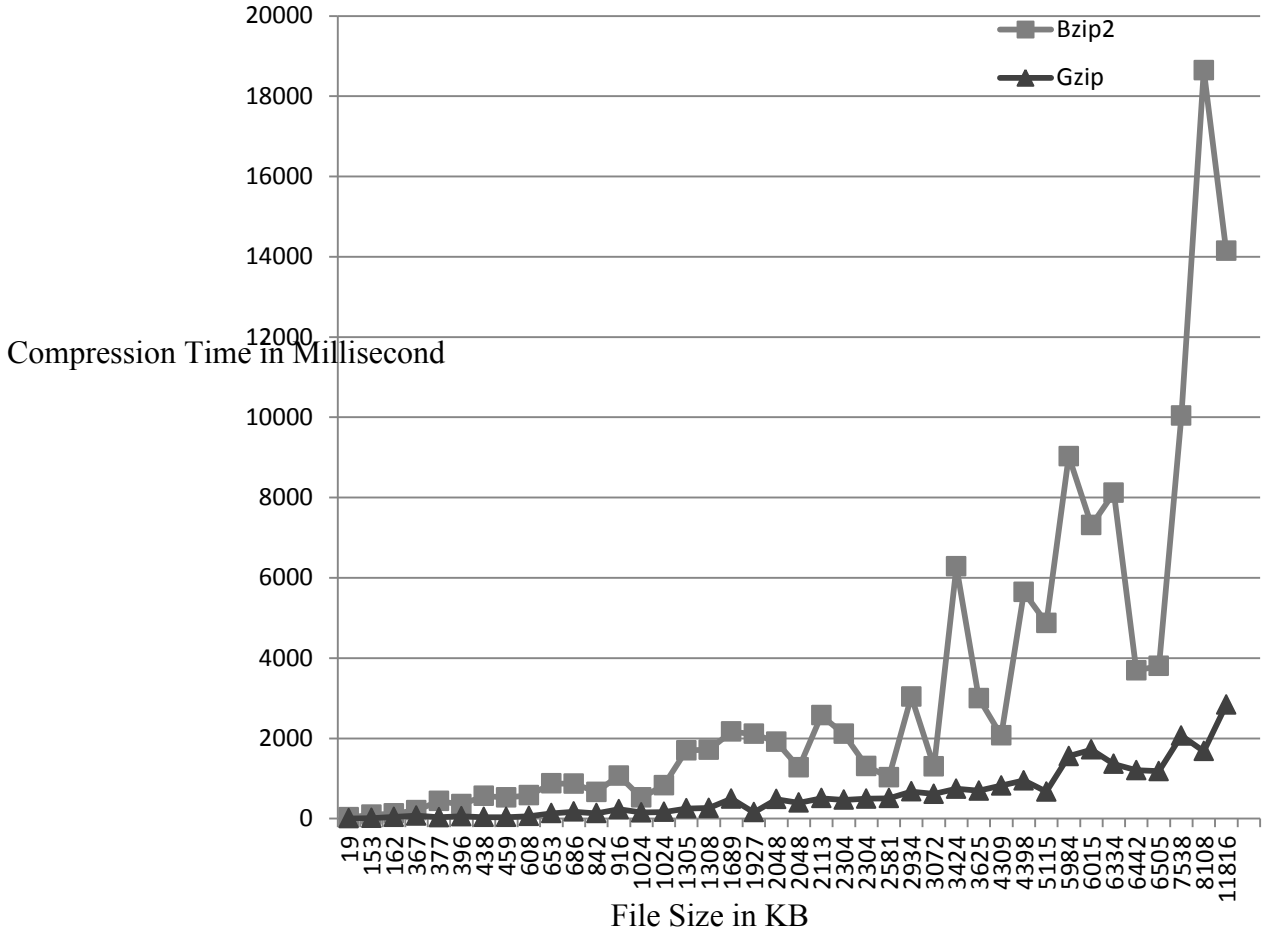


Fig 4.2 File Size Vs Compression Time

4.2.3 Saving Percentage

Average saving percentage, using bzip2 and gzip algorithm, for result obtained from 40 test data is calculated and presented on Table 4.7. From calculation, it can be said that bzip2 algorithm more saving percent than gzip.

Compression Algorithm	Average Saving Percentage
Bzip2	65.99407553
Gzip	63.1791215

Table 4.7 Average Saving Percent

The graphical representation of observed saving percentage for entire test data under study is shown in Fig 4.3.

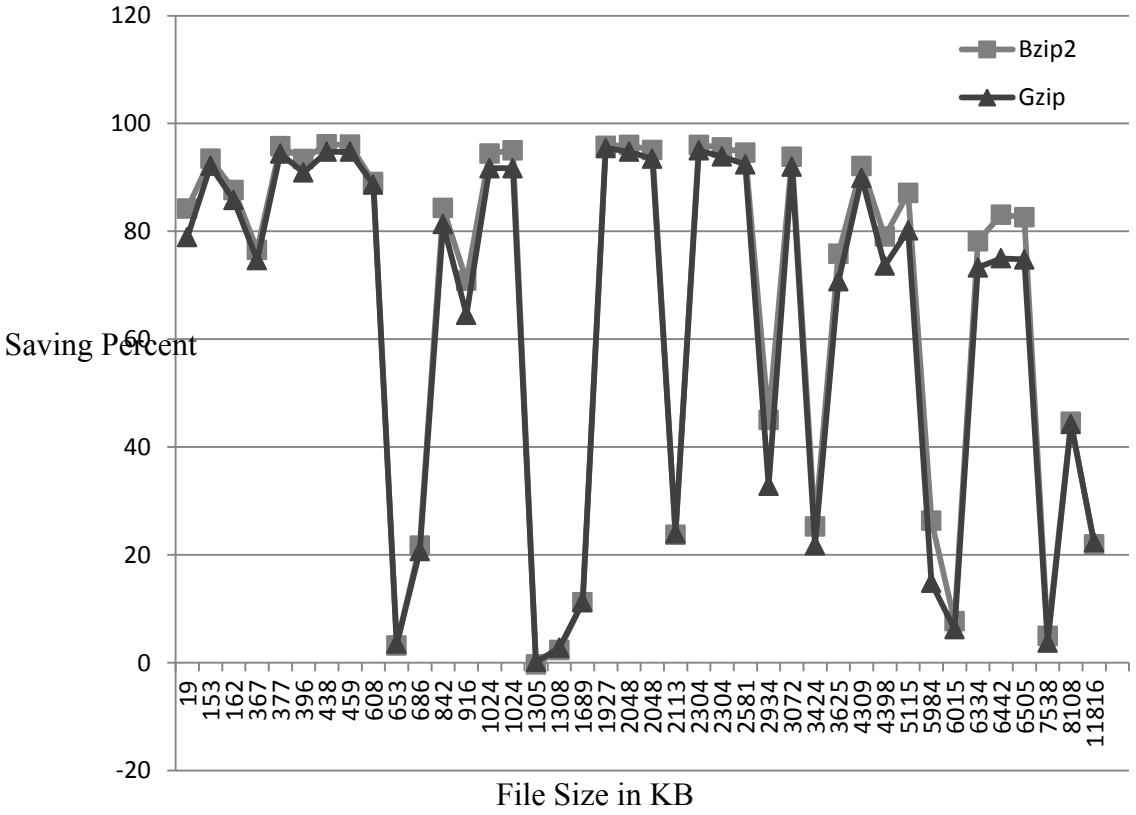


Fig 4.3 File Size Vs Saving Percent

When studied the saving percentage of test data, as whole bzip2 have better saving percent than gzip. But if the bzip2 carried out for data like ".docx" format file, it has negative saving percent value i.e. size of compressed file is increase from original file size. Since .docx is itself a compressed file and header information is needed. Hence, may be due to the added information while compression, there is increase in file size.

4.2.4 Decompression Time

Table 4.8 represents the average time taken for decompressing the compressed file using respective algorithms, bzip2 and gzip.

Compression Algorithm	Average Compressed File Size in KB	Decompression Time in millisecond
Bzip2	1245.625	1089.2
Gzip	1344.675	149.275

Table 4.8 Decompression Time

In both cases, decompression time is less as compare to compression time. On observing the decompression time of gzip and bzip3, gzip decompressed the file more quickly than bzip2.

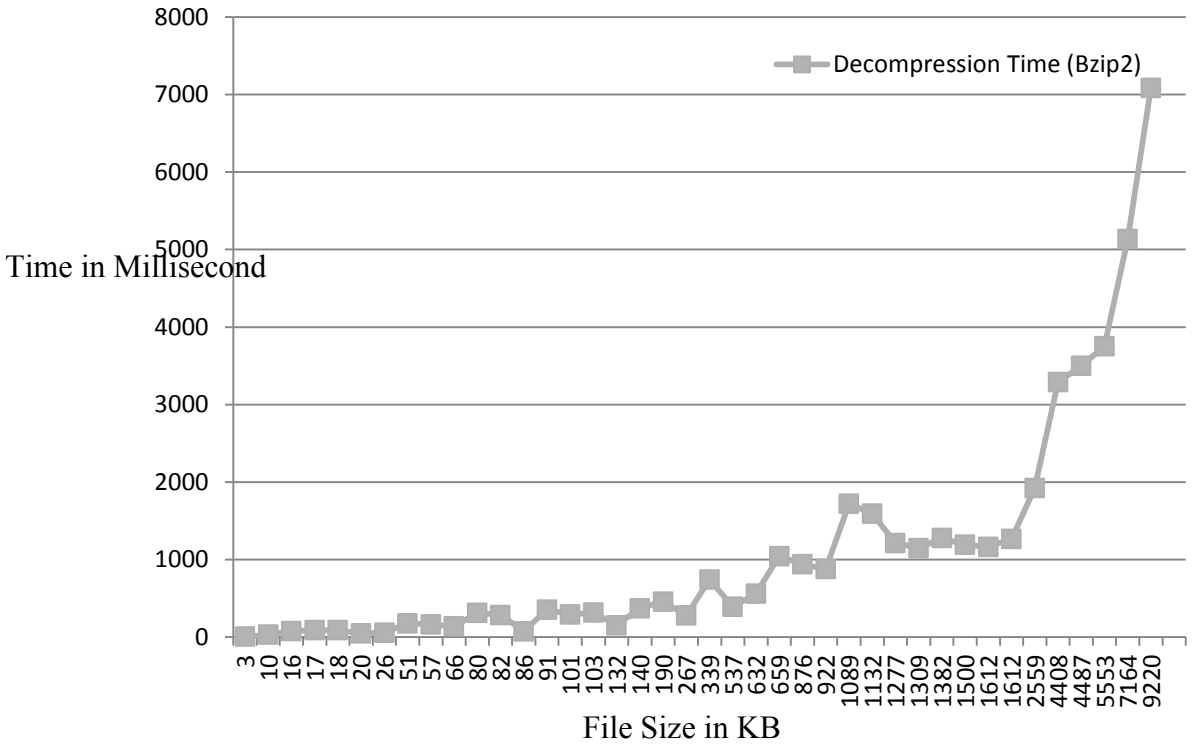


Fig 4.4 Decompression Time for Compressed File (bzip2)

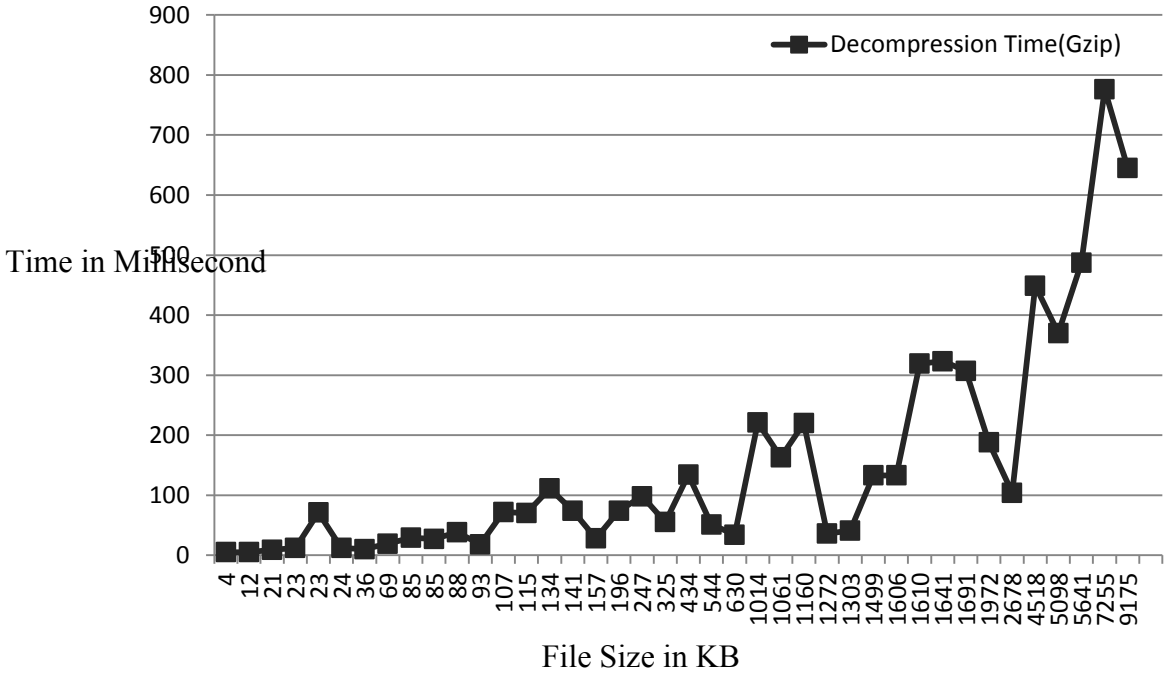


Fig 4.5 Decompression Time for Compressed File (gzip)

4.2.5 Overall

Table 4.9 represents the average file size (total size/no. of file) of 40 test data that was taken under study and their respective average size of compressed file observed for bzip2 and gzip algorithms.

Compression Algorithm	**Average File Size (KB)**	**Average Compressed File Size (KB)**
Bzip2	2834.25	1245.625
Gzip	2834.25	1344.675

Table 4.9 Average Compressed File Size

Fig 4.6 shows the graphical representation of compressed file using bzip2 and gzip in compare with original file size considered under study.

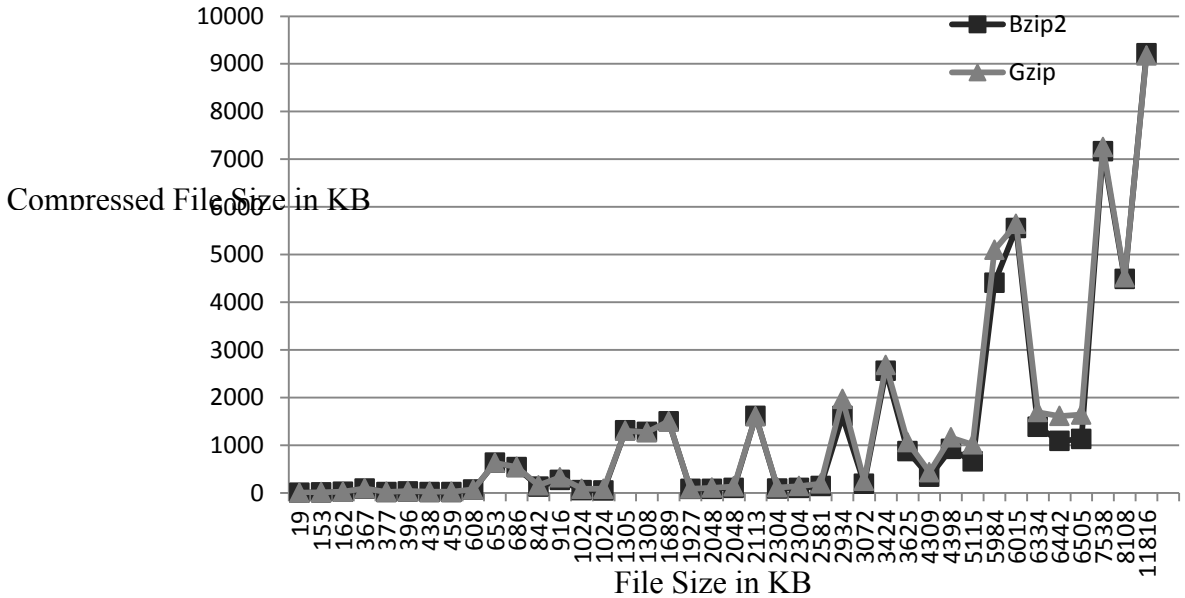


Fig 4.6 File Size Vs Compressed File Size

It is clear that the compression is only the substituting of repeated pattern by symbol. Since, xml and program files like css, php, sql have more repeating set of characters so such files are highly compressed in compare to the files like doc, docx, xls and pdf which contain less repeating set of characters. Database files, mdf, ldf and bak (used in sql management), are also highly compressed.

4.3 Verification and Validation

Any loss of data after decompressing is not acceptable in context of lossless compression. These data could be used in future for further analysis. Accuracy of the losslessness of text file depends on decompressing compressed file using respective algorithms and analysis contents. After decompressing, the file size and the file content obtained must have same size as original file and same content as original file. On analysis files after decompressing forty compressed file using respective algorithm for decompression, it is found that the size of file obtained is same as original file and there is no change on contents i.e the contents of original file and file obtained after decompression are same. Hence, it can be said that the losslessness of files has been verified. Therefore, accuracy = 100%

CHAPTER 5

CONCLUSION AND FURTHER STUDY

5.1 Conclusion

There is the variation of performance of the selected algorithms according to the measurement, while one algorithm gives higher saving percentage, processing time needed may be higher. Therefore, all these factors are considered for comparison in order to get the best solutions. An algorithm is said to be best one if it gives an acceptable saving percentage within reasonable time period for compression and decompression.

Bzip2 has compression ratio of nearly 38 whereas the compression ratio of gzip is 34. Bzip2 has compressed the file on average by 66% and gzip does by 63%. But gzip compression the file 80% faster than bzip2 as well as decompression speed is also 86% faster than bzip2. Considering all these factors, gzip could be said as better algorithm in compared to bzip2. Besides, as the needed either of algorithm could be an option.

If very fast compression is needed, gzip is the clear winner. Bzip2 have more saving percentage than gzip and hence bzip2 is getting popular beside the slower compression than gzip. Gzip is again winner in case of speed for decompression. Bzip2 is slower in this case also.

Gzip is very fast and bzip2 has notably better compression ratio. To choose which algorithm is best suited, it depends on intended application. If speed is matter, then gzip is the best options whereas if compression ration than bzip2.

5.2 Further Study

The performed work can be extended to evaluate the performance of lossy compression algorithm which will be useful in multimedia files compression like image and video.

The performed work can also be extended for parallel data compression of lossless compression algorithm which works on multiple processors using pthread. Using parallel data compression may provide more compressed file in high speed rate.

Further, there are lots of other lossless algorithms available and some are better as well. Hence, clear comparative study for new lossless algorithms and old popular with newer version could be made.

BIBLIOGRAPHY

[1] Bzip2 specification, http://www.bzip.org

[2] Christina Zeeh, “The Limpel Zip Algorithm”, Seminar – Famous Algorithms, Jan 16, 2003

[3] Guy E. Blelloch, “Introduction to Data Compression”, Computer Science Department, Carnegie Mellon University, Sept 25, 2010

[4] Gzip specification, http://www.gzip.org

[5] Haroon Altarawneh and Mohammad Altarawneh, “Data Compression Techniques on Text Files: A Comparison Study”, Albalqa Applied University, Salt, Jordan, International Journal of Computer Applications, Vol. 26 No. 5, July 2011

[6] Jacob Ziv and Abraham Lempel, “A Universal Algorithm for Sequential Data Compression”, IEEE Transactions on Information Theory, Vol. 23, No. 3, May 1997

[7] Mamata Sharma, “Compression using Huffman Coding”, S.L. Bawa D.A.V. College, ICJCSNS, Vol. 10, No. 5, May 2010

[8] Mark Nelson and Jean-Loup Gaily, “The Data Compression Book Second Edition”, ISBN: 1558514341

[9] Mohammad Al-laham and Ibrahiem M.M. El Emary, “Comparative Study Between Various Algorithm of Data Compression Techniques”, WCECS, SF, USA, 2007

[10] M. Burrows and D.J. Wheeler, “A Block-sorting Lossless Data Compression Algorithm”, SRC Research Report 124, Digital System Research Center, Palo Alto, CA, 1994

[11] PKWARE, “Data Compression Benchmark and ROI analysis”, Technical White Paper, 2008

[12] P. Deutsch, “DEFLATE Compressed Data Format Specification version 1.3”, Network Working Group, Aladdin Enterprises, May 1996

[13] Sebastian Deorowicz, “Universal Lossless Data Compression Algorithms”, Ph.D. Thesis, Institute of Computer Science, Sileasion University of Technology, Gliwice, 2003

[14] Senthil Shanmugasundaram and Robert Lourdusamy, “A Comparative Study of Text Compression Algorithm”, International Journal of Wisdom Based Computing, Vol.1 (3), December 2011

[15] S.R. Kodituwakku, U.S. Amarasinghe, “Compression of Lossless Data Compression for Text Data”, Indian Journal of Computer Science and Engineering Vol 1 No 4416-425

APPENDICES

Appendix A

Main Program for Bzip2

```
namespace Bzip2
{
   public static class BZip2
   {
      public static void Decompress(Stream inStream, Stream outStream, bool
isStreamOwner)
      {
         if (inStream == null || outStream == null)
         {
            throw new Exception("Null Stream");
         }
         try
         {
            using (BZip2InputStream bzipInput = new BZip2InputStream(inStream))
            {
               bzipInput.IsStreamOwner = isStreamOwner;
               StreamUtils.Copy(bzipInput, outStream, new byte[4096]);
            }
         }
         finally
         {
            if (isStreamOwner)
            {
               outStream.Close();
            }
         }
      }

      public static void Compress(Stream inStream, Stream outStream, bool
isStreamOwner, int level)
      {
         if (inStream == null || outStream == null)
         {
            throw new Exception("Null Stream");
         }

         try
         {
```

```
            using (BZip2OutputStream bzipOutput = new
BZip2OutputStream(outStream, level))
            {
               bzipOutput.IsStreamOwner = isStreamOwner;
               StreamUtils.Copy(inStream, bzipOutput, new byte[4096]);
            }
         }
         finally
         {
            if (isStreamOwner)
            {
               inStream.Close();
            }
         }
      }
   }
}

/*************************************************************/

      void MoveToFrontCodeAndSend()
      {
         BsPutIntVS(24, origPtr);
         GenerateMTFValues();
         SendMTFValues();
      }

void GenerateMTFValues()
      {
         char[] yy = new char[256];
         int i, j;
         char tmp;
         char tmp2;
         int zPend;
         int wr;
         int EOB;

         MakeMaps();
         EOB = nInUse + 1;
         for (i = 0; i <= EOB; i++)
         {
            mtfFreq[i] = 0;
         }
         wr = 0;
         zPend = 0;
         for (i = 0; i < nInUse; i++)
         {
```

```
    yy[i] = (char)i;
}


for (i = 0; i <= last; i++)
{
    char ll_i;

    ll_i = unseqToSeq[block[zptr[i]]];

    j = 0;
    tmp = yy[j];
    while (ll_i != tmp)
    {
        j++;
        tmp2 = tmp;
        tmp = yy[j];
        yy[j] = tmp2;
    }
    yy[0] = tmp;

    if (j == 0)
    {
        zPend++;
    }
    else
    {
        if (zPend > 0)
        {
            zPend--;
            while (true)
            {
                switch (zPend % 2)
                {
                    case 0:
                        szptr[wr] = (short)BZip2Constants.RunA;
                        wr++;
                        mtfFreq[BZip2Constants.RunA]++;
                        break;
                    case 1:
                        szptr[wr] = (short)BZip2Constants.RunB;
                        wr++;
                        mtfFreq[BZip2Constants.RunB]++;
                        break;
                }
                if (zPend < 2)
                {
```

```
                        break;
                    }
                    zPend = (zPend - 2) / 2;
                }
                zPend = 0;
            }
            szptr[wr] = (short)(j + 1);
            wr++;
            mtfFreq[j + 1]++;
        }
    }

    if (zPend > 0)
    {
        zPend--;
        while (true)
        {
            switch (zPend % 2)
            {
                case 0:
                    szptr[wr] = (short)BZip2Constants.RunA;
                    wr++;
                    mtfFreq[BZip2Constants.RunA]++;
                    break;
                case 1:
                    szptr[wr] = (short)BZip2Constants.RunB;
                    wr++;
                    mtfFreq[BZip2Constants.RunB]++;
                    break;
            }
            if (zPend < 2)
            {
                break;
            }
            zPend = (zPend - 2) / 2;
        }
    }

    szptr[wr] = (short)EOB;
    wr++;
    mtfFreq[EOB]++;

    nMTF = wr;
}

void SendMTFValues()
    {
```

```
char[][] len = new char[BZip2Constants.GroupCount][];
for (int i = 0; i < BZip2Constants.GroupCount; ++i)
{
   len[i] = new char[BZip2Constants.MaximumAlphaSize];
}

int gs, ge, totc, bt, bc, iter;
int nSelectors = 0, alphaSize, minLen, maxLen, selCtr;
int nGroups;

alphaSize = nInUse + 2;
for (int t = 0; t < BZip2Constants.GroupCount; t++)
{
   for (int v = 0; v < alphaSize; v++)
   {
      len[t][v] = (char)GREATER_ICOST;
   }
}

/*--- Decide how many coding tables to use ---*/
if (nMTF <= 0)
{
   Panic();
}

if (nMTF < 200)
{
   nGroups = 2;
}
else if (nMTF < 600)
{
   nGroups = 3;
}
else if (nMTF < 1200)
{
   nGroups = 4;
}
else if (nMTF < 2400)
{
   nGroups = 5;
}
else
{
   nGroups = 6;
}

/*--- Generate an initial set of coding tables ---*/
```

```
            int nPart = nGroups;
            int remF = nMTF;
            gs = 0;
            while (nPart > 0)
            {
               int tFreq = remF / nPart;
               int aFreq = 0;
               ge = gs - 1;
               while (aFreq < tFreq && ge < alphaSize - 1)
               {
                  ge++;
                  aFreq += mtfFreq[ge];
               }

               if (ge > gs && nPart != nGroups && nPart != 1 && ((nGroups - nPart) % 2
== 1))
               {
                  aFreq -= mtfFreq[ge];
                  ge--;
               }

               for (int v = 0; v < alphaSize; v++)
               {
                  if (v >= gs && v <= ge)
                  {
                     len[nPart - 1][v] = (char)LESSER_ICOST;
                  }
                  else
                  {
                     len[nPart - 1][v] = (char)GREATER_ICOST;
                  }
               }

               nPart--;
               gs = ge + 1;
               remF -= aFreq;
            }

            int[][] rfreq = new int[BZip2Constants.GroupCount][];
            for (int i = 0; i < BZip2Constants.GroupCount; ++i)
            {
               rfreq[i] = new int[BZip2Constants.MaximumAlphaSize];
            }

            int[] fave = new int[BZip2Constants.GroupCount];
            short[] cost = new short[BZip2Constants.GroupCount];
            /*---
```

```
Iterate up to N_ITERS times to improve the tables.
---*/
for (iter = 0; iter < BZip2Constants.NumberOfIterations; ++iter)
{
   for (int t = 0; t < nGroups; ++t)
   {
      fave[t] = 0;
   }

   for (int t = 0; t < nGroups; ++t)
   {
      for (int v = 0; v < alphaSize; ++v)
      {
         rfreq[t][v] = 0;
      }
   }

   nSelectors = 0;
   totc = 0;
   gs = 0;
   while (true)
   {
      /*--- Set group start & end marks. --*/
      if (gs >= nMTF)
      {
         break;
      }
      ge = gs + BZip2Constants.GroupSize - 1;
      if (ge >= nMTF)
      {
         ge = nMTF - 1;
      }

      /*--
      Calculate the cost of this group as coded
      by each of the coding tables.
      --*/
      for (int t = 0; t < nGroups; t++)
      {
         cost[t] = 0;
      }

      if (nGroups == 6)
      {
         short cost0, cost1, cost2, cost3, cost4, cost5;
         cost0 = cost1 = cost2 = cost3 = cost4 = cost5 = 0;
         for (int i = gs; i <= ge; ++i)
```

```
    {
        short icv = szptr[i];
        cost0 += (short)len[0][icv];
        cost1 += (short)len[1][icv];
        cost2 += (short)len[2][icv];
        cost3 += (short)len[3][icv];
        cost4 += (short)len[4][icv];
        cost5 += (short)len[5][icv];
    }
    cost[0] = cost0;
    cost[1] = cost1;
    cost[2] = cost2;
    cost[3] = cost3;
    cost[4] = cost4;
    cost[5] = cost5;
}
else
{
    for (int i = gs; i <= ge; ++i)
    {
        short icv = szptr[i];
        for (int t = 0; t < nGroups; t++)
        {
            cost[t] += (short)len[t][icv];
        }
    }
}

/*--
Find the coding table which is best for this group,
and record its identity in the selector table.
--*/
bc = 999999999;
bt = -1;
for (int t = 0; t < nGroups; ++t)
{
    if (cost[t] < bc)
    {
        bc = cost[t];
        bt = t;
    }
}
totc += bc;
fave[bt]++;
selector[nSelectors] = (char)bt;
nSelectors++;
```

```
            /*--
            Increment the symbol frequencies for the selected table.
            --*/
            for (int i = gs; i <= ge; ++i)
            {
               ++rfreq[bt][szptr[i]];
            }

            gs = ge + 1;
         }

         /*--
         Recompute the tables based on the accumulated frequencies.
         --*/
         for (int t = 0; t < nGroups; ++t)
         {
            HbMakeCodeLengths(len[t], rfreq[t], alphaSize, 20);
         }
      }

      rfreq = null;
      fave = null;
      cost = null;

      if (!(nGroups < 8))
      {
         Panic();
      }

      if (!(nSelectors < 32768 && nSelectors <= (2 + (900000 /
BZip2Constants.GroupSize))))
      {
         Panic();
      }

      /*--- Compute MTF values for the selectors. ---*/
      char[] pos = new char[BZip2Constants.GroupCount];
      char ll_i, tmp2, tmp;

      for (int i = 0; i < nGroups; i++)
      {
         pos[i] = (char)i;
      }

      for (int i = 0; i < nSelectors; i++)
      {
         ll_i = selector[i];
```

```
    int j = 0;
    tmp = pos[j];
    while (ll_i != tmp)
    {
        j++;
        tmp2 = tmp;
        tmp = pos[j];
        pos[j] = tmp2;
    }
    pos[0] = tmp;
    selectorMtf[i] = (char)j;
}

int[][] code = new int[BZip2Constants.GroupCount][];

for (int i = 0; i < BZip2Constants.GroupCount; ++i)
{
    code[i] = new int[BZip2Constants.MaximumAlphaSize];
}

/*--- Assign actual codes for the tables. --*/
for (int t = 0; t < nGroups; t++)
{
    minLen = 32;
    maxLen = 0;
    for (int i = 0; i < alphaSize; i++)
    {
        if (len[t][i] > maxLen)
        {
            maxLen = len[t][i];
        }
        if (len[t][i] < minLen)
        {
            minLen = len[t][i];
        }
    }
    if (maxLen > 20)
    {
        Panic();
    }
    if (minLen < 1)
    {
        Panic();
    }
    HbAssignCodes(code[t], len[t], minLen, maxLen, alphaSize);
}
```

```
/*--- Transmit the mapping table. ---*/
bool[] inUse16 = new bool[16];
for (int i = 0; i < 16; ++i)
{
   inUse16[i] = false;
   for (int j = 0; j < 16; ++j)
   {
      if (inUse[i * 16 + j])
      {
         inUse16[i] = true;
      }
   }
}

for (int i = 0; i < 16; ++i)
{
   if (inUse16[i])
   {
      BsW(1, 1);
   }
   else
   {
      BsW(1, 0);
   }
}

for (int i = 0; i < 16; ++i)
{
   if (inUse16[i])
   {
      for (int j = 0; j < 16; ++j)
      {
         if (inUse[i * 16 + j])
         {
            BsW(1, 1);
         }
         else
         {
            BsW(1, 0);
         }
      }
   }
}

/*--- Now the selectors. ---*/
BsW(3, nGroups);
BsW(15, nSelectors);
```

```
for (int i = 0; i < nSelectors; ++i)
{
   for (int j = 0; j < selectorMtf[i]; ++j)
   {
      BsW(1, 1);
   }
   BsW(1, 0);
}

/*--- Now the coding tables. ---*/
for (int t = 0; t < nGroups; ++t)
{
   int curr = len[t][0];
   BsW(5, curr);
   for (int i = 0; i < alphaSize; ++i)
   {
      while (curr < len[t][i])
      {
         BsW(2, 2);
         curr++; /* 10 */
      }
      while (curr > len[t][i])
      {
         BsW(2, 3);
         curr--; /* 11 */
      }
      BsW(1, 0);
   }
}

/*--- And finally, the block data proper ---*/
selCtr = 0;
gs = 0;
while (true)
{
   if (gs >= nMTF)
   {
      break;
   }
   ge = gs + BZip2Constants.GroupSize - 1;
   if (ge >= nMTF)
   {
      ge = nMTF - 1;
   }

   for (int i = gs; i <= ge; i++)
   {
```

```
            BsW(len[selector[selCtr]][szptr[i]], code[selector[selCtr]][szptr[i]]);
        }

        gs = ge + 1;
        ++selCtr;
    }
    if (!(selCtr == nSelectors))
    {
        Panic();
    }
}
```

Appendix B

Main Program for Gzip

```
namespace Gzip
{
   public static class GZip
   {
      public static void Decompress(Stream inStream, Stream outStream, bool
isStreamOwner)
      {
         if (inStream == null || outStream == null)
         {
            throw new Exception("Null Stream");
         }
         try
         {
            using (GZipInputStream gzipInput = new GZipInputStream(inStream))
            {
               StreamUtils.Copy(gzipInput, outStream, new byte[4096]);
            }
         }
         finally
         {
            if (isStreamOwner)
            {
               outStream.Close();
            }
         }
      }

      public static void Compress(Stream inStream, Stream outStream, bool
isStreamOwner, int level)
      {
         if (inStream == null || outStream == null)
         {
            throw new Exception("Null Stream");
         }
         try
         {
            using (GZipOutputStream gzipOutput = new GZipOutputStream(outStream,
level))
            {
               StreamUtils.Copy(inStream, gzipOutput, new byte[4096]);
            }
         }
```

```
        finally
        {
           if (isStreamOwner)
           {
              inStream.Close();
           }
        }
     }
  }
}

/*****************************************************************/

protected void Deflate()

                {
                        while (!deflater_.IsNeedingInput)
                        {
                                int deflateCount = deflater_.Deflate(buffer_, 0,
buffer_.Length);
                                if (deflateCount <= 0) {
                                        break;
                                }

                                baseOutputStream_.Write(buffer_, 0, deflateCount);
                        }
                        if (!deflater_.IsNeedingInput) {
                                throw new BaseException("DeflaterOutputStream can't
deflate all input?");
                        }
                }

/*************************************************************/

public int Deflate(byte[] output, int offset, int length)
                {
                        int origLength = length;

                        if (state == CLOSED_STATE) {
                                throw new InvalidOperationException("Deflater
closed");
                        }

                        if (state < BUSY_STATE) {
                                // output header
                                int header = (DEFLATED +
```

```
                                        ((DeflaterConstants.MAX_WBITS - 8) << 4))
<< 8;
                                int level_flags = (level - 1) >> 1;
                                if (level_flags < 0 || level_flags > 3) {
                                        level_flags = 3;
                                }
                                header |= level_flags << 6;
                                if ((state & IS_SETDICT) != 0) {
                                        // Dictionary was set
                                        header |= DeflaterConstants.PRESET_DICT;
                                }
                                header += 31 - (header % 31);

                                pending.WriteShortMSB(header);
                                if ((state & IS_SETDICT) != 0) {
                                        int chksum = engine.Adler;
                                        engine.ResetAdler();
                                        pending.WriteShortMSB(chksum >> 16);
                                        pending.WriteShortMSB(chksum & 0xffff);
                                }

                                state = BUSY_STATE | (state & (IS_FLUSHING |
IS_FINISHING));
                        }

                        for (;;) {
                                int count = pending.Flush(output, offset, length);
                                offset   += count;
                                totalOut += count;
                                length   -= count;

                                if (length == 0 || state == FINISHED_STATE) {
                                        break;
                                }

                                if (!engine.Deflate((state & IS_FLUSHING) != 0, (state
& IS_FINISHING) != 0)) {
                                        if (state == BUSY_STATE) {
                                                // We need more input now
                                                return origLength - length;
                                        } else if (state == FLUSHING_STATE) {
                                                if (level != NO_COMPRESSION) {
                                                        /* We have to supply some
lookahead.  8 bit lookahead
                                                         * is needed by the zlib inflater,
and we must fill
```

```
                                                * the next byte, so that all bits
are flushed.
                                                */
                                               int neededbits = 8 + ((-
pending.BitCount) & 7);
                                               while (neededbits > 0) {
                                                       /* write a static tree block
consisting solely of
                                                        * an EOF:
                                                        */
                                                       pending.WriteBits(2, 10);
                                                       neededbits -= 10;
                                               }
                                       }
                                       state = BUSY_STATE;
                               } else if (state == FINISHING_STATE) {
                                       pending.AlignToByte();

                                       // Compressed data is complete.  Write
footer information if required.
                                       if (!noZlibHeaderOrFooter) {
                                               int adler = engine.Adler;
                                               pending.WriteShortMSB(adler
>> 16);
                                               pending.WriteShortMSB(adler &
0xffff);
                                       }
                                       state = FINISHED_STATE;
                               }
                       }
               }
               return origLength - length;
       }
```

Appendix C

Main Program

```
public static void Bzip2Main(string[] args)
      {
         ArgumentParser parser = new ArgumentParser(args);
         Timer tr = new Timer();

         switch (parser.Command)
         {
            case Command.Help:
               ShowHelp();
               break;

            case Command.Compress:
               //get all the file to be compress on folder
               //Console.WriteLine("Compressing {0} to {1}", parser.Source,
parser.Target);
               LogFile(string.Format("Compressing {0} to {1}", parser.Source,
parser.Target));
               tr.Start();
               BZip2.Compress(File.OpenRead(parser.Source),
File.Create(parser.Target), true, 4096);
               tr.Stop();
               LogFile("Time to Compress: " + tr.Interval.ToString());
               break;

            case Command.Decompress:
               //get all the decompressed file name to be uncompressed
               //Console.WriteLine("Decompressing {0} to {1}", parser.Source,
parser.Target);
               LogFile(string.Format("Compressing {0} to {1}", parser.Source,
parser.Target));
               tr.Start();
               BZip2.Decompress(File.OpenRead(parser.Source),
File.Create(parser.Target), true);
               tr.Stop();
               LogFile("Time to Decompress: " + tr.Interval.ToString());
               break;
         }
      }
```

```
public static void GzipMain(string[] args)
      {
         ArgumentParser parser = new ArgumentParser(args);

         switch (parser.Command)
         {
            case Command.Help:
               ShowHelp();
               break;

            case Command.Compress:
               Console.WriteLine("Compressing {0} to {1}", parser.Source,
parser.Target);
               GZip.Compress(File.OpenRead(parser.Source),
File.Create(parser.Target), true, 4096);
               break;

            case Command.Decompress:
               Console.WriteLine("Decompressing {0} to {1}", parser.Source,
parser.Target);
               GZip.Decompress(File.OpenRead(parser.Source),
File.Create(parser.Target), true);
               break;
         }
      }
```

Appendix D (License Bzip2)

/* This file was derived from a file containing this license:
 *
 * This file is a part of bzip2 and/or libbzip2, a program and library for lossless,
 * block-sorting data compression.
 *
 * Copyright (C) 1996-1998 Julian R Seward. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. The origin of this software must not be misrepresented; you must
 * not claim that you wrote the original software. If you use this
 * software in a product, an acknowledgment in the product
 * documentation would be appreciated but is not required.
 *
 * 3. Altered source versions must be plainly marked as such, and must
 * not be misrepresented as being the original software.
 *
 * 4. The name of the author may not be used to endorse or promote
 * products derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
AUTHOR BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
 *
 * Java version ported by Keiron Liddle, Aftex Software <keiron@aftexsw.com>
1999-2001
 */

Appendix E (License Gzip)

```
// Copyright (C) 2001 Mike Krueger
//
// This file was translated from java, it was part of the GNU Classpath
// Copyright (C) 2001 Free Software Foundation, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
//
// Linking this library statically or dynamically with other modules is
// making a combined work based on this library.  Thus, the terms and
// conditions of the GNU General Public License cover the whole
// combination.
//
// As a special exception, the copyright holders of this library give you
// permission to link this library with independent modules to produce an
// executable, regardless of the license terms of these independent
// modules, and to copy and distribute the resulting executable under
// terms of your choice, provided that you also meet, for each linked
// independent module, the terms and conditions of the license of that
// module.  An independent module is a module which is not derived from
// or based on this library.  If you modify this library, you may extend
// this exception to your version of the library, but you are not
// obligated to do so.  If you do not wish to do so, delete this
// exception statement from your version.
```