PERFORMANCE EVALUATION OF COMPRESSION TECHNIQUES ON SCIENTIFIC DATASET

LAM WAI LEONG

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY UNIVERSITY OF MALAYA KUALA LUMPUR

2006

PERFORMANCE EVALUATION OF COMPRESSION TECHNIQUES ON SCIENTIFIC DATASET

LAM WAI LEONG

DISSERTATION SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER COMPUTER SCIENCE

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY UNIVERSITY OF MALAYA KUALA LUMPUR

2006

Abstract

Data communication is vital, as the world is getting smaller with the help of Internet. The challenge to improve quality and responsiveness of communication is in the network bandwidth bottleneck. However, with compression technologies, the impact of transferring data can be optimized.

There are various compression technologies in the market from different origins both available commercially and public domain. Performance in compression technologies are measured according to required computation power and compression ratios achieved.

However, not all data can be compressed effectively, where desired compression rates are achieved. The reason is that most data are obtained from digitizing or converted from analog signals. Examples: audio, photos, graphs plotted by input sensors.

An important characteristic of data compression lies in the compression ratio and compression speed of a particular data compression tool. Though most theoretical background of compression tool compresses datasets based on Lempel-ziv's algorithm, in reality, these tools varied when it comes to compressing a binary file to a text file or a graphical one. This is evidence in the statistically analysis of the file format. This project looks into various data compression technique and when to use them, with specifically focus on

evaluating the performance of existing data compression and extraction algorithms that best suit scientific dataset.

This project applies various tests on selected range of scientific datasets to ascertain the overall performance against a benchmarking compression technique. The tests are based on a real time network transmission of compression and extraction on a set of scientific datasets over a networked environment.

This project proves that a generic compression algorithm fair better compare to a more format specific compression algorithm when use on a scientific datasets. The outcome and procedures used in this project use as a template for choosing a suitable compression tool for any particular format of dataset. This template shall minimise any doubt and confusion of choosing and using a compression techniques.

Acknowledgement

I would like to express my deepest gratitude to my supervisor, Dr. Rosli Salleh, for his invaluable guidance, insight and encouragement throughout the entire project.

Besides that, I would like to express my sincerest gratitude and utmost appreciation to my lecturer Mr. Ang Tan Fong for their invaluable advice and guidance throughout the whole development project. Their contributions are truly appreciated.

I would also like to thank my family members for their unending support and understanding. Last but not least, my utmost gratitude is conveyed to my beloved course-mates and friends, especially Mr Davis Tan Kok Bing, Mr. Alvin Yew Wei Han and Mr. Yoo Sang Nge for their support and sharing of their much-appreciated knowledge with me. They have been very helpful and supportive.

Thank you very much!

Table of Contents

List of Figures

List of Tables

Abbreviations

AAC	Advanced Audio Coding
AC	Associative Coding
ARI	Arithmetic Coding
ASCII	American Standard Code for Information Interchange
AVC	Advanced Video Coding
BMP	bit-mapped graphics
BWT	Burrows-Wheeler Transform
CCITT	Consultative Committee for International Telegraphy and Telephony.
CD	compact disc
CDDA	Compact Disc Digital Audio
CELP	Code Excited Linear Predictor
CIF	Common Intermediate Format
CM	Context Modeling
CPU	central processing unit
CTW	Context Tree Weighting
DARPA	The Defense Advanced Research Projects Agency
DB	Decibels
DC	Distance Coding
DCT	Discrete Cosine Transform
DM	Dynamic Markov Modeling
DVD-R	DVD-Recordable
FDS	Fire Dynamic Simulator
FFT	Fast Fourier transform
FLAC	Free Lossless Audio Codec
GIF	Graphics Interchange Format
HTML	Hypertext Markup Language
Huff	Huffman
IBC	International Benchmarking Clearinghouse
IP	Internet Protocol
ITU	International Telecommunication Union

JPEG	Joint Photographic Experts Group
JVT	Joint Video Team
LZ	Lempel-Ziv
LZRW	Lempel-Ziv Ross Williams
LZW	Lempel-Zif-Welch
LPC	Linear Predictive Coding
MB	Mega Byte
MP3	MPEG, audio layer 3
MPEG	Moving Picture Experts Group
MS	Millisecond
NCBI	National Center for Biotechnology Information
NIST	National Institute of Standards and Technology
PNG	Portable Network Graphics
PPM	Prediction by Partial Match
PSD	Photoshop document
PSP	Paint Shop Pro document
RAM	Random Access Memory
RLC	Run-Length Coding
RLE	Run-Length Encoding
RM	Real Media
SE	Second Edition
SF	Shannon-Fano
TCP/IP	Transmission Control Protocol/Internet Protocol
TIFF	Tagged Image File Format
TTA	The True Audio
VCEG	Video Coding Experts Group
WAN	Wide Area Network
WMA	Windows Media Audio

CHAPTER 1: Introduction

With the rapid growth of the scientific research and the establishment of compression algorithms as the fundamental layer of choice in most research environments, the drawbacks of compression techniques have become more obvious. Any form of communication, compressed data communication only works when both the sender and receiver of the information understand the encoding scheme.

On rapid development in demand of scientific research, compression is useful because it helps to reduce the consumption of expensive resources, such as disk space or transmission bandwidth. On the downside of it, compressed data must be uncompressed to be viewed (or heard), and this extra processing may be detrimental to some applications.

After decade with the time when a computer known as Apple II and the monitor was a monochrome and window was never heard of. That was the beginning of the technology era; that was the time when Internet and networking was only known to university and advanced research authority like DARPA.

It used to be that the data to be shared among peer can always fitted into a single diskette of 1.4 MB. As the advancement in the technology area, with the boom of internet, the size of the data to be shared among peer become the bottleneck that were never thought of. The advancement in the electronic industries also contributed into the mounting problem capturing large size of digital data.

If the technology trends are as predicted by Moore's law, in no time, the internet and its backbone technology (TCP/IP) will halt and break down, the hard drive will and RAM will not be able to keep up with the data rate. Without a proper solution, the advancement in technology would simply slowdown gradually. This is the price to pay for higher definition graphics, greater quality entertainment, more realistic sight and sound. To be exact, there are more information embedded into a dataset, the greater the size; the greater the data sizes, eventually it will reach the physical limit, the limit of our silicon technology.

The only solution to the problem was data compression. The idea was to compress the data into smaller size that can help in reducing storage problem for the ever growing large sized data and increase the throughput over the network.

1.1.1 Introduction to Data Compression

Based on the *Information Theory* by Claude E. Shannon in the 50's, scientist and mathematician were able to come out with the algorithm that compresses the data or message based on the statistical redundancy of that particular data or message [17]. For example, the letter 'e' is much more common in English text than the letter 'z', and the probability that the letter 'q' will be followed by the letter 'z' is rather small.

In computer science studies, data compression is the process of encoding information using fewer bits than a more obvious representation would use, through use of specific encoding schemes. For example, this article could be encoded with fewer bits if we accept the convention that the word "compression" is encoded as "comp". From which compression technology spawns into various shape and sizes.

Since the first introduction of the *Information Theory*, almost 30 years ago, Abraham Lempel and Jacob Ziv introduced the first pointer-based encoding in 1977, followed by the work of Terry Welch, which form the LZRW algorithm [24]. The initial target of this algorithm is text (ASCII) contents. With the emergence of the digital contents, more and more algorithms were developed, which could be generally grouped into two major categories. One category is lossless compression, and other one is lossy compression. As the word implies, lossless compression will pertain the contents of the source, while lossy compression allows lost in some portion of information.

These two major categories are to be further divide into various other variants that are design to specifically target to tackle the different format coded of the datasets.

1.1.2 Introduction to Simulation

A simulation is defined as an imitation of some real devices or state of affairs [32]. Simulation attempts to represent certain features of the behavioral of a physical or abstract system by the behavioral of another system.

Simulation is used in many contexts, including the modeling of natural systems, and human systems to gain insight into the operation of those systems; and simulation in technology and safety engineering where the goal is to test some real-world practical scenario. Simulation, using a *simulator* or otherwise experimenting with a fictitious situation can show the eventual real effects of some possible conditions.

There is various type of simulation, but in this project we are only interested in computer simulation. The main reason are mainly on the reliability and the trust people put in computer simulations depends on the validity of the simulation model, thus verification and validation are most crucial importance in the development of computer simulations. Moreover, important aspect of computer simulations is that of reproducibility of the results, meaning that a simulation model should not provide a different answer for each execution. An exception to reproducibility is human in the loop simulations such as flight simulations and computer games. Here a human is part of the simulation and thus influences the outcome in a way that is hard if not impossible to reproduce exactly.

Computer simulation is a useful part of modeling many natural systems in physics, chemistry and biology, and human systems in economics as well as in

engineering to gain insight into the operation of those systems. A good example of the usefulness of using computers to simulate can be found in the field of network traffic simulation. Computer simulations are often considered to be *human out of the loop* simulations [33].

Traditionally, the formal modeling of systems has been via a mathematical model, which attempts to find analytical solutions to problems which enable the prediction of the behavioral of the system from a set of parameters and initial conditions. Computer simulation is often used as an adjunct to, or substitution for, modeling systems for which simple closed form analytic solutions are not possible. There are many different types of computer simulation, the common feature are, they all share is the attempt to generate a sample of representative scenarios for a model in which a complete enumeration of all possible states of the model would be prohibitive or impossible.

1.1.3 Summary

We now know the motivation behind compression technology and generally how compression technologies works. But in reality we are still far from understanding how each and every compression algorithm treats different types of data sets. In the next chapter, we will start to look at how the specialised algorithm tackle and compress specific datasets.

1.2 Problem Statement and Objective

This chapter explains the motivation behind this study and the objective that wish to accomplish.

1.2.1 Problem Statement

With the increasing data size in the digital world, due to new and more sophisticated data capturing technology, it is just the matter of time when our archive and storage technology reaches its physical limit. And the impending problem will also cause our existing networking infrastructure to seized functioning due to the heavy load of data transmission. All these in time will affect all human communication activities. Yet, with the overwhelming number of compression technologies available in the scientific community, ranging from the most general type to highly specific and proprietary, it poses another question, "Which one is more effective than the other, when to use and why?"

This study will try to answer the question on how compression technology impacts the performance of human activities, especially in the scientific community and how to evaluate which compression technology best suit in scientific usage.

1.2.2 Objectives of the Project

The following are the objectives of the project:

- To analyse significance of compression algorithm over data transmission network
- To evaluate the performance of the compression algorithms over the scientific datasets
- To identify significance of compression towards extraction performance
- To identify the behaviours of a compression algorithms

1.2.3 Project Scope

The following are the goal of the project:

- To identify compression characteristic of the scientific datasets
- To develop a simulator program using C/C++ language for compressing and decompressing datasets over TCP/IP network
- The simulator will include both client and server which evaluate on three algorithm as zlib, LZRW and bzip
- To investigate the significant of compression technology and different type of compression techniques for scientific datasets

1.3 Methodology

The purpose of this study is to examine the compression and extraction performance on a set of scientific dataset.

The four purposes of this chapter are to (1) describe the research methodology of this study, (2) explain the sample dataset generation, (3) describe the procedure used in selecting the appropriate simulator algorithm, and (4) provide an explanation of the procedures used to analyse the data.

1.3.1 Research Methodology

A benchmarking comparison methodology was used for this study. A benchmark is selected to compare with the compression and extraction result. The term 'benchmarking' is commonly applied to a research methodology designed to compare and differentiate data from different sets of data that are supposed to arrive at the same results.

Harrington & Harrington define benchmarking as "a systematic way to identify, understand, and creatively evolve superior products, services, designs, equipment, processes, and practices to improve one's real performance" [2]. The International Benchmarking Clearinghouse, or IBC, defines benchmarking as "the process of continuously comparing and measuring an organisation with leaders anywhere in the world to gain information that will help to take action to improve the performance" [34]. According to Camp, "Benchmarking is the search for industry best practices that lead to superior performance"[3].

For these reasons, the benchmarking comparison research methodology is selected as the methodology used in this study to assess and analyse the performance of various compression and extraction technology.

1.3.2 Sample Datasets Generation

The sample datasets uses in the evaluation must fulfill the following characteristics:

- Random and different scales.
- Vary in large quantity, while some data might vary in decimal points value.

The scientific datasets chosen for this project which representing research area are from biological dataset, water quality dataset and fire dynamic dataset. These all dataset are real life sampling for actual research purposes.

1.3.3 Simulator Selection

One the simulator selected for this project, the “Fire Dynamic Simulator”, is because of its ability to generate datasets on multiple aspects on fire breakout scenario, which includes density, pressure, heat, chemical composition, and velocity [8]. Each dataset is measured with high resolution that are taken on hundred of thousands, to millions of grid cell in a given space, example, a room. The time steps are from thousands to hundreds of thousand.

Therefore, these samples are suitable to use as the sample datasets in the attempt to evaluate the performance on compression and extraction of scientific datasets.

1.3.4 Method of Analysis

The data analysis consisted of examining the three major areas, the compression ratio, compression time, and the throughput of the generated data sets through a simulated data transmission. All three criteria are subject to comparison based on a benchmarking algorithm over the dataset size. Tables are constructed form comparison on different compression algorithm of the above criteria.

1.3.5 Summary

The purpose of this chapter was to describe the research methodology of this study, explained the sample dataset generation, described the procedure used in selecting the appropriate simulator algorithm, and provides an explanation of the procedures used to analyse data.

1.4 Report Organisations

This report has a total of 7 chapters. It is organised as follows:

Chapter 1 is the introduction of the project. This chapter also defined the objectives and goals of this project as well as describing the research methods used.

Chapter 2 In this chapter, we look into details the various compression algorithms that exist in the current market. Hence, it explore into the different area of data compression. We will also look at what technology of compression algorithms based on.

Chapter 3 In this chapter, we will discuss how this project is derived from, and how it is being developed. We will also look at how specific algorithms and datasets are selected for this project.

Chapter 4 This chapter describes and analyses the system used to simulate the test in this project.

Chapter 5 This chapter covers the design aspects of the system. It also describes the flow of the system accompanied by the system flow diagrams.

Chapter 6 This chapter covers the detailed implementation of the system and discusses the simulations and the results. This chapter also summarises how the performance of the compression algorithms being gauged.

Chapter 7 This chapter describes the overall findings and conclusion of this project and summarises the research and the development of this system.

CHAPTER 2: Literature Review

Compression can be used and has been applied on variety of data types. Some common data types are audio data, image, text file and video data. In this study we will look at the 4 major types of data types: audio, image, video and text.

2.1 Audio Compression

Various popular audio compression format includes MP3, RM (Real Media), Ogg and FLAC. Generally the two major groups of compressed audio file formats are as follow:

- formats with lossless compression, such as Free Lossless Audio Codec (FLAC), Monkey's Audio (filename extension APE), WavPack, Shorten, TTA and lossless Windows Media Audio (WMA).
- formats with lossy compression, such as MP3, Ogg Vorbis (filename extension OGG), lossy Windows Media Audio (WMA) and Advanced Audio Coding (AAC).

Lossy file formats are based on psychoacoustic models that leave out sounds that humans cannot or can hardly hear, e.g. a low volume sound after a big volume sound. MP3 is such an example.

Lossless audio formats (such as TTA) provide compression about 2:1, but no data/quality is lost in the compression - when uncompressed; the data will be identical to the original. Lossless audio codecs are a good choice to keep the music's original quality. For example, using the free The True Audio (TTA)

lossless audio codec you can store up to 20 audio CDs from your music collection on one single DVD-R for playback.

One of the most popular audio file formats was MP3, which uses the MPEG-1 audio layer 3 codec to provide acceptable lossy compression for music files. The compression is about 10:1 compared with uncompressed WAV files (in a standard compression scheme), therefore a CD with MP3 files can store about 10 hours of music, compared to one hour of the standard Compact Disc Digital Audio (CDDA), which uses WAV files. As mention in Jocelyn Dabeau article, *An Introduction to MP3*, the MP3 compression "takes into account the perception of sound waves by the human ear", and then apply "traditional compression techniques to achieve a high level of data reduction while retaining near-CD quality sound" [35].

There are many newer audio formats and codecs claiming to achieve improved compression and quality over MP3. Ogg Vorbis is an unpatented, open and free codec [36]. Microsoft has its Windows Media Audio format.

Lossless compression of sound is not nearly as widely used outside of professional applications, as lossy compression can provide a much greater data compression ratio with nearly the same apparent quality [13].

Below we look at how analogue audio is sampled and various type of audio compression technique.

2.1.1 Audio Sampling

The digital representation of audio data offers many advantages such as high noise immunity, stability, and reproducibility [14]. Audio in digital form also allows for efficient implementation and execution of many audio processing functions through the computer.

The conversion of audio from analog to digital begins by sampling the audio input at regular, discrete intervals of time and quantising the sampled values into a discrete number of evenly spaced levels. According to the Nyquist theory, a time-sampled signal can faithfully represent a signal up to half the sampling rate. Above that threshold, the frequencies become blurred and signal noise becomes readily apparent [37].

The usual sampling frequencies in today typically used range from 8 kHz for basic speech to 48 kHz for commercial DAT machines. The number of quantiser levels is typically a power of 2 to make full use of a fixed number of bits per audio sample. The typical range for bits per sample is between 8 and 16 bits. This allows for a range of 256 to 65,536 levels of quantisation per sample. With each additional bit of quantiser spacing, the signal to noise ratio increases by roughly 6 decibels (dB). Thus, the dynamic range capability of these representations is from 48 to 96 dB, respectively [38].

The data rates associated with uncompressed digital audio are substantial. For audio data on a CD, for example, which is sampled at 44.1 kHz with 16 bits per

channel for two channels, about 1.4 megabits per second are processed. A clear need exists for some form of compression to enable the more efficient storage and transmission of digital audio data [30].

2.1.2 Voc File Compression

The simplest compression techniques simply removed any silence from the entire sample. Creative Labs introduced this form of compression with their introduction of the SoundBlaster line of sound cards [39]. This method analyses the whole sample and then codes the silence into the sample using byte codes. It is very similar to run-length coding.

2.1.3 Linear Predictive Coding and Code Excited Linear Predictor

This was an early development in audio compression that was used primarily for speech. A Linear Predictive Coding (LPC) encoder compares speech to an analytical model of the vocal tract, then throws away the speech and stores the parameters of the best-fit model. The output quality was poor and was often compared to computer speech and thus is not used much today [40]

Then a later development, Code Excited Linear Predictor (CELP), increased the complexity of the speech model further, while allowing for greater compression due to faster computers, and produced much better results [41]. Sound quality improved, while the compression ratio increased. The algorithm compares speech with an analytical model of the vocal tract and computes the errors

between the original speech and the model. It transmits both model parameters and a very compressed representation of the errors.

2.1.4 Mu-law and A-law compression

Logarithmic compression is a good method because it matches the way the human ear works [12]. It only loses information which the ear would not hear anyway, and gives good quality results for both speech and music. Although the compression ratio is not very high it requires very little processing power to achieve. It is the international standard telephony encoding format, also known as International Telecommunication Union - ITU (formerly Consultative Committee for International Telegraphy and Telephony. - CCITT) standard. It is commonly used in North America and Japan for ISDN 8 kHz sampled, voice grade, digital telephone service.

It packs each 16-bit sample into 8 bits by using a logarithmic table to encode a 13-bit dynamic range, dropping the least significant 3 bits of precision. The quantisation levels are dispersed unevenly instead of linearly to mimic the way that the human ear perceives sound levels differently at different frequencies. Unlike linear quantisation, the logarithmic step spacing's represent low-amplitude samples with greater accuracy than higher-amplitude samples. This method is fast and compresses data into half the size of the original sample. This method also is used quite widely due to the universal nature of its adoption.

2.2 Video Compression

The increasing demand to incorporate video data into telecommunications services, the corporate environment, the entertainment industry, and even at home has made digital video technology a necessity. A problem is that still image and digital video data rates are very large, typically in the range of 150Mbits/sec [12]. Data rates of this magnitude would consume a lot of the bandwidth, storage and computing resources in the typical personal computer. For this reason, video compression standards have been developed to eliminate picture redundancy, allowing video information to be transmitted and stored in a compact and efficient manner [10].

2.2.1 Video Compression Standards

During the '80s and '90s, Discrete Cosine Transform (DCT) based compression algorithms and international standards were developed to alleviate storage and bandwidth limitations imposed by digital still image and motion video applications [22].

Today there are two DCT-based standards that are widely used and accepted worldwide:

- H.261 (Video codec for audiovisual services)
- MPEG (Motion Picture Experts Group)

Each of these standards is well suited for particular applications: H.261 for video conferencing, and MPEG for high-quality, multimedia systems.

2.2.2 Video Compression Processing Functions

As mentioned earlier, the JPEG, H.261, and MPEG video compression standards are all based on the DCT. In addition to being DCT-based, many processing functions and compression principles are common to these standards [22].

The basic compression scheme for all three standards can be summarised as follows: divide the picture into 8x8 blocks, determine relevant picture information, discard redundant or insignificant information, and encode relevant picture information with the least number of bits.

Common functions to all three standards are:

- DCT
- Zig-Zag Scanning
- Quantisation
- Entropy Coding
- Motion Estimation

2.2.2.1 DCT & Zig-Zag Scanning

The Discrete Cosine Transform is closely related to the Discrete Fourier Transform (FFT) and, as such, allows data to be represented in terms of its frequency components. Similarly, in image processing applications the two dimensional (2D) DCT maps a picture or a picture segment into its 2D frequency components [16].

For video compression applications, since the variations in the block tend to be low, the great majority of these transformations result in a more compact representation of the block. The block energy is packed into the corresponding lower frequency bins [16].

The DCT component at coordinates (0,0) is referred to as the DC bin. All other components are referred to as AC bins.

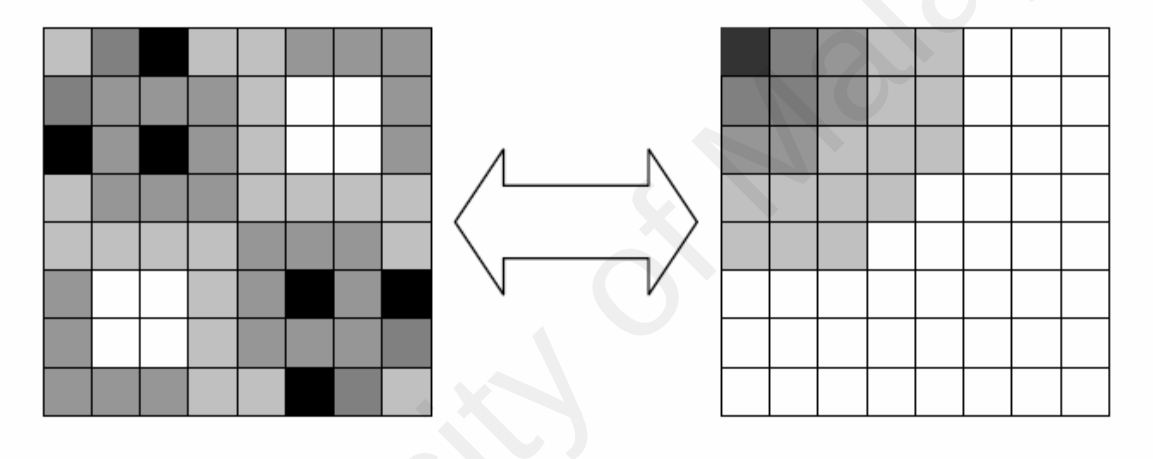


Figure 2.1 : The DCT Operation

Since the mapping is from lower to higher frequencies in the horizontal and vertical directions, zig-zag scanning of the resulting 2D frequency bins clusters packets of picture information from low to high frequencies into a 1D stream of bins.

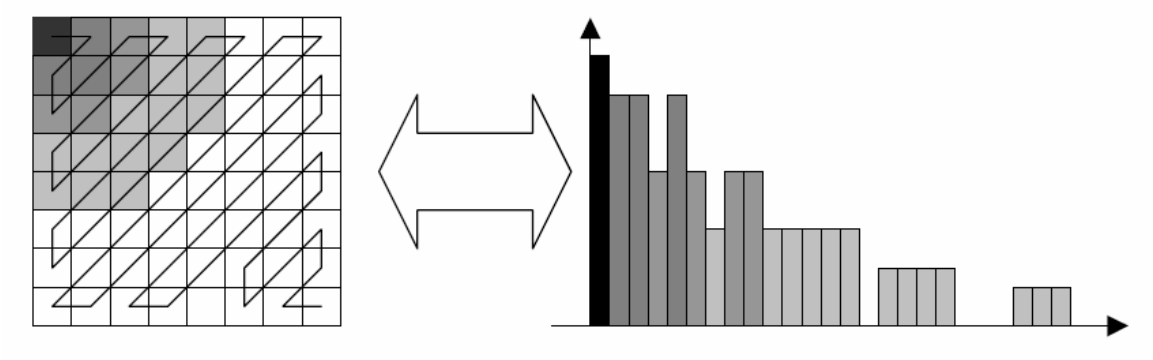


Figure 2.2 : Zig-zag scanning

2.2.2.2 Quantisation

Quantisation is the primary source of data loss in DCT based image compression algorithms. Quantisation reduces the amount of information required to represent the frequency bins by converting amplitudes that fall in certain ranges to one in a set of quantisation levels [22]. For simplicity, all the standard image compression algorithms use linear quantisation where the step size quantisation levels are constant.

Quantisation in the frequency domain has many advantages over directly quantizing the pixel values. Quantisation of the pixel values results in a visual artifact called "contour" distortion where small changes in amplitude in a gradient area cause step-sized changes in the reconstructed amplitude. Except for the DC bin, quantisation error for each of the frequency bins average out to zero over the 8 x 8 block.

2.2.2.3 Entropy Coding

Entropy coding is a loss-less compression scheme based on statistical properties of the picture or the stream of information to be compressed [15]. Although entropy coding is implemented slightly different in each of the standards, the basic “entropy coding” scheme consists of encoding the most frequently occurring patterns with the least number of bits. In, this manner, data can be compressed by an additional factor of 3 or 4. Entropy coding for video compression applications is a two step process: Zero Run-Length Coding (RLC) and Huffman coding [15].

RLC data is an intermediate symbolic representation of the quantized bins which utilizes a pair of numbers. The first number represents the number of consecutive zeros while the second number represents the value between zero-run lengths. For instance the RLC code (5,8) represents the sequence (0,0,0,0,0,8) of numbers.

Huffman coding assigns a variable length code to the RLC data, producing variable length bitstream data [16]. This requires Huffman tables which can be pre-computed based on statistical properties of the image or can be pre-determined if a default table is to be used (as it is in H.261 and MPEG). In either case, the same table is used to decode the bitstream data. As mentioned above, frequently occurring RLC patterns are coded with the least number of bits. At this point the digital stream, which is a representation of the picture, has no specific

boundaries or fixed length. This information can now be stored or appropriately prepared for transmission.

2.2.2.4 Motion Estimation

In general, successive pictures in a motion video sequence tend to be highly correlated, that is, the pictures change slightly over a small period of time [42]. This implies that the arithmetic difference between these pictures is small. For this reason, compression ratios for motion video sequences may be increased by encoding the arithmetic difference between two or more successive frames.

In contrast, objects that are in motion increase the arithmetic difference between frames which in turn implies that more bits are required to encode the sequence. To address this issue, motion estimation is utilised to determine the displacement of an object

Motion estimation is the process by which elements in a picture are best correlated to elements in other pictures (ahead or behind) by the estimated amount of motion. The amount of motion is encapsulated in the motion vector. Forward motion vectors refer to correlation with previous pictures. Backward motion vectors refer to correlation with future pictures.

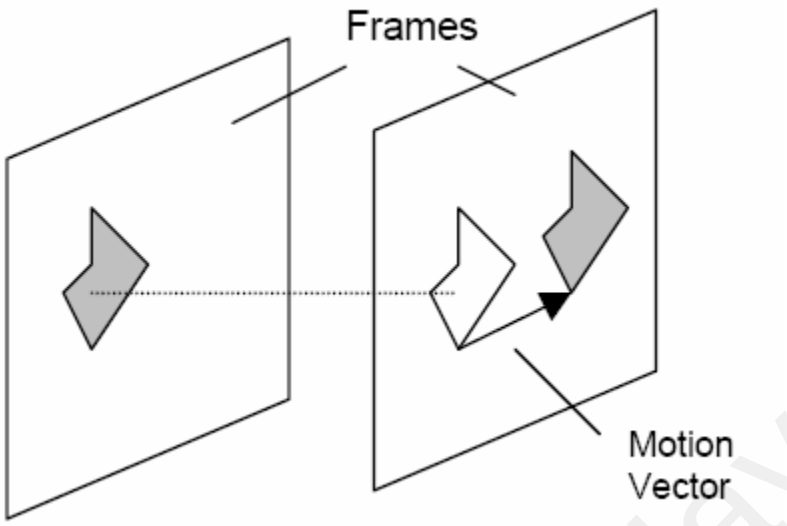


Figure 2.3 : Motion Estimation

An efficient motion estimation algorithm increases frame correlation, which in turn minimises pixel arithmetic difference. Resulting in not only higher compression ratios but also in higher quality decoded video sequences. Motion estimation is an extremely computationally intensive operation difficult to implement in real-time. For this reason, varieties of motion estimation algorithms have been implemented by the industry [42].

2.2.3 The H.261 Compression Algorithm

Video conferencing and video telephony are the intended applications for the H.261 compression algorithm [12]. For these applications, representation of limited motion video (taking heads) is a key component.

To allow for low-cost implementations, H.261 fixes many of the system parameters. Only the YUV color component separation with the 4:2:0 sampling

ratio is allowed by the standard. In addition, H.261 allows for only two frame sizes, CIF (352x288) and QCIF (176x144).

As with the JPEG standard, each color component picture is partitioned into 8x8 pixel blocks of picture samples. Instead of coding each block separately, H.261 groups 4 Y blocks, 1 U block, and 1 V block together into a unit called a macroblock. The macroblock is the basic unit for compression [12].

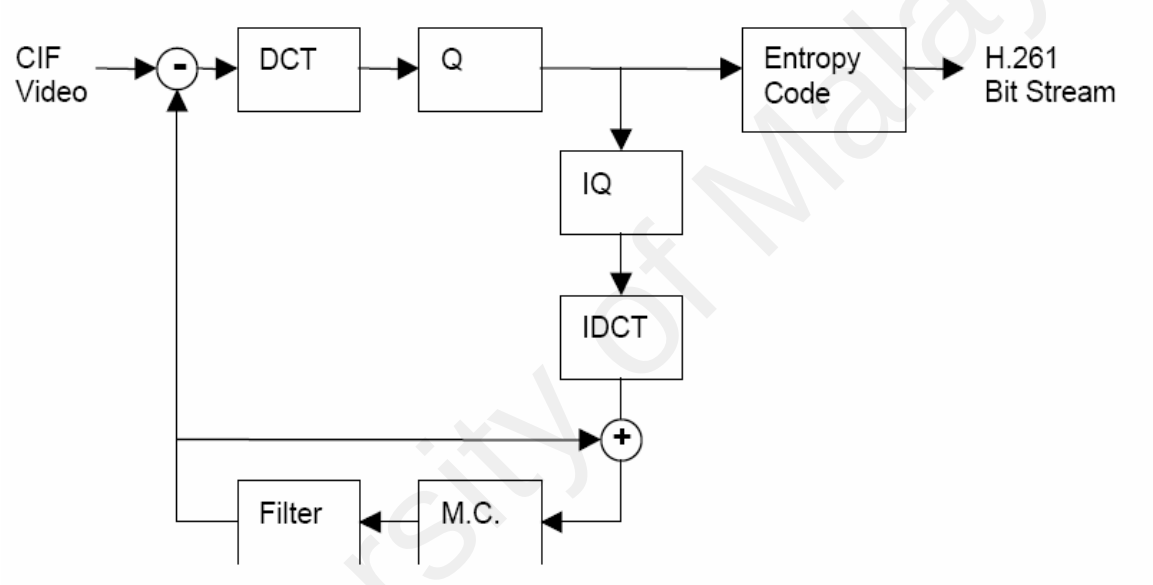


Figure 2.4 : H.261 Encoder

To compress each macroblock, the H.261 standard allows the compressor to select from several compression options [12]. The H.261 standard only specifies the decoding of each of the compression options. The method used to select the options is not standardised. This allows vendors to differentiate their products by providing methods with different cost-quality tradeoffs. A typical method used to compress H.261 is described below.

First, motion estimation is performed on each macroblock. Since objects in the frame may be moving in different directions, each macroblock is allowed to have a different motion vector. The motion vector is used as a displacement vector to fetch a macroblock from the preceding frame to be used as a prediction. Motion estimation in H.261 is only performed relative to the preceding frame, and on full-pixel offsets up to a maximum of +/-15 in the horizontal and vertical directions. To improve the prediction, H.261 allows for an optional loop-filter to be applied to the prediction on a macroblock basis.

Next, a decision must be made to code either the arithmetic difference between the offset prediction macroblock and the current macroblock or to code the current macroblock from scratch. Since the arithmetic difference is usually small, coding the arithmetic difference results in higher compression.

An 8x8 DCT is applied to each block in either the arithmetic difference macroblock or the current macroblock. Instead of quantisation matrices, H.261 uses one quantisation scale for all frequency bins. Since the DC bin is the most important, it is separately quantized to a fixed 8 bit scale. Adjustment of the quantisation scale on a per macroblock basis is the primary method for controlling the quality and compression ratio in H.261.

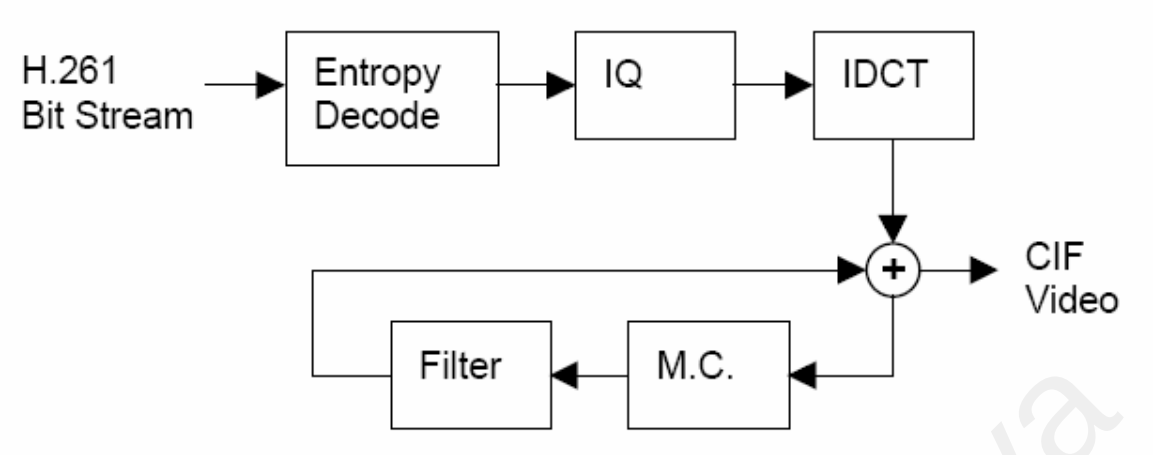


Figure 2.5 : H.261 Decoder

The final stage of compression is the zig-zag scanning, run-length encoding and entropy coding. H.261 specifies fixed Huffman coding tables for entropy coding.

To decompress an H.261 frame inverse operations are performed in reverse order. Motion estimation is not necessary since the motion vectors are embedded in the compressed bitstream. The H.261 de-compressor simply applies the motion vector offset to retrieve the prediction, if necessary.

ITU-T H.264 / MPEG-4 Advanced Video Coding (commonly referred as H.264/AVC) is the newest entry in the series of international video coding standards [45]. It is currently the most powerful and state-of-the-art standard, and was developed by a Joint Video Team (JVT) consisting of experts from ITU-T's Video Coding Experts Group (VCEG) and ISO/IEC's Moving Picture Experts Group (MPEG).

In the process of creation, a standard was created that improved coding efficiency by a factor of at least about two (on average) over MPEG-2, the most widely used video coding standard [45].

With the wide breadth of applications in the market, the application focus for the work was correspondingly broad ranging from video conferencing to entertainment (broadcasting over cable, satellite, terrestrial, cable modem, DSL; storage on DVDs and hard disks; video on demand etc.) to streaming video, surveillance and military applications, and digital cinema.

2.2.4 The MPEG Compression Algorithm

MPEG compression algorithms were developed to address the need for higher quality pictures and increased system flexibility, which are required by multi-media systems [12]. Since it was developed later, MPEG was able to leverage the efforts behind the development of the H.261 algorithms.

As with H.261, only the YUV color component separation with the 4:2:0 sampling ratio is allowed by the MPEG standard. Unlike H.261, the frame size is not fixed although a 352x240 frame size is typically used. MPEG adopted the macroblock of H.261 (4 Y blocks, 1 U block, and 1 V block) as the basic unit for compression. To compress each macroblock, the MPEG standard allows the compressor to select from several compression options [12].

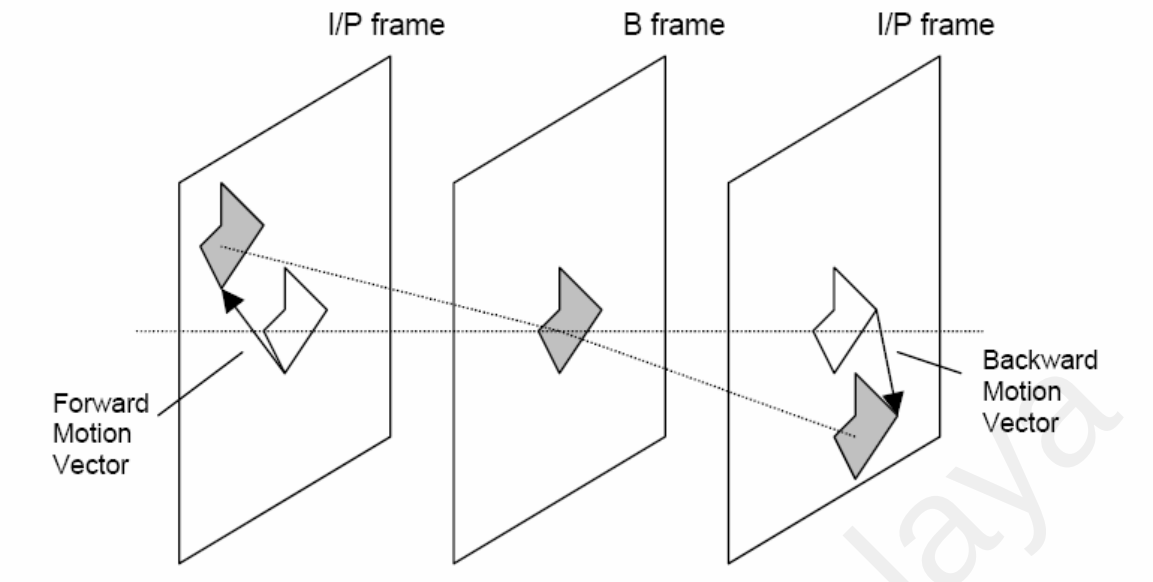


Figure 2.6 : MPEG Motion Compensation

There are many more options available under the MPEG standard than under H.261 [12]. As with H.26l, the MPEG standard only specifies the decoding of each of the compression options. The method used to select the options is not standardised, allowing vendors to differentiate their products by providing methods with different cost-quality trade-offs. A typical method used to compress MPEG is described below.

First, motion estimation is performed on each macroblock. In addition to motion estimation from just the preceding frame, MPEG allows for prediction from frames in the past or future or a combination of a past and future frame (with restrictions).

Since objects in the frame may not be moving steadily from frame to frame, each macroblock is allowed to have up to two motion vectors (one relative to a past

frame and another relative to a future frame). Note that to allow for predictions from future frames, the extra frames must be buffered and the sequence coded out-of-order.

Motion estimation is also allowed over a greater range (up to +/- 1023) and with half-pixel resolution. The loop-filter of H.261 is not included in MPEG because the half-pixel resolution motion vectors serve the same purpose.

Next, a four-way decision must be made. MPEG allows the prediction formed from the arithmetic difference between the current macroblock and an offset macroblock from a past frame, future frame, an average between past and future frame, to be coded; or to code the current macroblock from scratch. A different decision can be made for each macroblock subject to the restrictions that follow. Key frames (called Intra or I frames) which do not allow any predicted macroblocks are coded periodically to allow for random access into the video stream. Forward predicted frames (called P frames) allow macroblocks predicted from past P frames or I frames or macroblocks coded from scratch. I frames and P frames are used as past and future frames for Bi-directional predicted frames (called B frames). B frames allow for all four types of macroblocks.

An 8x8 DCT is applied to each block in either the arithmetic difference or current macroblock. MPEG uses both matrices (like JPEG) and a scale factor (like H.261) for quantisation. Since the DC bin is the most important, it is quantized to a fixed 8 bit scale.

Since the visual effects of frequency bin quantisation are different for predicted and current blocks, MPEG allows for two matrices (one for each type). Typically, the matrices are set once for a picture sequence and the quantisation scale is adjusted to control the compression ratio.

The final stage of compression is the zig-zag scanning, run-length encoding and entropy coding. Like H.261, MPEG specifies fixed Huffman coding tables for entropy coding.

To decompress an MPEG frame each operation is performed in reverse except for motion estimation. Since the motion vectors and the decision are embedded in the compressed bit-stream, the MPEG de-compressor just needs to apply the motion vector offsets to retrieve the prediction from the past and/or future frames if necessary.

2.3 Image Compression

In image compression, there are numerous compressions available, for example, JPG, GIF, TIFF, PNG, BMP. But what are they, and what makes them different from the others?

TIFF is, in principle, a very flexible format that can be lossless or lossy. The details of the image storage algorithm are included as part of the file. In practice, TIFF is used almost exclusively as a lossless image storage format that uses no compression at all. Most graphics programs that use TIFF do not compression. Consequently, file sizes are quite big [29].

This is usually the best quality output from a digital camera. Digital cameras often offer around three JPG quality settings plus TIFF. Since JPG always means at least some loss of quality, TIFF means better quality. However, the file size is huge compared to even the best JPG setting, and the advantages may not be noticeable.

A more important use of TIFF is as the working storage format as you edit and manipulate digital images. You do not want to go through several loads, edit, save cycles with JPG storage, as the degradation accumulates with each new save. One or two JPG saves at high quality may not be noticeable, but the tenth certainly will be. TIFF is lossless, so there is no degradation associated with saving a TIFF file.

PNG is also a lossless storage format [29]. However, in contrast with common TIFF usage, it looks for patterns in the image that it can use to compress file size. The compression is exactly reversible, so the image is recovered exactly. GIF creates a table of up to 256 colors from a pool of 16 million. If the image has fewer than 256 colors, GIF can render the image exactly. When the image contains many colors, software that creates the GIF uses any of several algorithms to approximate the colors in the image with the limited palette of 256 colors available. Better algorithms search the image to find an optimum set of 256 colors. Sometimes GIF uses the nearest color to represent each pixel, and sometimes it uses "error diffusion" to adjust the color of nearby pixels to correct for the error in each pixel.

PNG is of principal value in two applications:

1. If you have an image with large areas of exactly uniform color, but contains more than 256 colors, PNG is your choice. Its strategy is similar to that of GIF, but it supports 16 million colors, not just 256.

If you want to display a photograph exactly without loss on the web, PNG is your choice. Later generation web browsers support PNG, and PNG is the only lossless format that web browsers support.

GIF achieves compression in two ways. First, it reduces the number of colors of color-rich images, thereby reducing the number of bits needed per pixel, as just described. Second, it replaces commonly occurring patterns (especially large areas of uniform color) with a short abbreviation: instead of storing "white, white,

white, white, white," it stores "5 white." Thus, GIF is "lossless" only for images with 256 colors or less. For a rich, true color image, GIF may "lose" 99.998% of the colors.

JPG is optimized for photographs and similar continuous tone images that contain many, many colors. It can achieve astounding compression ratios even while maintaining very high image quality. GIF compression is unkind to such images. JPG works by analyzing images and discarding kinds of information that the eye is least likely to notice. It stores information as 24 bit color. Important: the degree of compression of JPG is adjustable. At moderate compression levels of photographic images, it is very difficult for the eye to discern any difference from the original, even at extreme magnification. Compression factors of more than 20 are often quite acceptable. Better graphics programs, such as Paint Shop Pro and Photoshop, allow you to view the image quality and file size as a function of compression level, so that you can conveniently choose the balance between qualities and file size.

This is the format of choice for nearly all photographs on the web. You can achieve excellent quality even at rather high compression settings. Digital cameras save in a JPG format by default. Switching to TIFF or RAW improves quality in principle, but the difference is difficult to see.

RAW is an image output option available on some digital cameras [29]. Though lossless, it is a factor of three of four smaller than TIFF files of the same image. The disadvantage is that there is a different RAW format for each manufacturer, and so you may have to use the manufacturer's software to view the images. (Some graphics applications can read some manufacturer's RAW formats.) [29]

PSD, **PSP** are proprietary formats used by graphics programs. Photoshop's files have the PSD (Photoshop document) extension, while Paint Shop Pro files use PSP [29]. These are the preferred working formats as you edit images in the software, because only the proprietary formats retain all the editing power of the programs. These packages use layers, for example, to build complex images, and layer information may be lost in the nonproprietary formats such as TIFF and JPG. However, be sure to save your end result, as a standard TIFF or JPG, or you may not be able to view it in a few years when your software has changed [43].

Currently, GIF and JPG are the formats used for nearly all web images. PNG is supported by most of the latest generation browsers. TIFF is not widely supported by web browsers and should be avoided for web use [43]. PNG does everything GIF does, and better, so expect to see PNG replace GIF in the future. PNG will not replace JPG, since JPG is capable of much greater compression of photographic images, even when set for quite minimal loss of quality.

2.4 Text Compression

Text compression is typically used to save storage or communication costs. It is cheaper ways to communicate compressed text files instead of original text files. Moreover, compressed files are cheaper to store.

For this reasons, various text encoding algorithms have been developed, in addition the corresponding decoding algorithms. Furthermore, a text encoding algorithm takes a text file and generates a shorter compressed file from it. With the compressed file contains all the information necessary to restore the original file, which can be done by calling the corresponding decoding algorithm. The most widely used text compression algorithms are based on Lempel-Ziv techniques [4].

Whitespace compression – Generally, Whitespace compression can be characterised as "removing what we are not interested in" [44]. This technique is technically a lossy-compression technique; it is still useful for many types of data representations we find in the real world. For example, even though HTML is far more readable in a text editor if indentation and vertical spacing is added, none of this "whitespace" should make any difference to the rendering of the HTML document by a Web browser. If you happen to know that an HTML document is destined only for a Web browser then it might be a good idea to take out all the whitespace to make it transmit faster and occupy less space in storage. What we remove in whitespace compression never really had any functional purpose to start with.

Run-Length encoding. - The Run-Length Encoding (RLE) is the simplest widely used lossless compression technique. Like whitespace compression, it is "affordable" -- especially to decode [26]. From then, the idea behind it is that many data representations consist largely of strings of repeated bytes. If repeated bytes are predominant within the expected data representation, it might be adequate and efficient to always have the algorithm specify one or more bytes of iteration count, followed by one character. Moreover, if one-length character strings occur, these strings will require two (or more) bytes to encode them, in other words, 00000001 01011000 might be the output bitstream required for just one ASCII "X" of the input stream. In addition, a hundred "X"s in a row would be output as 01100100 01011000, which is quite good.

Huffman encoding. – The Huffman encoding looks at the symbol table of a whole data set. The compression is achieved by finding the "weights" of each symbol in the data set [16]. There are some symbols occur more frequently than others do; so Huffman encoding suggests that the frequent symbols need not be encoded using as many bits as the less frequent symbols. There are variations on Huffman-style encoding, but the original (and frequent) variation involves looking for the most common symbol, and encoding it using just one bit, say 1. If you encounter a 0, you know you're on the way to encoding a longer variable length symbol.

For instance, let's imagine that apply a Huffman encoding to our local phone-book example (assuming that we have already whitespace-compressed the report).

Huffman encoding is still fairly cheap to decode, cycle-wise. But it requires a table lookup, so it cannot be quite as cheap as RLE, however. The encoding side of Huffman is fairly expensive, though; the whole data set has to be scanned, and a frequency table built up. In some cases a "shortcut" is appropriate with Huffman coding. Standard Huffman coding applies to a particular data set being encoded, with the set-specific symbol table prepended to the output datastream.

Then again, if not just the single data set -- but the whole type of data encoded -- has the same regularities; we can opt for a global Huffman table. However, if we have such a global Huffman table, we can hardcode the lookups into our executables, which makes both compression and decompression quite a bit cheaper (except for the initial global sampling and hard-coding). For instance, if we know our data set would be English-language prose, letter-frequency tables are well known and quite consistent across data sets.

Lempel-Ziv compression. - The most significant lossless compression technique is Lempel-Ziv [5]. What is explained here is LZ78, but LZ77 and other variants work in a similar fashion. The behind idea in LZ78 is to encode a streaming byte sequence using a dynamic table.

What LZ78 does is fill up one symbol table with (hopefully) helpful entries, then write it, clear it, and start a new one. In this regard, a symbol table of 32 entries is still probably too small, since that will get cleared before a lot of reuse of 772 and the like is achieved. But the small symbol table is easy to illustrate.

In typical data sets, Lempel-Ziv variants achieve much better compression rates than Huffman or RLE. On the other hand, Lempel-Ziv variants are very pricey cycle-wise, and can use large tables in memory. Most real-life compression tools and libraries use a combination of Lempel-Ziv and Huffman techniques. Below are a more detailed explanation on LZ77 and LZ78

2.4.1 LZ77

The LZ77 algorithm works by keeping a history window of the most recently seen data and comparing the current data being encoded with the data in the history window. What are actually placed into the compressed stream are references to the position in the history window, and the length of the match. If a match cannot be found the character it is simply encoded into the stream after being flagged as a literal. As of 2004, the most popular LZ77 based compression method is called DEFLATE; it combines LZ77 with Huffman coding [26].

2.4.2 LZ78

While the LZ77 algorithm works on past data, the LZ78 algorithm attempts to work on future data [4]. It does this by forward scanning the input buffer and matching it against a dictionary it maintains. It will scan into the buffer until it cannot find a match in the dictionary. At this point it will output the location of the word in the dictionary, if one is available, the match length and the character that caused a match failure. The resulting word is then added to the dictionary [4].

Though initially popular, the popularity of LZ78 later dampened, possibly because for the first few decades after it was introduced, parts of LZ78 were patent encumbered in the United States. The most popular form of LZ78 compression was the LZRW algorithm, a modification of the LZ78 algorithm made by Terry Welch [19].

2.4.3 LZ78 Example

This example shows the LZ78 algorithm in action, showing the status of the output and the dictionary at every stage, both in encoding and decoding the message [4]. In order to keep things clear, let us assume that we're dealing with a simple alphabet - capital letters only, and no punctuation or spaces. This example has been constructed to give reasonable compression on a very short message; when used on real data, repetition is generally less pronounced, and so the initial parts of a message will see little compression. As the message grows, however, the compression ratio tends asymptotically to the maximum. A message to be sent might then look like the following:

TOBEORNOTTOBEORTOBEORNOT#

The # is a marker used to show that the end of the message has been reached. Clearly, then, we have 27 symbols in our alphabet. A computer will render these as strings of bits; 5-bit strings are needed to give sufficient combinations to encompass the entire dictionary. As the dictionary grows, the strings will need to grow in length to accommodate the additional entries. A 5-bit string gives $2^5 = 32$ possible combinations of bits, and so when the 33rd dictionary word is created, the algorithm will have to start using 6-bit strings. Note that since the all-zero string 00000 is used, and is labelled "0", the 33rd dictionary entry will be labelled 32. The initial dictionary, then, will consist of the following:

# = 00000
A = 00001
B = 00010
C = 00011…….. Z = 11010

2.4.3.1 Encoding

If we weren't using LZ78, and just sent the message as it stands (25 symbols at 5 bits each), it would require 125 bits. We will be able to compare this figure to the LZ78 output later. We are now in a position to apply LZ78 to the message.

Table 2.1 Encoding Table

Symbol:	Bit Code: (= output)	New Dictionary Entry:
T	20 = 10100	28: TO
O	15 = 01111	29: OB
B	2 = 00010	30: BE
E	5 = 00101	31: EO
O	15 = 01111	32: OR (start using 6-bit strings)
R	18 = 010010	33: RN
N	14 = 001110	34: NO
O	15 = 001111	35: OT
T	20 = 010100	36: TT
TO	28 = 011100	37: TOB
BE	30 = 011110	38: BEO
OR	32 = 100000	39: ORT
TOB	37 = 100101	40: TOBE
EO	31 = 011111	41: EOR
RN	33 = 100001	42: RNO
OT	35 = 100011	43: OT#
#	0 = 000000	

Total Length = 5*5 + 12*6 = 97 bits.

In using, LZ78 we have made a saving of 28 bits out of 125 -- we have reduced the message by almost 22%. If the message were longer, then the dictionary words would begin to represent longer and longer sections of text, allowing repeated words to be sent very compactly.

2.4.3.2 Decoding

Imagine now that we have received the message produced above, and wish to decode it. We need to know in advance the initial dictionary used, but we can reconstruct the additional entries as we go, since they are always simply concatenations of previous entries.

Table 2.2 Decoding Table

Bits:		Output:	New Entry:	
			Full:	Partial:
10100	= 20	T	28: T?	
01111	= 15	O	28: TO	29: O?
00010	= 2	B	29: OB	30: B?
00101	= 5	E	30: BE	31: E?
01111	= 15	O	31: EO	32: O? (start using 6-bit strings)
010010	= 18	R	32: OR	33: R?
001110	= 14	N	33: RN	34: N?
001111	= 15	O	34: NO	35: O?
010100	= 20	T	35: OT	36: T?
011100	= 28	TO	36: TT	37: TO? (for 36, only add 1st element)
011110	= 30	BE	37: TOB	38: BE? (of next dictionary word)
100000	= 32	OR	38: BEO	39: OR?
100101	= 37	TOB	39: ORT	40: TOB?
011111	= 31	EO	40: TOBE	41: EO?
100001	= 33	RN	41: EOR	42: RN?
100011	= 35	OT	42: RNO	43: OT?
000000	= 0	#		

The only slight complication comes if the newly-created dictionary word is sent immediately. In the decoding example above, when the decoder receives the first symbol, T, it knows that symbol 28 begins with a T, but what does it end with? The problem is illustrated below. We are decoding part of a message that reads ABABA:

Table 2.3 Decoding Example

Bits:	Output:	New Entry:	
		Full:	Partial:
.			
.			
.			
011101 = 29	AB	46: (word)	47: AB?
101111 = 47	EAB?		

At first glance, this may appear to be asking the impossible of the decoder. We know ahead of time that entry 47 should be ABA, but how can the decoder work this out? The critical step is to note that 47 are built out of 29 plus whatever comes next. 47, therefore, ends with "whatever comes next". But, since it was sent immediately, it must also start with "whatever comes next", and so must end with the same symbol it starts with, namely A. This trick allows the decoder to see that 47 must be ABA.

More generally the situation occurs whenever the encoder encounters the input of the form *cScSc*, where *c* is a single character, *S* is a string and *cS* is already in the dictionary. The encoder outputs the symbol for *cS* putting new symbol for *cSc* in the dictionary. Next it sees the *cSc* in the input and sends the new symbol it just inserted into the dictionary. By the reasoning presented in the above

example this is the only case were the newly-created symbol is send immediately.

2.5 Other Commercialised and Noncommercialised Compression Programs

Today, data compression technology is still one of the most active researches in the scientific community. Yet, because of the significant usefulness and profitability in the marketing sense, most of the compression technologies are commercialized.

Below are some of the examples of the existing commercial and non-commercial compression programs in the market. The table shows the compression software and the technologies behind its compression algorithm [6].

Table 2.4 Other Available Compression Programs In the Market

Program	Author	Used Algorithms
7-Zip	Igor Pavlov	f + LZMA + PPMII + LZ77 + BWT
ABC	Jürgen Abel	BWT
ACB	George Buyanovsky	AC
Archiver	JaboSoft	
ARHANGEL	George Lyapko	
ARJ	ARJ Software	LZSS + Huff
ASH	Eugene Shelwien	CM
BAR	Frank Jennings	BWT
BCArchiver	Jetico, Inc	
BEE	Andrew Filinsky	CM
BioArc	Merlin+ Ltd	f
BJWFLATE	Ben Jos Walbeehm	
BMA	Alexander Cherenkov	f + BWT
BMF	Dmitry Shkarin	
BOA	Ian Sutton	PPM
BSSC	Sergeo Sizikov	f + BWT + DC
BZIP2	Julian Seward	
Cabarc	Microsoft	f + LZX + Huff + SF
Compressia	Yaakov Gringeler	f + BWT + ARI + PPMII
CTW	Frans Willems	CTW
CTXf	Nikita Lesnikov	f + PPMII
DACT	Roy Keene	
DC	Edgar Binder	f + BWT + DC + ARI
DST	Tommaso Guglielmi	LZ77 + PPM + Huff
Durilca	Dmitry Shkarin	f + PPMII
DZIP	Stefan Schwoon	
Emilcont	Berto Destasio	CM
Enc	Serge Osnach	f + PPMII
EPM	Serge Osnach	f + PPMII
ERI	Alexander Ratushnyak	
GRZip	Grebnov Ilya	
GRZipII	Grebnov Ilya	BWT,ST4 + MTF,WFC + ARI
GZip	Jean-loup Gailly	LZ77

HIPP	Bogatov Roman	PPM
ICEOWS	Raphaël Mounier	
JAR	ARJ Software	f + LZSS + Huff
Jcalg1	Jeremy Collake	LZSS
KZip	Ken Silverman	
LHA	Haruyasu Yoshizaki	
Lz2a	Brendan G Bohannon	LZ
LZOP	Markus Oberhumer	LZ
LZPX	Ilia Muraviev	LZP + ARI
M03	Mij4x	
M99	M. A. Maniscalco	
MAR	Xann	LZH BWT PPM
MRP	Ichiro Matsuda	
Ocamyd	Frank Schwellinger	DM
PAC	Gérard Meunier	BWT + LZ77 + Huff
PAQ6	Matt Mahoney	CM
PAQAR	M.Mahoney / A.Ratushnyak	CM
PASQDA	Przemyslaw Skibinski	f + CM
PIMPLE	Ilia Muraviev	
PKZIP	PKWARE Inc.	LZ77
PPMd	Dmitry Shkarin	PPMII
PPMN	Max Smirnov	f + PPM
PPMonstr	Dmitry Shkarin	PPMII
PPMVC	D.Shkarin P.Skibinski	PPMII
PPMY	Eugene Shelwien	PPM
PPMZ2	Charles Bloom	PPM
PSA	Serge Pachkovsky	
Quark	Frederic Bautista	LZ
RK	Malcolm Taylor	f + LZ + PPMZ
RKC	Malcolm Taylor	f + LZ + PPMZ
Rzip	Andrew Tridgell	
SBC	Sami J. Makinen	f + BWT + DC + ARI
ShipInBottle	Alexander Turikov	
Slim	Serge Voskoboynikov	f + PPMII
Squeez	R.Nausedat / S.Ritter	
SRANK	Peter Fenwick	

Stuffit	Allume Systems	
Szip	Michael Schindler	
TC	Ilia Muraviev	LZRW
Transform	Michael Bone	BWT
UFA	Igor Pavlov	
UHARC	Uwe Herklotz	f + PPM + LZP + LZ77 + ARI
UHBC	Uwe Herklotz	BWT + ARI
UPX	M.Oberhumer & L.Molnár	
WinACE	Marcel Lemke	f + LZ77 + Huff
WinHKI	Hanspeter Imp	
WinImp	Technelysium Pty Ltd	f + LZ77 + BWT + Huff
WinRAR	Eugene Roshal	f + LZ77 + PPMII + Huff
WinRK	Malcolm Taylor	f + PPMD+ PPMZ + ROLZ + CM
WinZip	WinZip Computing	LZH + LZRW + SF + Huff + PPMd
WRT	Przemyslaw Skibinski	
YBS	Vadim Yoockin	f + BWT + DC + ARI
ZZIP	Damien Debin	f + BWT

f = Program uses filters, (external) dictionaries and/or file preprocessing.

AC = Associative Coding

ARI = Arithmetic Coding

BWT = Burrows-Wheeler Transform

CM = Context Modeling

CTW = Context Tree Weighting

DC = Distance Coding

DM = Dynamic Markov Modeling

Huff = Huffman

LZ = Lempel-Ziv compression

PPM = Prediction by Partial Match

SF= Shannon-Fano

2.6 Summary

This chapter shows the many existing type of compression algorithms that are design to tackle or handle specific type of datasets. From text to audio and video, each has its own type of compression technique and motivation. As different technique will yield different results, the selection of compression techniques is largely dependent on the user and purpose of the compressed data.

In summary, higher compression rates can be achieved by eliminating details in data - referred to as lossy compression, data size at the expense of data resolution or granularity (image, video or audio quality in laymen terms).

It would not be acceptable for lossy compression to be applied to scientific datasets, as there will be loss of data, which may be the clue for scientific discoveries and further analysis on higher details on data obtained.

Most codecs have compression features built-in as part of increasing user's acceptance by improving storage and transmission performance. Such codecs is at the expense of computing or processing power, however due to high speed computing power now is cheaply available such as Intel Pentiums and AMD Athlons are the commodity processors available that runs on Ghz frequencies.

CHAPTER 3: Development Methods

Generally, scientific datasets are a group of data that gathered through measurements in field of interested, or by simulation. The data has the following characteristics:

- Multiple observing objects. Scientific measurement normal will cover a few objects of interest. For example, when measuring weather, cloud, wind, landscape, and temperature will be observed.
- Multiple dimension of measurement. When collecting data for scientific analysis, different aspect of the object will be observed, in different units of measurement. For example, the measurement taken on wind in weather analysis includes the speed and the direction of the wind.
- High resolution. A good resolution is important for scientific analysis. Poor resolution will lead to inaccurate conclusion. In order to obtain a good resolution on the data measure, larger datasets will be generated.
- High precision. Generally, scientific dataset is measure with certain precision of decimal points. Better precision will lead to more accurate result.

From the characteristic above, we could know that the scientific datasets are random and of different scales. Some data will vary in large quantity, while some data might vary in decimal points value. Different field of interests also could lead to different datasets in similar phenomena.

Gathering datasets will be useful for the following:

- Scientific datasets always represent statistical measurement of a phenomenon. With this datasets, the characteristic of the phenomenon could be studied.
- Modeling of the phenomenon. With the characteristic of the phenomenon, its physical or computer model could be formed. Forming the model could help to simulate the phenomenon with different condition.
- Prediction and further analysis. Modeling of a phenomenon will allow us to simulate a situation before we meet it. It is useful if the situation is hazardous. It also allows us to gather datasets for phenomenon that we could not observe at closer distance, for example, hurricane.

There are also other types of datasets available, such as text datasets, numerical datasets, alphanumeric datasets and etc. But they are either too narrow in term of data type variety or contain too many redundant data type or characteristic, therefore the scientific datasets were the most suitable candidate due to its random characteristic and size. The scientific dataset is the only dataset type that covers all aspect of all the other dataset types.

Here, there are three different categories of scientific datasets chosen from various area of aspect in this project. These three categories of scientific datasets representing real life data used in the industry serve as analytical data for research purposes.

The scientific datasets are obtaining from NCBI mapviewer [7], Fire Dynamic Simulator (FDS) [8] and Water Quality Data (WQD) [9].

Table 3.1 : Datasets Comparisons

Datasets type	NCBI mapviewer	Fire Dynamic Simulator	Water Quality Data
What they use	Combination of graphical and alphanumerical	Combination of binary data and numeric	Combination of numeric and alphabets
Where they use	FASTA or Gen Bank	Research in fire simulation	Analytical data processing
Why they use	Covers a broad spectrum of data capturing	Looks good in presentation	Maximize memory management
How they use	Specialised software and hardware	Specialised software	Specialised software
Disadvantages	Small data size usually	Large and random in data size	Medium data size
Advantages	Small data size	Encapsulate more information and random in nature	More random in nature than numerical datasets

The first dataset chosen for this project are obtaining from National Center for Biotechnology Information (NCBI) organisation. The organization was established in 1988 as a national resource for molecular biology information, NCBI purposes are to create public databases, conducts research in computational biology, develops software tools for analyzing genome data, and disseminates biomedical information – this serve as for the better understanding of molecular processes affecting human health and disease [7].

There are 2 different formats for this scientific datasets on this category which is FASTA and GenBank (GB). GenBank is the NIH database maintained and distributed by NCBI that stores all known public DNA sequences. The sequence data are submitted to GenBank from individual scientists from around the world, as well as from the large centers especially involved in the Human Genome Project. There are number of DNA sequences stored in the GenBank database, from all organisms, potentially continues to grow at a rapid rate.

The Fire Dynamics Simulator has been under development for almost about 25 years. At National Institute of Standards and Technology (NIST), Howard Baum and Ronald Rehm laid the theoretical groundwork for the model and devised the basic numerical solvers [8].

The name of the program is known as NIST Fire Dynamics Simulator or FDS. FDS is a Fortran 90 computer program that solves the governing equations of fluid dynamics, and Smokeview is a companion program written in C/OpenGL programming language that produces images and animations of the results. The revision are from Version 1 of FDS was publicly released in February 2000, version 2 in December 2001, and version 3 in November 2002. The present version of FDS is 4, released in July 2004 [8].

The Fire Dynamic Simulator is a computational fluid dynamics (CFD) model of fire-driven fluid flow. The software solves numerically a form of the Navier-Stokes equations appropriate for low-speed, thermally-driven flow with an emphasis on smoke and heat transport from fires.

Fire Dynamic Simulator is a simulator but the dataset generated is a real data for scientific analysis. Therefore, the datasets produced is valid as scientific data.

The dataset from Fire Dynamic Simulator version 4 is chosen for numerical scientific dataset. This simulator matches the above mentioned characteristics. The following lists the consideration point:

- It generates datasets on multiple aspects, which are density, pressure, heat, chemical composition, and velocity.
- Each dataset is measured with high resolution. The measurement is taken on hundred of thousands, to millions of grid cell in a given space, example, a room. The time steps are from thousands to hundreds of thousand.

Additionally, as in most datasets, the data is recorded in binary format. This is a compact format and normally does not work well with text compression algorithm.

The third category of the dataset is obtaining from U.S. Environmental Protection Agency (EPA) on Water Quality Data. The U.S. Environmental Protection Agency (EPA) maintains two data management systems, which contains water quality information for the nation's waters: the Legacy Data Center (LDC), and STORET. The LDC is a static, archived database and STORET is an operational system actively being populated with water quality data [9].

The STORET (short for STOrage and RETrieval) is a repository for water quality, biological, and physical data and is used by state environmental agencies, EPA and other federal agencies, universities, private citizens, and many others.

Each datasets sampling result in the LDC and in STORET is accompanied by information on where the sample was taken (latitude, longitude, state, county, Hydrologic Unit Code and a brief site identification), when the sample was gathered, the medium sampled (e.g., water, sediment, fish tissue), and the name of the organization that sponsored the monitoring [9]. Besides that, STORET contains information on why the data were gathered; sampling and analytical methods used; the laboratory used to analyze the samples; the quality control checks used when sampling, handling the samples, and analyzing the data; and the personnel responsible for the data.

The datasets chosen for this project are supplied to EPA before 1999 were all placed in Legacy STORET. This system, designed in the 1960s, was a pioneer in the long term archival of field water monitoring results [9].

There is quite a number of compression and decompression algorithms exist nowadays, since the first publish of Lempel-Ziv algorithm on 1977, the LZ77 algorithm [5]. The famous commercial software includes pkzip, winzip, and winrar for windows platform, and compression algorithm famous on UNIX platform, the gzip/zlib. The gzip/zlib algorithm is a variant of LZ77 algorithm, with enhancement on its general purpose compression, i.e., on text file or on file with random data, like binary file.

There are also a lot of algorithms that work on specific dataset, for example, JPEG, GIF, PNG format that compress picture. These kinds of algorithms have more efficiency than LZ77 or LZ78 algorithm on picture compression, but they only work well on specific dataset, which is images for scientific dataset with binary format as discuss previously, we would need to find a more generic purpose algorithm.

The gzip/zlib algorithm is chosen for it is the variant of LZ algorithm, and also its history in UNIX platform. Another algorithm, the bzip compression, also chosen as it is also well known on UNIX platform [18]. The bzip algorithm using different approach than the LZ77, therefore, it is suitable for comparing the efficiency of

compression among different approach. A benchmarking algorithm, LZRW compression, is chosen, as it is also a variant of LZ77 compression.

The gzip/zlib and bzip algorithm are used for general compression on UNIX system. This generally suite the purpose of this approach as the contents of the dataset will also be random data.

Table 3. 2 Compression Algorithm Comparisons

Algorithm	Zlib	LZRW	Bzip
Founder	Jean-loup Gailly (compression) Mark Adler (decompression)	Dr. Ross N. Williams	Julian Seward
Year Developed	2005	1990	1996
Characteristic	Lossless data compression	Lossless data compression	Lossless data compression
Usage	Generic text	Generic text	Generic text
Key attributes/properties	Deflation technique (combination of LZ77 and Huffman coding)	Statistical modelling technique	Burrows-Wheeler block-sorting text Technique

Based on the numerous compression algorithms, these three algorithms have been chosen. The selection for these algorithms is based on the year of development, availability and its open source for further development.

3.1 LZW Explained

The original Lempel Ziv approach to data compression was first published in in1977 [5]. Terry Welch's refinements to the algorithm were published in 1984. The algorithm is surprisingly simple [19]. In a nutshell, LZRW compression replaces strings of characters with single codes. It does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters.

The code that the LZW algorithm outputs can be of any arbitrary length, but it must have more bits in it than a single character [18]. The first 256 codes (when using eight bit characters) are by default assigned to the standard character set. The remaining codes are assigned to strings as the algorithm proceeds. The sample program runs as shown with 12 bit codes. This means codes 0-255 refer to individual bytes, while codes 256-4095 refers to substrings.

3.1.1 Compression

The LZW compression algorithm in its simplest form is shown in Figure 3.1. A quick examination of the algorithm shows that LZW is always trying to output codes for strings that are already known. And each time a new code is output, a new string is added to the string table.

```
Routine LZW_COMPRESS

   STRING = get input character
   WHILE there are still input characters DO
      CHARACTER = get input character
      IF STRING+CHARACTER is in the string table then
         STRING = STRING+character
      ELSE
         output the code for STRING
         add STRING+CHARACTER to the string table
         STRING = CHARACTER
      END of IF
   END of WHILE
   output the code for STRING
```

Figure 3.1 The Compression Algorithm

A sample string used to demonstrate the algorithm is shown in Figure 3.2. The input string is a short list of English words separated by the '/' character. Stepping through the start of the algorithm for this string, you can see that the first pass through the loop, a check is performed to see if the string "/W" is in the table. Since it isn't, the code for '/' is output, and the string "/W" is added to the table. Since we have 256 characters already defined for codes 0-255, the first string definition can be assigned to code 256. After the third letter, 'E', has been read in, the second string code, "WE" is added to the table, and the code for letter 'W' is output. This continues until in the second word, the characters '/' and 'W' are read in, matching string number 256. In this case, the code 256 is output, and a three character string is added to the string table. The process continues until the string is exhausted and all of the codes have been output.

Input String = /WED/WE/WEE/WEB/WET			
Character Input	**Code Output**	**New code value**	**New String**
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE
/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T		

Figure 3.2 The Compression Process

The sample output for the string is shown in Figure 3.2 along with the resulting string table. As can be seen, the string table fills up rapidly, since a new string is added to the table each time a code is output. In this highly redundant input, 5 code substitutions were output, along with 7 characters. If we were using 9 bit codes for output, the 19 character input string would be reduced to a 13.5 byte output string. Of course, this example was carefully chosen to demonstrate code substitution. In real world examples, compression usually doesn't begin until a sizable table has been built, usually after at least one hundred or so bytes have been read in.

3.1.2 Decompression

The companion algorithm for compression is the decompression algorithm. It needs to be able to take the stream of codes output from the compression algorithm, and use them to exactly recreate the input stream. One reason for the efficiency of the LZW algorithm is that it does not need to pass the string table to the decompression code. The table can be built exactly as it was during compression, using the input stream as data. This is possible because the compression algorithm always outputs the STRING and CHARACTER components of a code before it uses it in the output stream. This means that the compressed data is not burdened with carrying a large string translation table.

```
Routine LZW_DECOMPRESS

   Read OLD_CODE
   output OLD_CODE
   WHILE there are still input characters DO
      Read NEW_CODE
      STRING = get translation of NEW_CODE
      output STRING
      CHARACTER = first character in STRING
      add OLD_CODE + CHARACTER to the translation table
      OLD_CODE = NEW_CODE

      END of WHILE
```

Figure 3.3 The Decompression Algorithm

The algorithm is shown in Figure 3.3. Just like the compression algorithm, it adds a new string to the string table each time it reads in a new code. All it needs to do in addition to that is translate each incoming code into a string and send it to the output.

Figure 3.4 shows the output of the algorithm given the input created by the compression earlier in the article. The important thing to note is that the string table ends up looking exactly like the table built up during compression. The output string is identical to the input string from the compression algorithm. Note that the first 256 codes are already defined to translate to single character strings, just like in the compression code.

Input Codes: / W E D 256 E 260 261 257 B 260 T				
Input/ NEW_CODE	**OLD_CODE**	**STRING/ Output**	**CHARACTER**	**New table entry**
/	/	/		
W	/	**W**	**W**	**256 = /W**
E	**W**	**E**	**E**	**257 = WE**
D	**E**	**D**	**D**	**258 = ED**
256	**D**	**/W**	/	**259 = D/**
E	**256**	**E**	**E**	**260 = /WE**
260	**E**	**/WE**	/	**261 = E/**
261	**260**	**E/**	**E**	**262 = /WEE**
257	**261**	**WE**	**W**	**263 = E/W**
B	**257**	**B**	**B**	**264 = WEB**
260	**B**	**/WE**	/	**265 = B/**
T	**260**	**T**	**T**	**266 = /WET**

Figure 3.4 The Decompression Process

3.2 Programming Approaches

There are several programming approaches such as C/C++, assembly language, Java, Delphi and Visual Basic to develop a simulator. These include procedural approach, structured approach and object-oriented approach that are widely used in developing a simulator. For this project, object-oriented approach was adapted for it facilitates a more usability features over other approaches. For example, in object-oriented approach, the codes are more reusable, they can also be overloaded with more than one method. It also features polymorphism which other approaches cannot place on par.

There are also many programming tools that can be used for this project but C programming language was chosen based on its fame in powerful low end programming features and simplicity in usage.

3.3 Evaluation Approaches

In chapter 6, a list of graphs and tables will be shown and discussed for the efficiency and effectiveness of each algorithm, in terms of compression ratio, compression time consumed and data throughput.

The motivations behind evaluating these areas are due to the notion that, these areas are the most important areas pertaining to the performance of a compression technology. There might be other issue that may affect the performance, but as in a general observation, these areas that we are testing on,

covers 95% of the performance issues.

Data throughputs were also evaluated to prove that by applying compression on the subjected dataset, it would reduce the time delay and increases the total throughput.

The dataset used will be one of the data file generated by the FDS program, in binary data format. The sizes of the datasets used are approximate to 5 MB, 10 MB, 20 MB, and 30 MB. (1 MB = 1048576 bytes of data).

3.4 Summary

This chapter covers the detail of the simulation. It discussed the reason why simulation is used to simulate the compression processes as well as the approach used to develop a simulator. It also covers in details of the LZW compression technology.

The compression simulator will be developed using the object-oriented approach which is C/C++ programming language. The C/C++ programming language will be used as the tools to develop the simulator. The next chapter will discuss the analysis of the simulator architecture.

CHAPTER 4: System Analysis

This chapter provides an in depth analysis of the compression evaluation simulator. The chapter begins with the overview of the simulation concept. The aim is to provide an explanation of the simulator architecture.

The following section discusses the simulator architecture. It is followed by an analysis of components as well as the requirement to develop the simulator.

The final section summarises the details of this chapter. It summarises the analysis of simulator as well as the simulator.

4.1 Simulation Concept

The main concept of this research is to simulate the activity of transmitting a compressed scientific dataset over a network environment as to evaluate the performance of various compression algorithms.

The simulation test is conducted on a clean installed computer so that is no other software or viruses that will afftect the performance of the result.

4.2 Simulation Architecture

Compression evaluation simulator is a flexible test bed for studying and evaluating the performance of compression technology and algorithm. The simulator is written in C Language whereby it is developed in object-Oriented programming approach. The simulator architecture is based on a client-server approach.

The client main program is the main component of the entire simulation, it contain the testing compression algorithm that will be use for evaluation. It also performs all the compression and decompression of the subjected datasets, and also time logging for evaluation purposes.

4.3 Simulation Requirement

This research require a scientific dataset to be tested using various compression algorithm. A scientific dataset are to be selected based on its characteristic as describe in chapter 3. This simulation requires WAN connectivity such as Internet.

The selections of compression algorithm are selected based on its historic background and popularity. Thou all compression algorithm may derive from the same theory, in specific; they are different in term of compression logic.

4.4 Simulation Limitation

This simulation evaluate only on three generic compression algorithm, it does not cover other type-specific compression algorithm such as video, audio or graphics. This simulation only test on one Internet link for WAN data transmission test.

4.5 Platform and System Specification

Note that the measurements are done on personal computer with the following specification:

Client PC:

- Pentium 4, 2.0 GHz processor
- 256 MB RAM
- Windows XP

Server PC:

- Pentium 4, 2.0 GHz processor
- 256 MB RAM
- Windows XP

LAN Connection: Ethernet 100 MB.

WAN Connection: 1 Mbps

The above mentioned on the system specification chosen because it fulfill the memory and processing speed of the datasets. The testing machine CPU specification can influence the result of the testing on datasets. The overall performance of the testing machine to produce a significant result of output is affected by hardware and network capability.

4.5.1 System Requirement

The system requirement to develop the compression performance evaluation simulator is categorized into functional requirement and non-functional requirement. The following section will discuss the functional requirement and non-functional requirement of the compression performance evaluation simulator.

4.5.1.1 Functional Requirement

This section describes the functional requirement of the compression performance evaluation simulator.

- The simulator will support different types of compression algorithm.
- Time used in all processes and tests will be displayed.
- The simulator allow user to configure the evaluation tests.
- The simulator allow user to add in more compression algorithm for evaluation purposes.

4.5.1.2 Non-Functional Requirement

This section describes the non-functional requirement of the compression performance evaluation simulator.

- **Reliability**
  - The system should be reliable in performing its simulation functions and network operations. For example, whenever a compression is executed, the system should be able to perform some functionality or generate some message to inform the user what is happening.

- **Usability**
  - The system should be easy to operate.
  - The test results should be easy to read and understand

  .
- **Flexibility**
  - The system should have the capabilities to take in new compression algorithm into the system.

4.6 Analysis

In this project, the compression algorithm will be analyse based on these 6 categories which is specifically internally developed for scientific dataset:

1. Speed of compression vs. Size of data sets
   - This test will tell us how the compression algorithm fair in term of compression speed, when the size of the datasets increases.
2. Size after compression vs. Size of data sets
   - This test will tell us the threshold of each compression algorithm.
3. Delay time (against Raw data transmission) vs. Size of data sets
   - This test is to ascertain the total time taken from compression to data transmission of each algorithm.
4. Compression Ratio
   - This test shows the how much can a datasets be compressed by each algorithm.

5. Data Rate
    - This test is to show the rate of compression against the dataset size.
6. Data Transmission time
    - This test will simulate the transmission time of a compressed datasets across a networked environment.

These 6 sets of results will tell us how each algorithm fair in each testing category, therefore given us a clear view of choosing suitable algorithm.

The choice of algorithm should base on both its compression ratio on the dataset, and also the compression speed on this dataset. The concern of the project is not only on the data rate for the transmission. It also seeks for a solution for better storage on local system. The local system has limited storage available compare to the server running on super computer. Therefore, the algorithm should have a good compression ratio on the dataset.

4.7 Summary

This chapter covers the major analysis on the key features of the compression performance evaluation simulation. The overall architecture of the simulation is analysed in order to find out how can other compression algorithm can be introduce into this simulator to further study and investigate the behaviors of a compression technology.

The procedure of testing and evaluating the compression technologies also covered in this chapter. It provides a good understanding of the simulation architecture as well as the steps required to evaluate the technologies and algorithm.

This chapter concludes by presenting the functional requirements and non-functional requirements of the simulator. Details of system design will be discussed in the following chapter.

CHAPTER 5: System Design and Implementation

The purpose of this project is to analyse and search for a solution in scientific dataset analysis. The approach used in this project is to compress the data for transfer and storage. Compression reduces the size of the total amount transfer, and also occupies less storage space.

The Fire Dynamic Simulator (FDS) does not send data to any port. Instead, it will write the output to local disk. Therefore, in this project, it is not able to simulate the port forwarding. Instead, we will use a few files for demonstration.

To compare the performance result, we will use 2 types of compression algorithm against a benchmarking compression algorithm to compress a binary file. These results are then to be comparing with same compression over an ASCII file. With this, it provides a frame of reference against the compression algorithm, which in term yields a clearer picture on the compression performance.

The procedures of simulation are as below:-

1. Transfer without compression using ASCII file
2. Transfer with compression using ASCII file
3. Repeat step 1 using binary file
4. Repeat step 2 using binary file
5. Repeat step 1 using NCBI file
6. Repeat step 2 using NCBI file
7. Repeat step 1 using Water Quality text file
8. Repeat step 2 using Water Quality text file

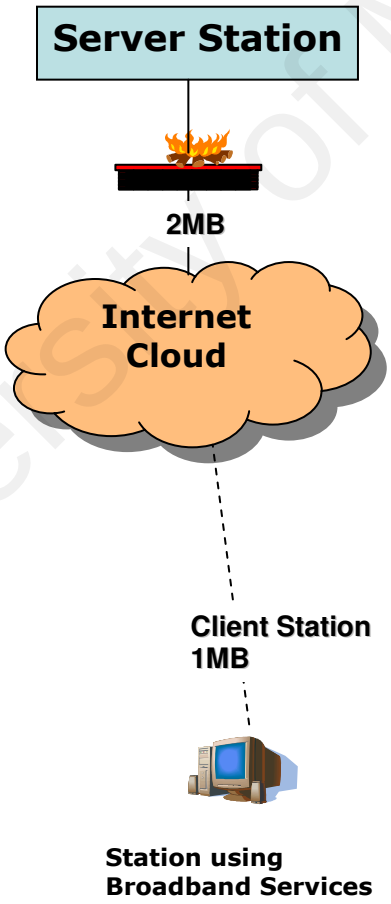
Step 3 until 8 are to be repeated with the following arrangement.

- with zlib v.1.2.3 compression algorithm
- with bzip v1.0.3 compression algorithm
- with LZRW3-A compression algorithm

The results are calculated to evaluate the performance for each one of the algorithms.

The physical connectivity of the system design and implemention is shown on Figure 5.1

Figure 5.1 Network Connectivity of the System Implementation



5.1 Flow diagram

5.1.1 Client Program: General Flow

This flow applicable is for all 3 algorithms.

5.1.1.1 Client Program Input/Output

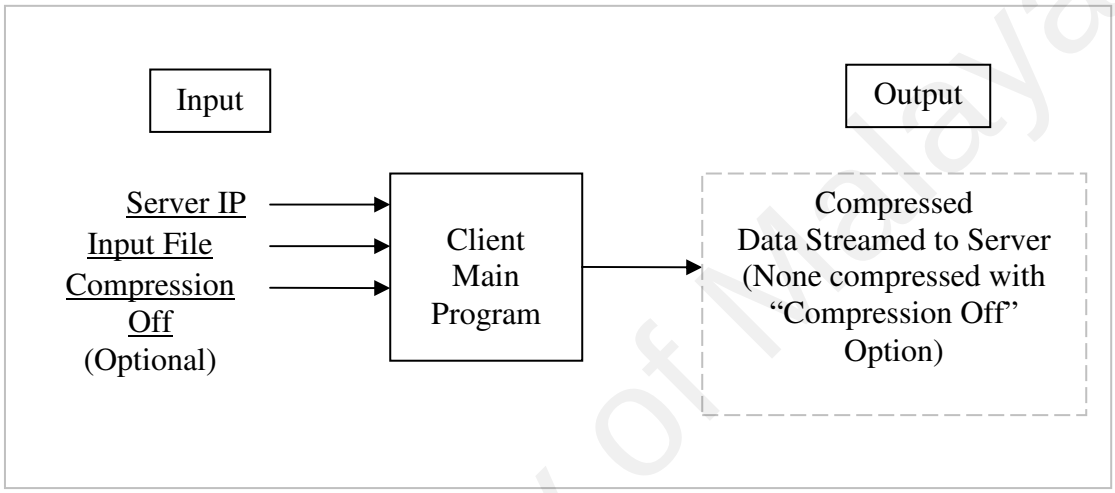


Figure 5.2 Client Program Input/Output

5.1.1.2 Client Program Flow:

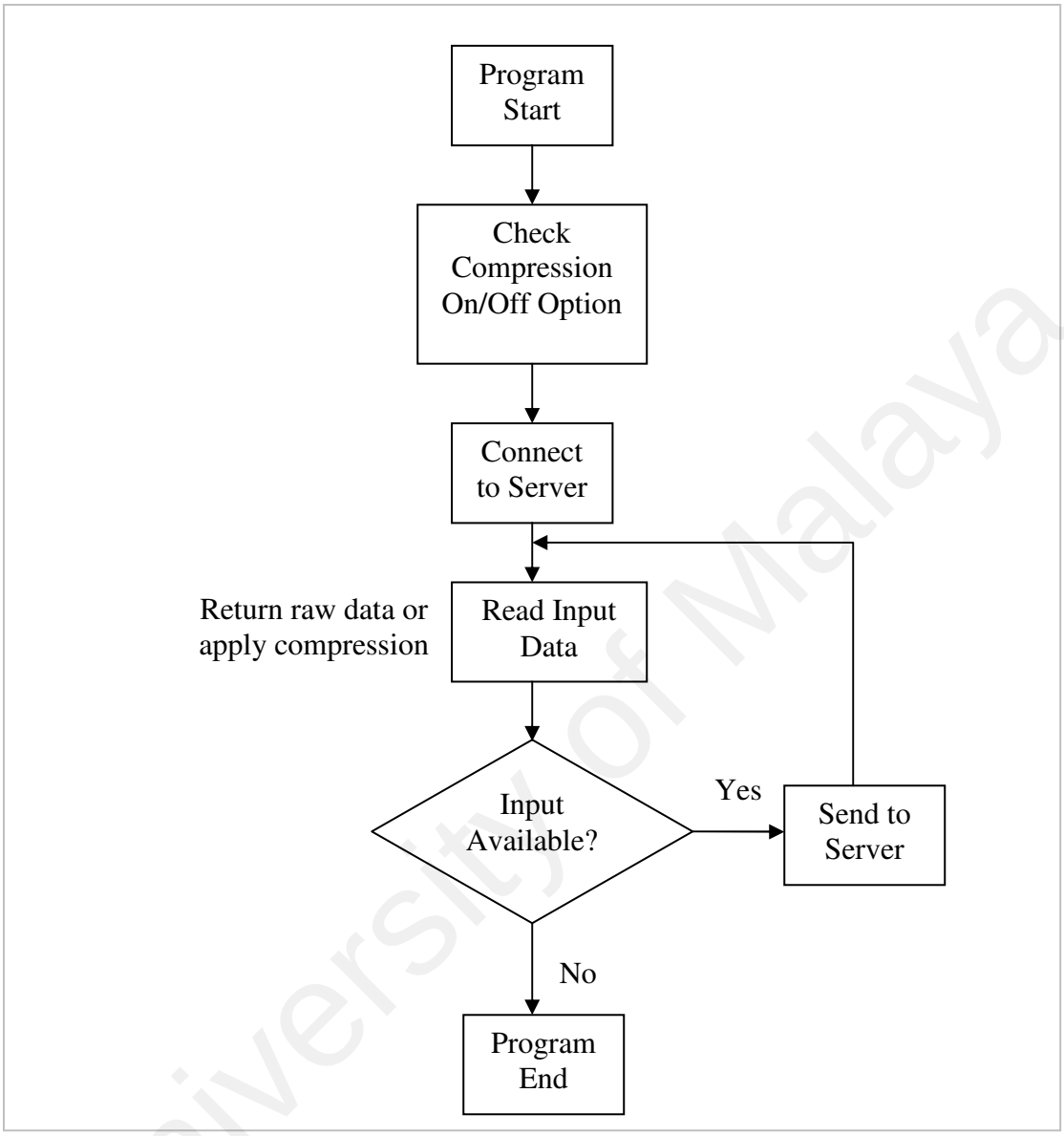


Figure 5.3 Client Program Flow

5.1.1.3 Client Read File Flow:

This is where the compression will apply, if selected. This flow only discusses the Compression process.

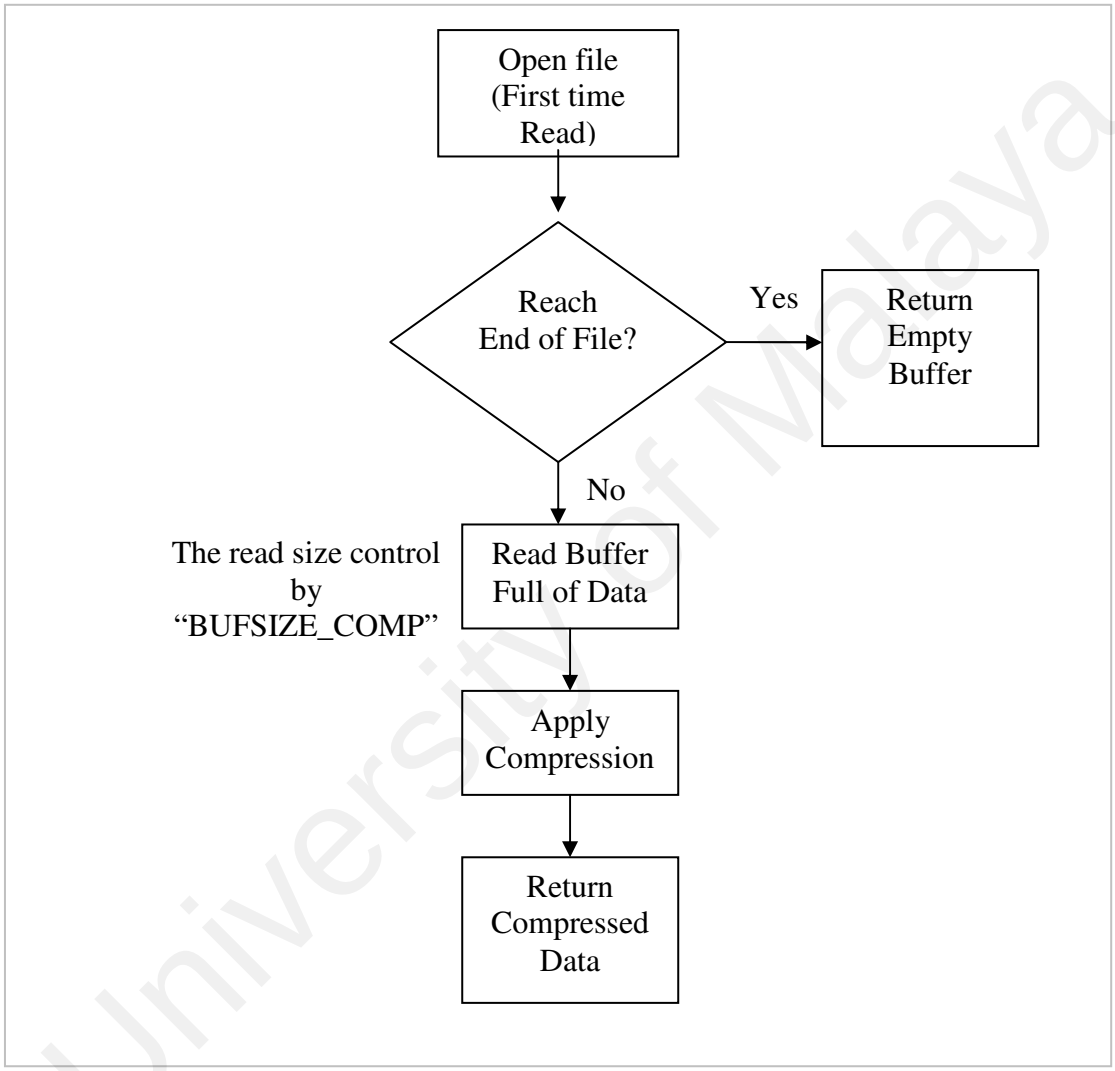


Figure 5.4 Client Read File Flow

Some Notes on the compression control:

Zlib: There is no control parameter available for Zlib compression algorithm.

Bzip: The following parameter is applied -

- blockSize100k = 5: Used for the buffer allocation for compression algorithm. The size should be 4 times larger than the input buffer size.
- Verbosity = 0: Control the information displayed during compression. 0 indicates no information displayed.
- Work Factor = 30: Control parameter for algorithm to choose between slower approach or fast but less efficient approach. This value is optimum after tested, which also suggested by the author of the algorithm.

LZRW: There is no control parameter available for LZRW compression algorithm.

5.1.2 Server Program: General Flow

This flow is applicable for all 3 algorithms.

5.1.2.1 Server Program Input/Output:

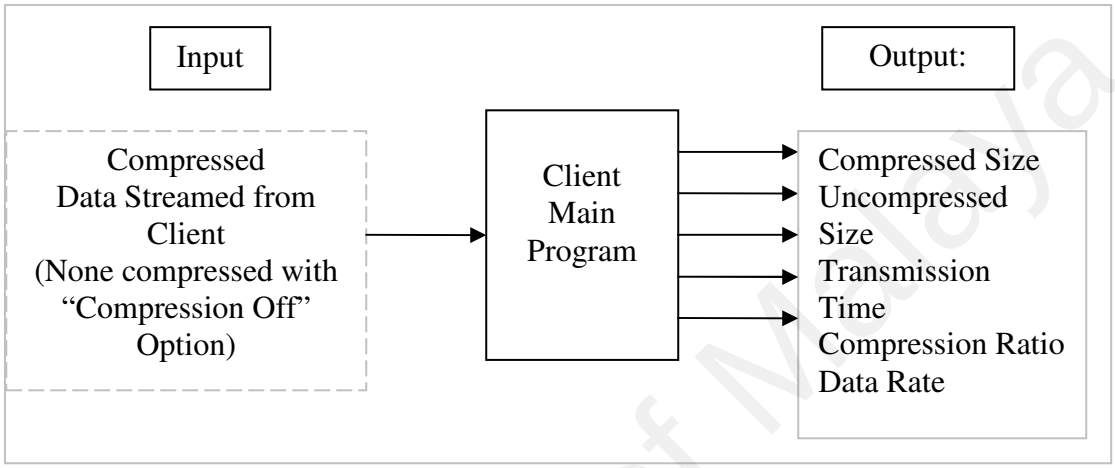


Figure 5.5 Server Program Input/Output

Note:

1. Compressed Size:

   This number indicates the total number of data, in bytes, received from client.

2. Uncompressed Size:

   a. For compressed data received from client, this is the total number of data, in bytes, after decompression process.

   b. For raw data received from client, this number will be the same as Compressed Size, indicating no compression performed.

3. Transmission Time

   This time, in milliseconds, is measured right before the first byte is received, until the last byte is received. There is no decompression done in between. This gives a more precise measurement on the compression and transmission time.

4. Compression Ratio

   The ratio is calculated with the formula:

   Compression Ratio = [1 - (Compressed Size / Uncompressed Size)] * 100 %

5. Date Rate

   The Data Rate is calculated with the formula:

   Date Rate = Uncompressed Size / Transmission Times

5.1.2.2 Server Program Flow:

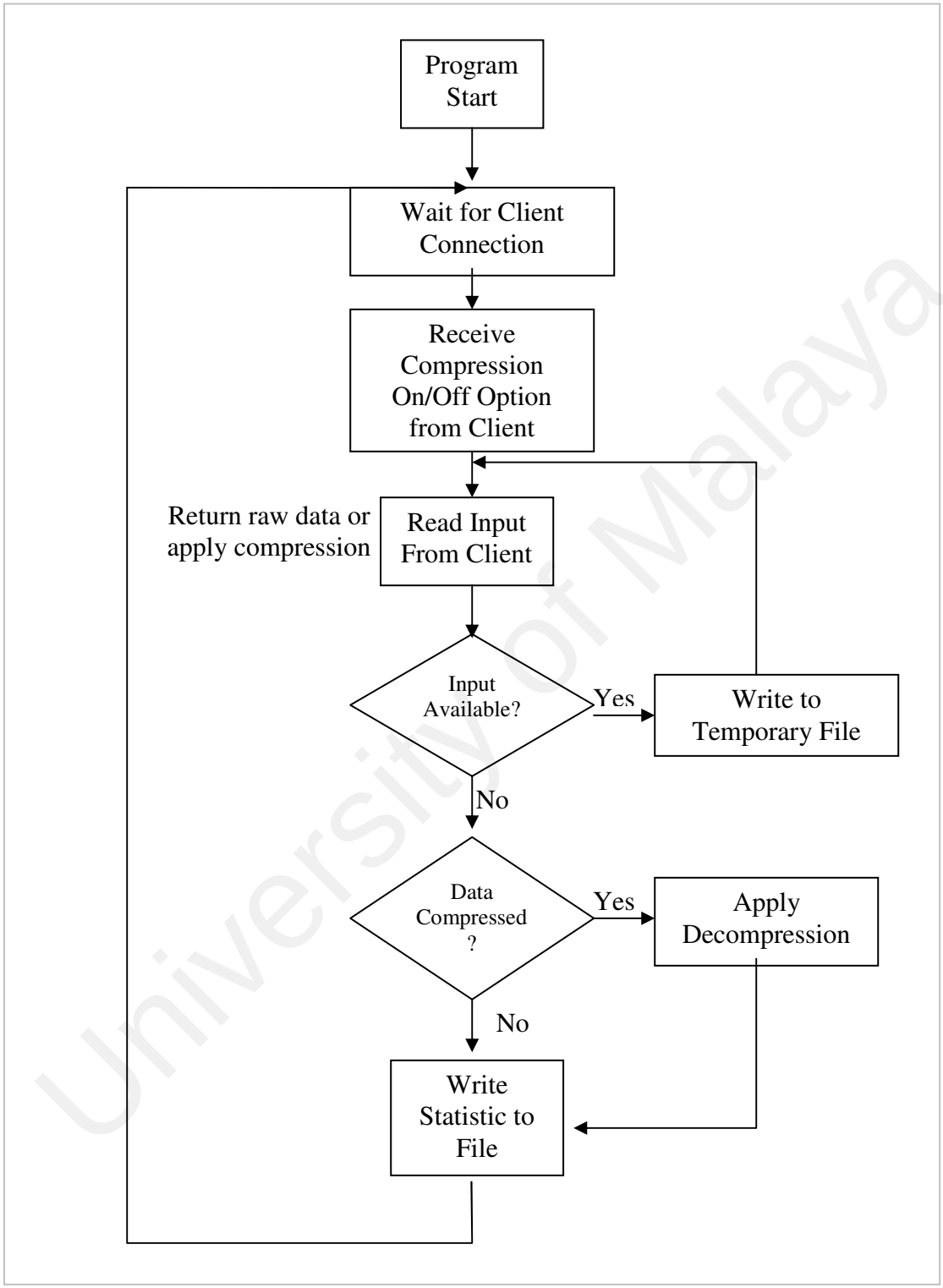


Figure 5.6 Server Program Flow

The decompression process uses the same parameter as the compression process:

Zlib: There is no control parameter available for Zlib compression algorithm.

Bzip: The following parameter is applied -

- blockSize100k = 5: Used for the buffer allocation for compression algorithm. The size should be 4 times larger than the input buffer size.
- Verbosity = 0: Control the information displayed during compression. 0 indicates no information displayed.
- Work Factor = 30: Control parameter for algorithm to choose between slower approach or fast but less efficient approach. This value is optimum after tested, which also suggested by the author of the algorithm.

LZRW: There is no control parameter available for LZRW compression algorithm.

5.2 Summary

This chapter covers the major design issues for the compression evaluation simulator. This includes an overview of the system architecture, which focus on the simulator design and implementation.

CHAPTER 6: System Testing

This chapter details the implementation and testing aspects of the scientific dataset in the simulator. It first begins with the implementation of the component classes.

The second section focuses on the testing of the different real life scientific dataset on the simulator. This section will describe the testing for specific scientific dataset according to the 3 category which is NCBI dataset, fire dynamic dataset and water quality dataset.

The final section of this chapter summarises the details of this chapter.

The testing is done in 6 parts:

1. Speed of compression vs. Size of data sets
2. Size after compression vs. Size of data sets
3. Delay time (against Raw data transmission) vs. Size of data sets
4. Compression Ratio
5. Data Rate
6. Data Transmission time

6.1 Speed of compression vs. Size of datasets

These table and graph show the relationship betweens the Speed of compression, in millisecond, for each algorithm, against the Data Size of the input file, in number of Byte.

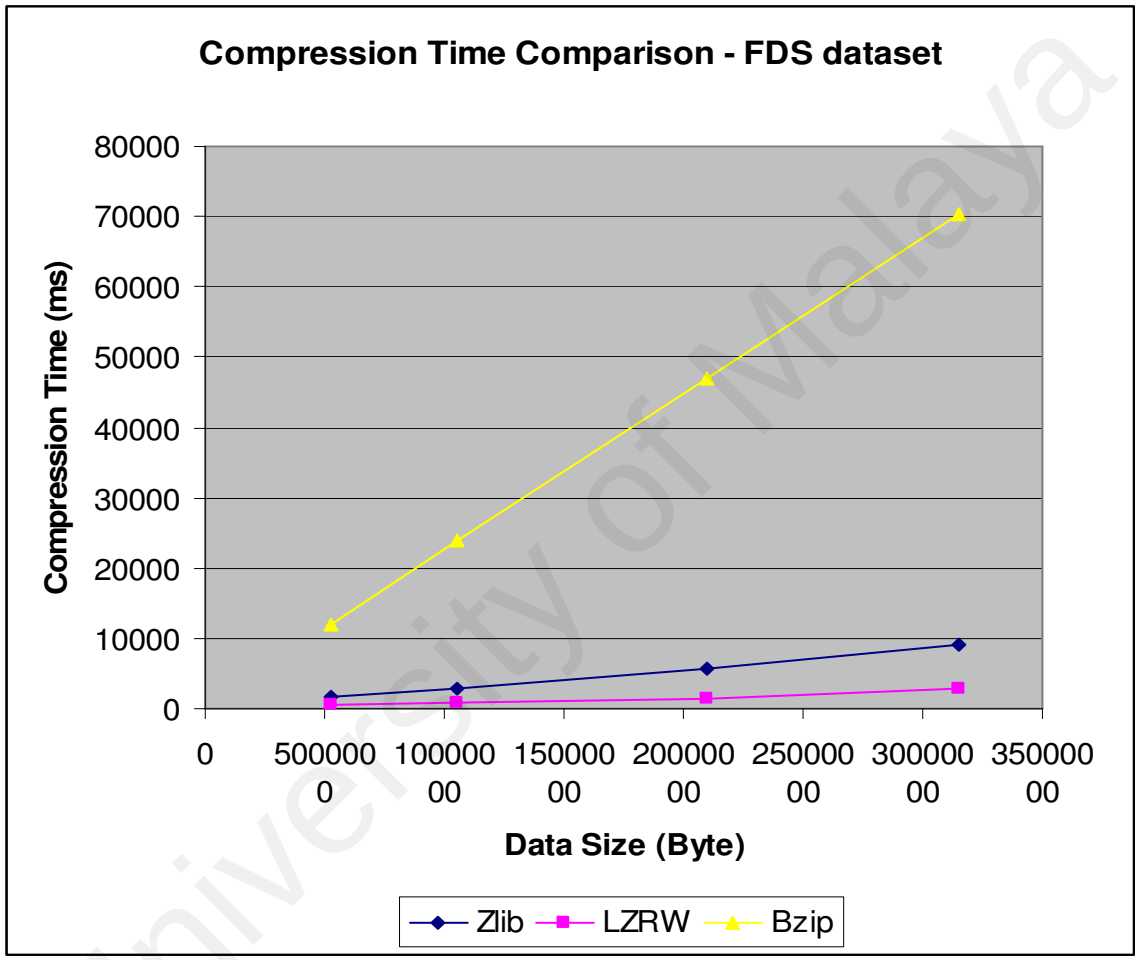


Figure 6.1 Compression Time - FDS dataset

Table 6.1 Compression Time - FDS dataset

Compression Time

Data Size	Zlib	LZRW	Bzip
5275466	1764.98	486.37	11948.63
10546978	2750.68	825.17	23783.79
21023274	5685.29	1488.94	46937.58
31499570	9176.05	2850.35	70387.92

The Figure 6.1 and Table 6.1 show result on FDS dataset of the test perform on three algorithms for compression time.

Figure 6.2 Compression Time - NCBI dataset

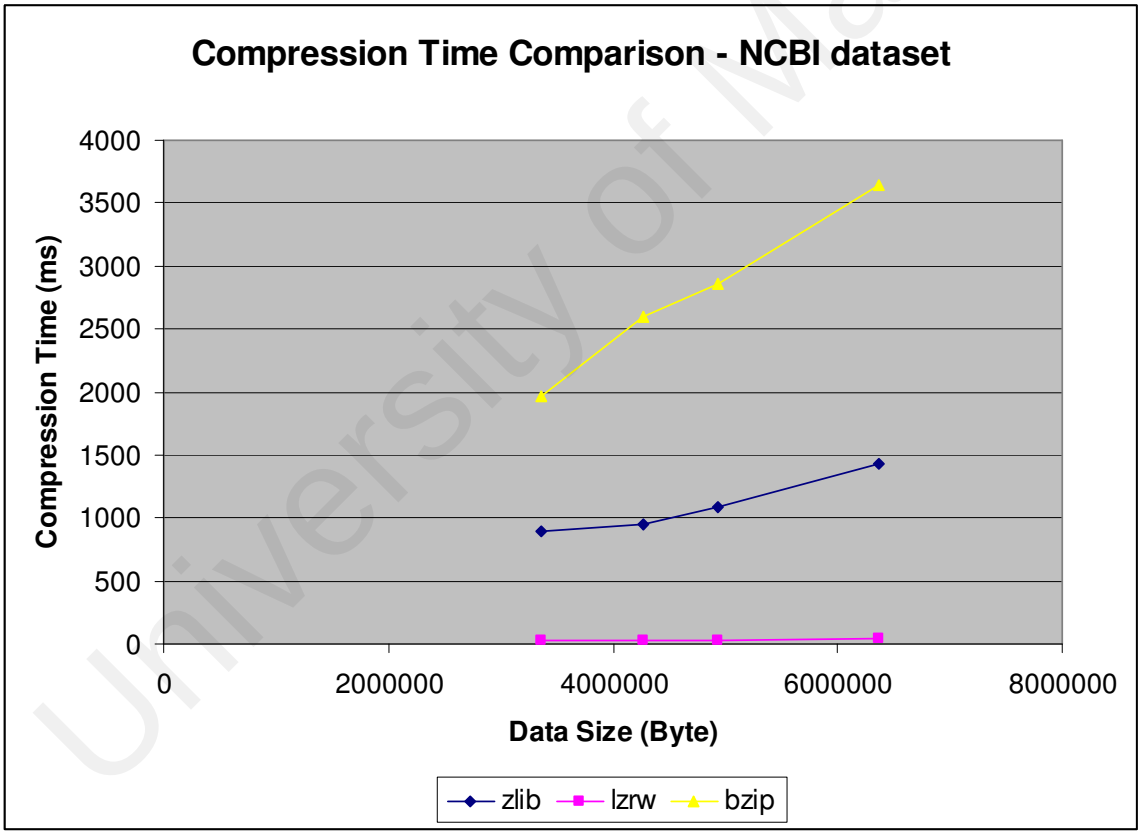


Table 6.2 Compression Time - NCBI dataset

Compression Time

Data Size	Zlib	LZRW	bzip
3365257	899.06	25.18	1962.67
4265064	949.2	27.05	2603.33
4933488	1087.4	29.78	2864.04
6360548	1435.41	43.19	3644.67

The Figure 6.2 and Table 6.2 show result on NCBI dataset of the test perform on three algorithms for compression time.

Figure 6.3 Compression Time – Water Quality dataset

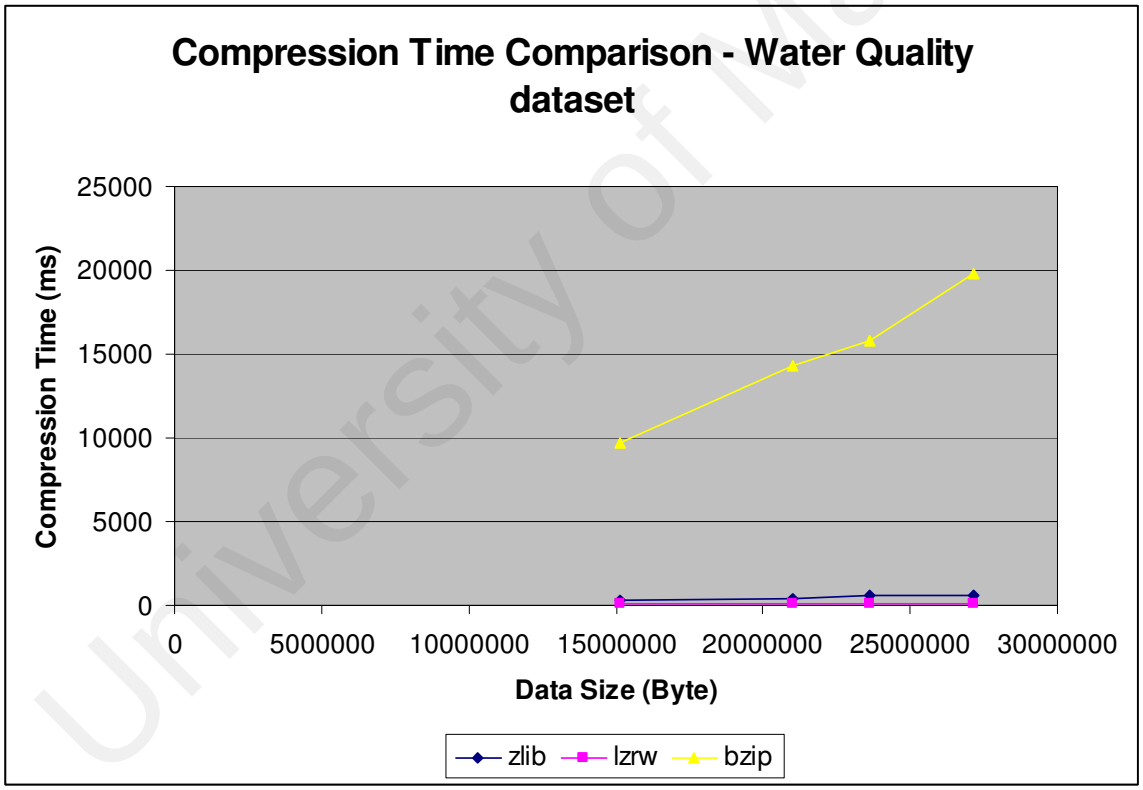


Table 6.3 Compression Time – Water Quality dataset

Compression Time			
Data Size	zlib	LZRW	bzip
15148619	330.51	55.05	9720.67
21004413	360.4	92.33	14287.33
23609061	597.67	144.87	15802.67
27149542	615.25	140.81	19808.67

The Figure 6.3 and Table 6.3 show result on Water Quality dataset of the test perform on three algorithms for compression time.

This test shows that, the compression time taken increased in almost proportional when the size of the datasets increases. The LZRW algorithm proves to perform better on a scientific datasets, whereas the bzip algorithm shows the worst performance over a scientific datasets. This proves that the algorithm logic plays a very important part in the compression process. As we all know that compression is a process to reducing the redundant data from the dataset, therefore the time taken to compress a dataset must also include the time to read through the entire dataset to come out with the logic on which character or pattern is to be reduced. So the bzip algorithm in this case, proves that its logic is not suitable for the random characteristic data of the scientific dataset.

6.2 Size after compression vs. Size of datasets

These table and graph show the relationship betweens the Data Size after compression, in number of Byte, for each algorithm, against the Data Size of the input file, in number of Byte.

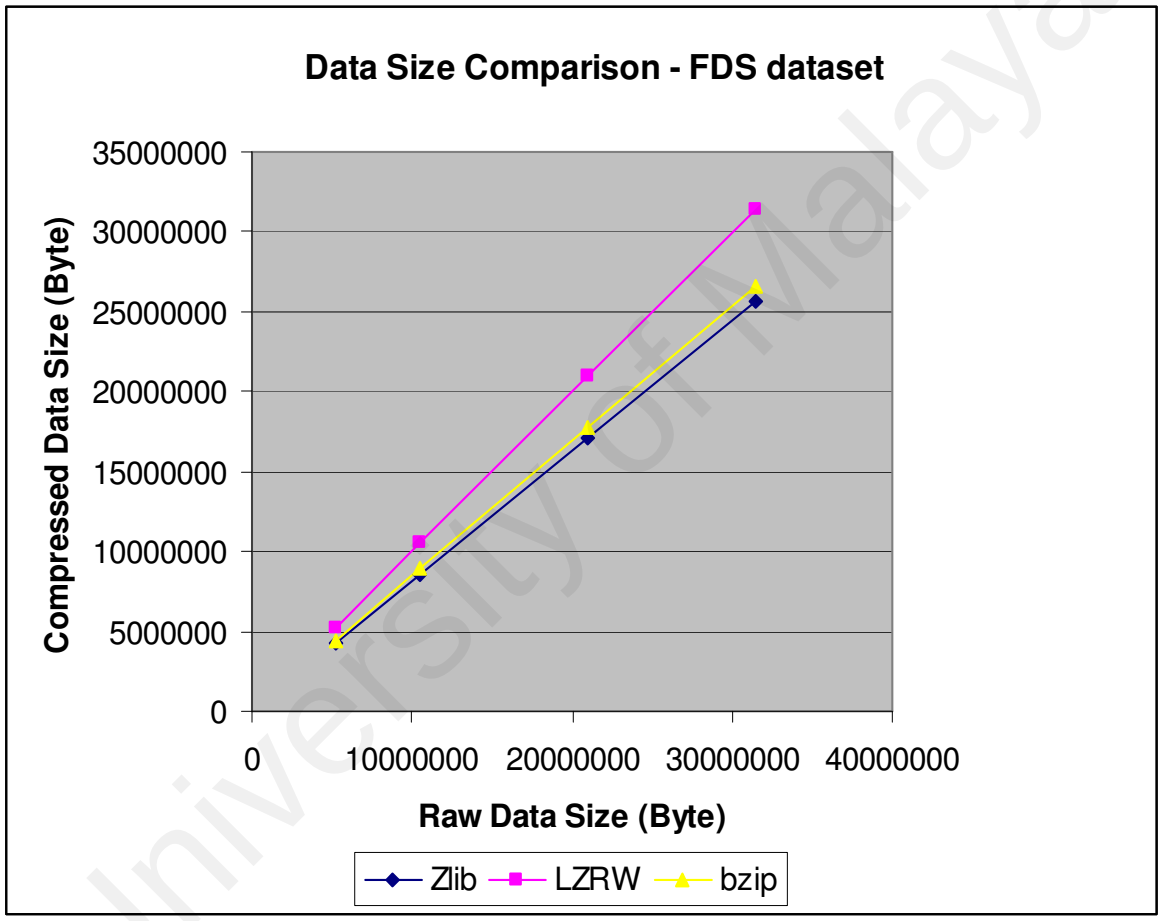


Figure 6.4 Compressed Data Size - FDS dataset

Table 6.4 Compressed Data Size - FDS dataset

Compression Size

Data Size	Zlib	LZRW	bzip
5275466	4318662.00	5267854.00	4473554.00
10546978	8612760.00	10530771.00	8922345.00
21023274	17133357.00	20986037.00	17737444.00
31499570	25634697.00	31435114.00	26533703.00

The Figure 6.4 and Table 6.4 show result on FDS dataset of the test perform on three algorithms for compressed data size.

Figure 6.5 Compressed Data Size - NCBI dataset

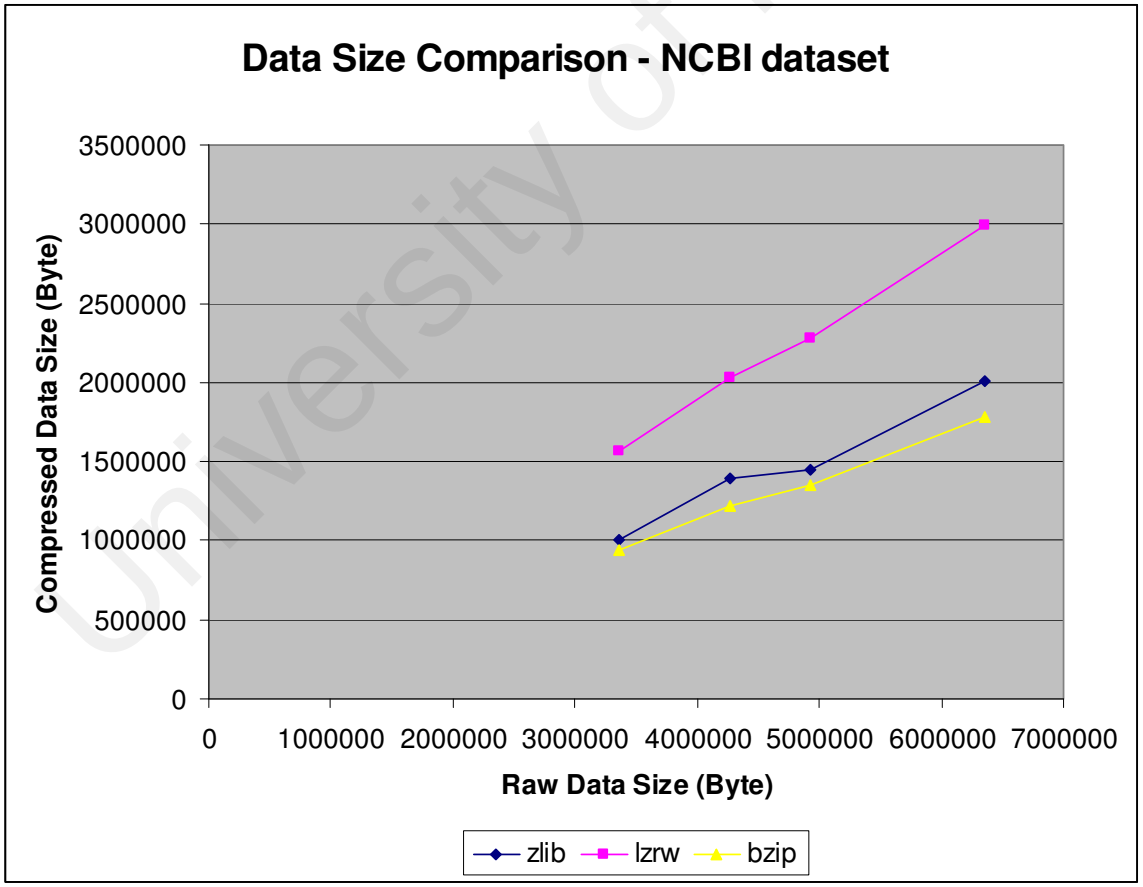


Table 6.5 Compressed Data Size - NCBI dataset

Compression Size			
Data Size	zlib	LZRW	bzip
3365257	999719	1569215	941548
4265064	1388489	2026173	1217808
4933488	1444812	2280488	1346137
6360548	2012552	2992561	1786697

The Figure 6.5 and Table 6.5 show result on NCBI dataset of the test perform on three algorithms for compressed data size.

Figure 6.6 Compressed Data Size – Water Quality dataset

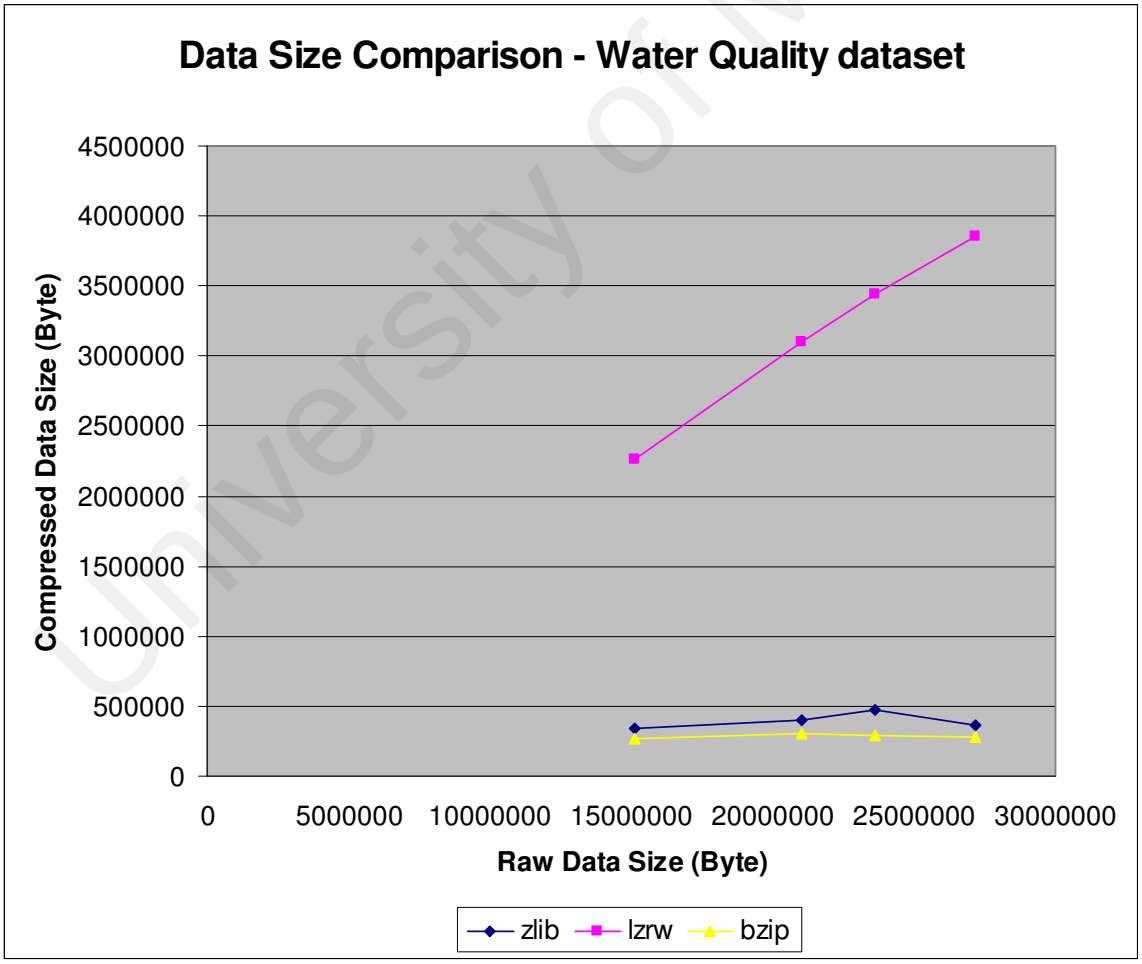


Table 6.6 Compressed Data Size – Water Quality dataset

Compression Size			
Data Size	zlib	LZRW	bzip
15148619	344381	2259686	265023
21004413	406111	3096002	305508
23609061	475239	3441771	287646
27149542	361140	3857183	274213

The Figure 6.6 and Table 6.6 show result on Water Quality dataset of the test perform on three algorithms for compressed data size.

This test shows the after compression size of the subjected datasets. Though the LZRW fair took the least time to compress a dataset, but the outcome from its compression shows that it is the least compressed. This result shows that, fast may not be a good thing in a compression process. The LZRW algorithm may be the fastest to compress the dataset, but it also proves that it is the least compressed as compare to the other algorithm. This is due to its algorithm logic that may have not able to reduce the redundancy from the dataset as much as the other two algorithms can do.

6.3 Delay time (against raw data transmission) vs. Size of datasets

These table and graph show the comparison betweens the reference time, which is the total time for transferring Raw data in different data size (measured in number of Byte), towards the total time to completely transfer the whole zipped data for each algorithm. The time measured for each algorithm includes the time to compress the raw data to generate the zipped data, and the time to transfer the zipped data to server side. This means, the time measurement is started when the first chunk of data is read from file, and ended when the last data is received.

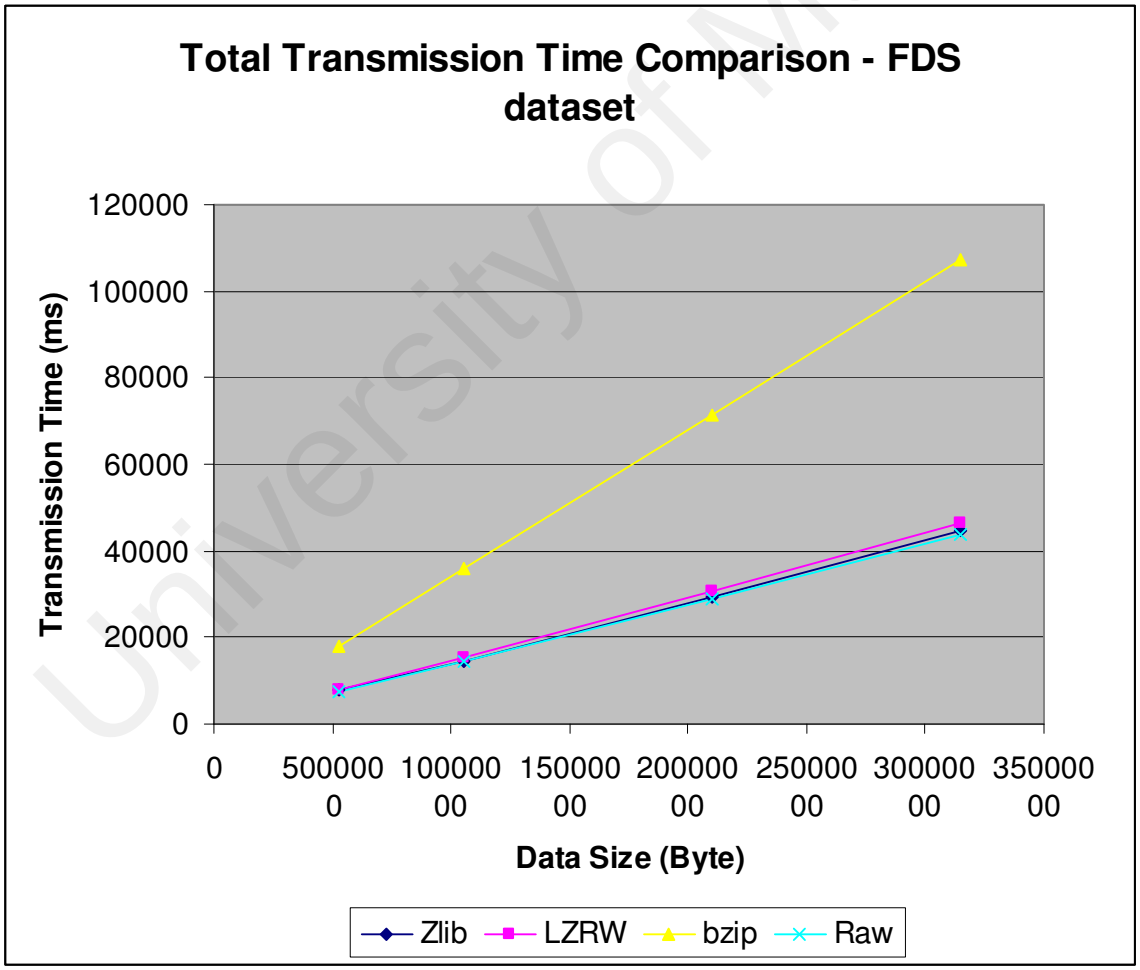


Figure 6.7 Total Transmission Time - FDS dataset

Table 6.7 Total Transmission Time - FDS dataset

Total Transmission Time

Data Size	Zlib	LZRW	bzip	Raw
5275466	7668.90	7687.90	18064.30	7230.30
10546978	14549.00	15250.90	36006.20	14448.70
21023274	29413.60	30552.90	71502.50	29118.10
31499570	44643.10	46342.60	107098.80	43582.10

The Figure 6.7 and Table 6.7 show result on FDS dataset of the test perform on three algorithms for total transmission time.

Figure 6.8 Total Transmission Time - NCBI dataset

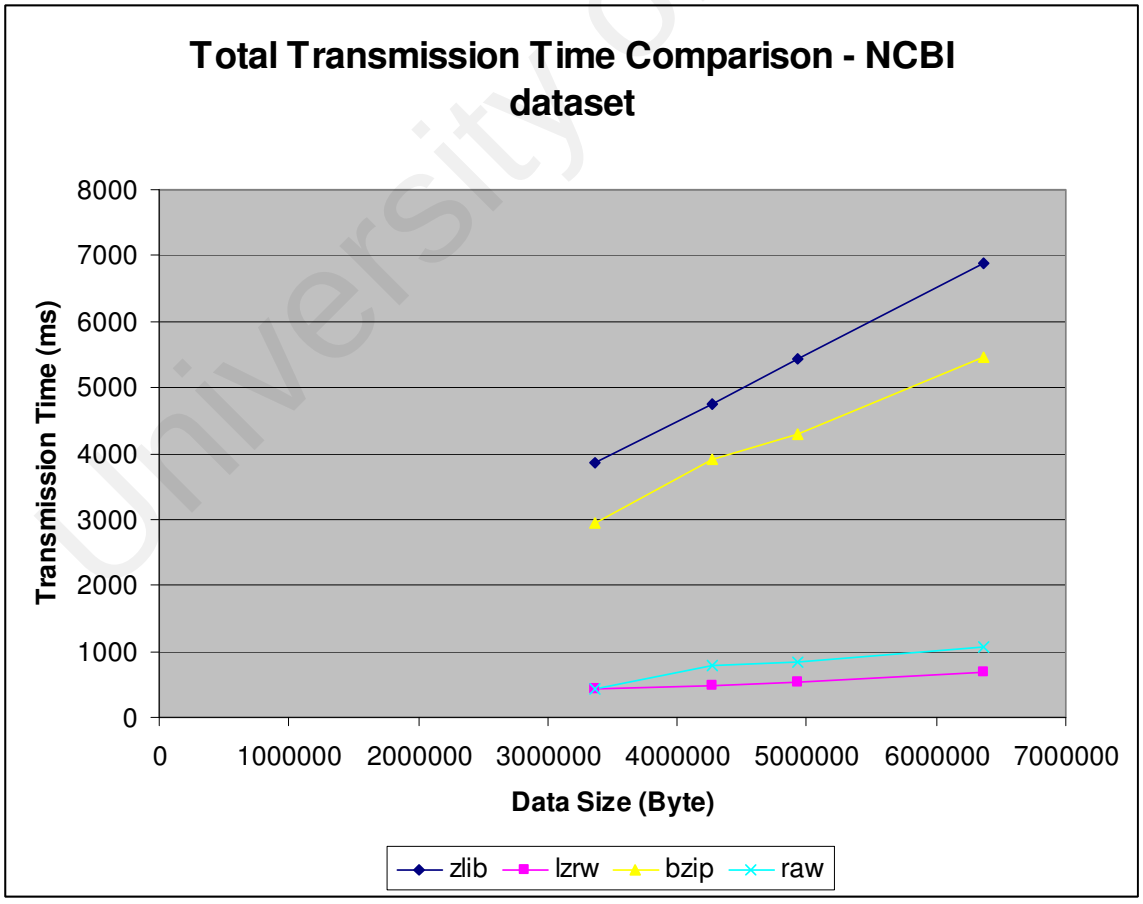


Table 6.8 Total Transmission Time - NCBI dataset

Total Transmission Time

Data Size	zlib	LZRW	bzip	raw
3365257	3866	421	2944	441
4265064	4746	471	3905	781
4933488	5437	541	4296	841
6360548	6890	691	5467	1062

The Figure 6.8 and Table 6.8 show result on NCBI dataset of the test perform on three algorithms for total transmission time.

Figure 6.9 Total Transmission Time – Water Quality dataset

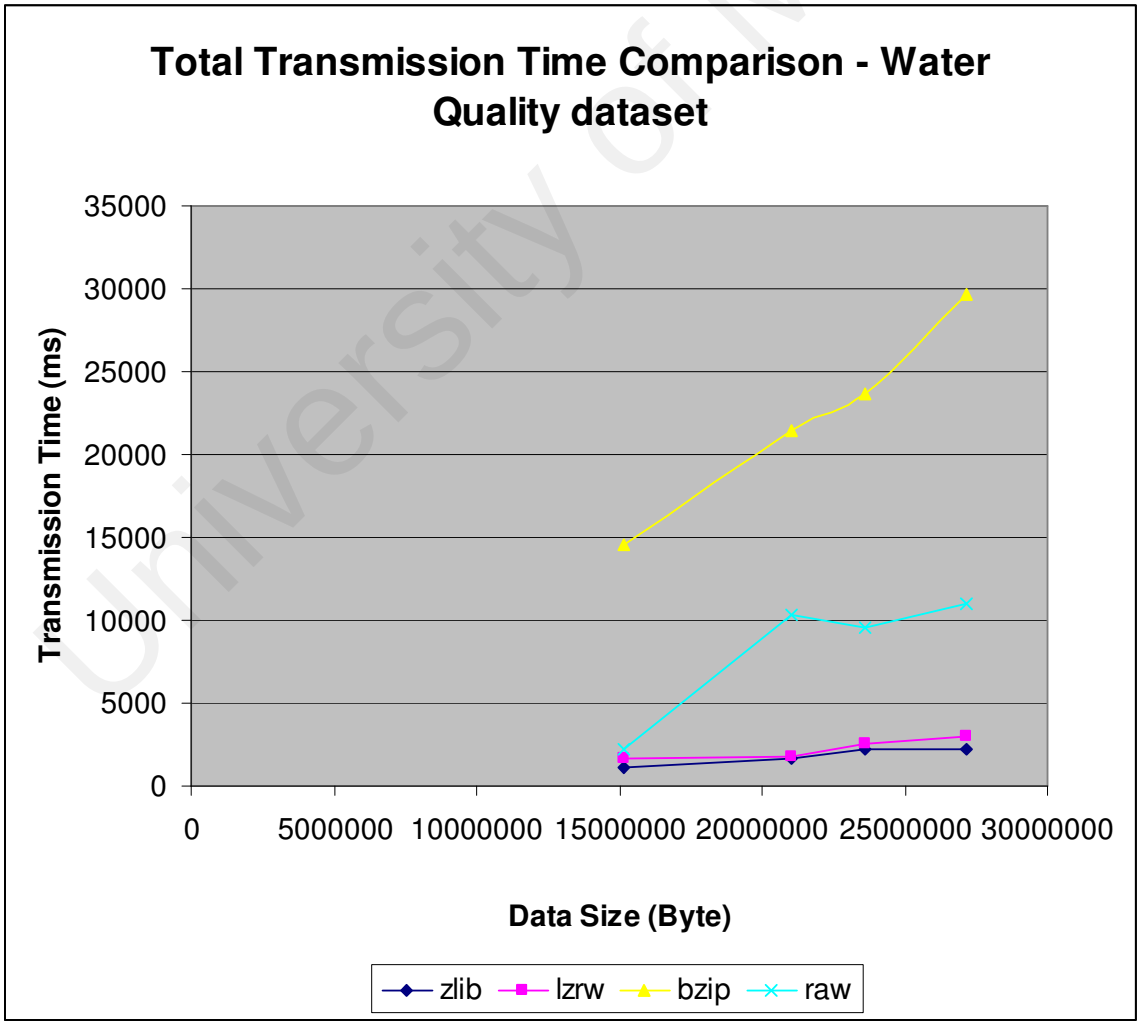


Table 6.9 Total Transmission Time – Water Quality dataset

Total Transmission Time

Data Size	Zlib	LZRW	bzip	raw
15148619	1101	1650	14581	2273
21004413	1662	1802	21431	10375
23609061	2173	2570	23704	9584
27149542	2253	2952	29713	10976

The Figure 6.9 and Table 6.9 show result on Water Quality dataset of the test perform on three algorithms for total transmission time.

This test relies very much on the first test; since it contributes most of the time spend in the entire process. With the most time taken on the first test, the bzip algorithm, again, proves to be the worst performed among the other compression algorithm. With test done on a simulated networked environment, without the delay and data lose, it is very obvious that with the least compressed dataset (the larges in size after compression) will take up the most time for the entire process. This test also shows that, it may not be always good to compress the dataset before transmitting. As we can see from the results, the total time taken to compress and transmit the data over the network are most of the times longer than transmitting the raw dataset.

6.4 Compression Ratio

These table and graph compare the Compression Ratio of each algorithm, in percentage, for each algorithm, against the Data Size of the input file, in number of Byte. The formula for compression Ratio is:

Compression Ratio = (1 – ([Compressed Data Size (Byte)] / [Raw Data Size (Byte)])) * 100 %

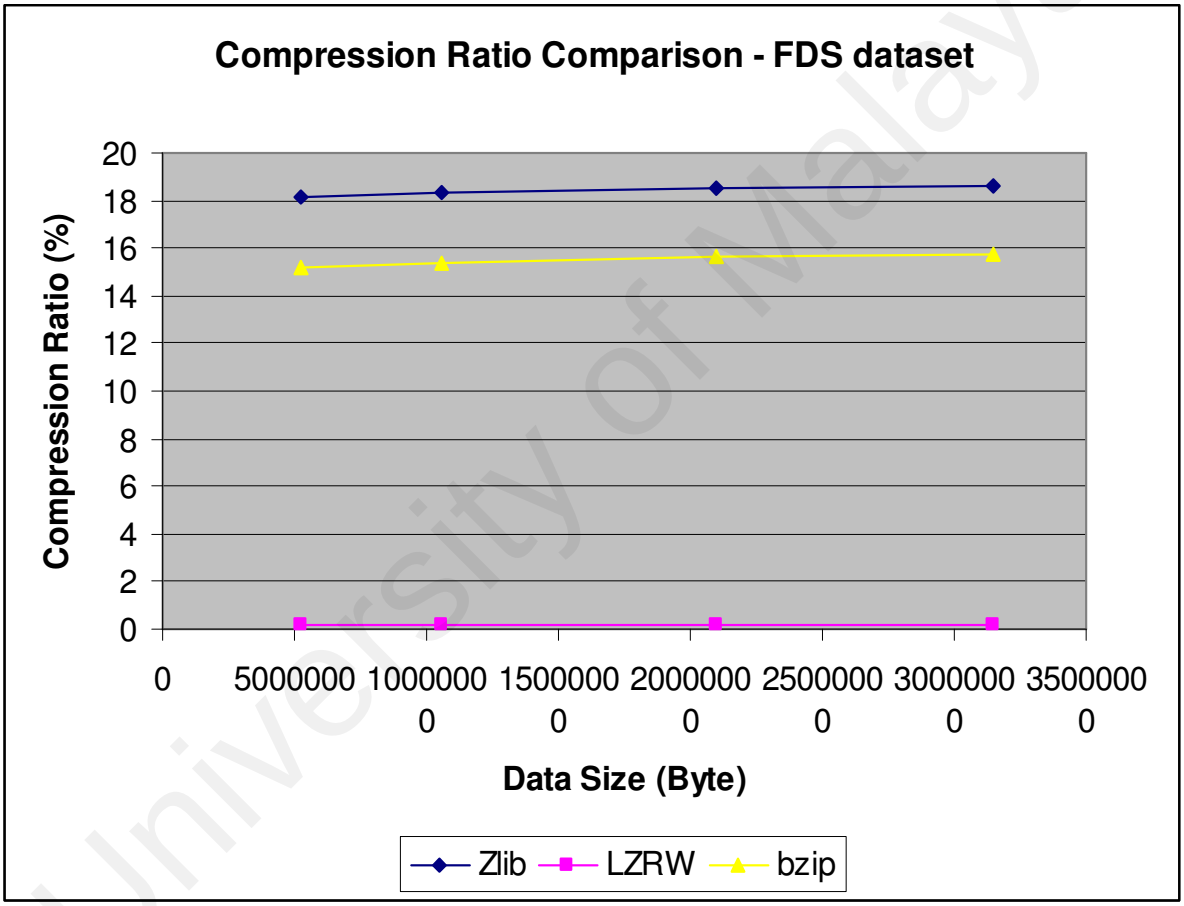


Figure 6.10 Compression Ratio - FDS dataset

Table 6.10 Compression Ratio - FDS dataset

Compression Ratio

Data Size	Zlib	LZRW	bzip
5275466	18.14	0.14	15.20
10546978	18.34	0.15	15.40
21023274	18.50	0.18	15.63
31499570	18.62	0.20	15.76

The Figure 6.10 and Table 6.10 show result on FDS dataset of the test perform on three algorithms for compression ratio. Based on FDS dataset, Zlib algorithm shows an impressive result average 18% compression ratio.

Figure 6.11 Compression Ratio - NCBI dataset

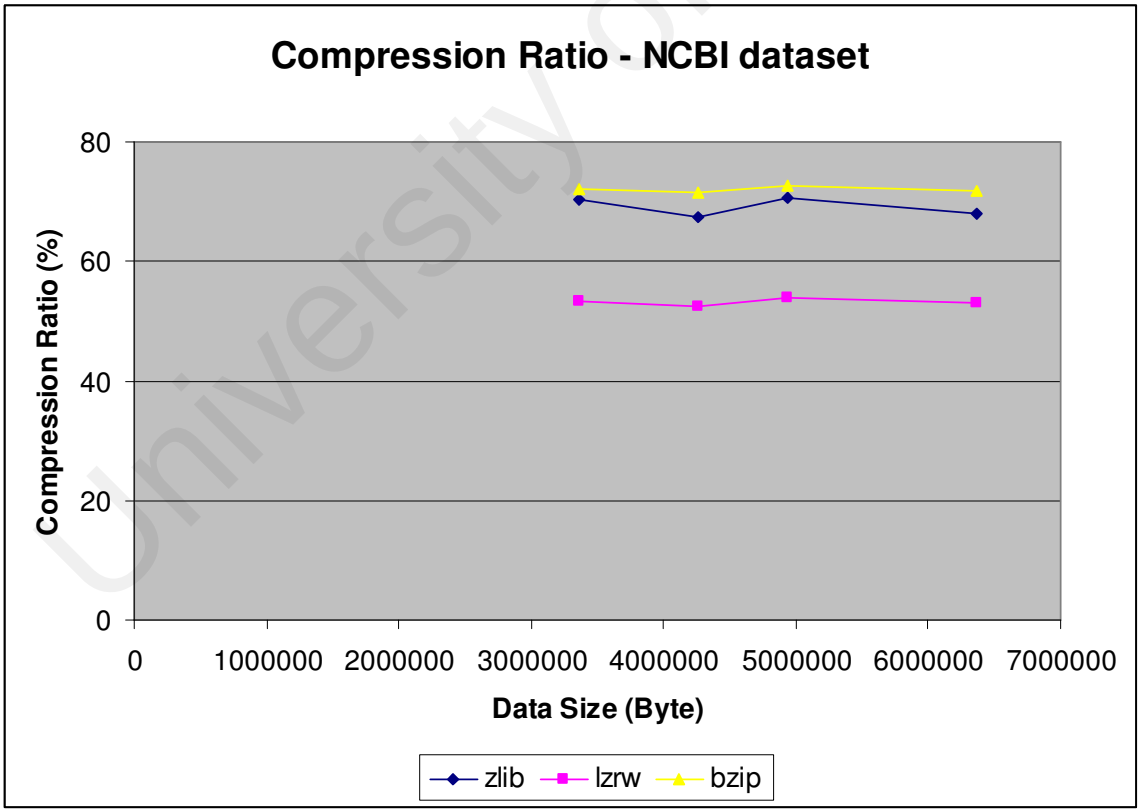


Table 6.11 Compression Ratio - NCBI dataset

Compression ratio

Data Size	zlib	LZRW	bzip
3365257	70.29	53.37	72.02
4265064	67.45	52.49	71.45
4933488	70.71	53.78	72.71
6360548	67.99	52.95	71.91

The Figure 6.11 and Table 6.11 show result on NCBI dataset of the test perform on three algorithms for compression ratio. Based on NCBI dataset, bzip algorithm shows an impressive result average 72% compression ratio.

Figure 6.12 Compression Ratio – Water Quality dataset

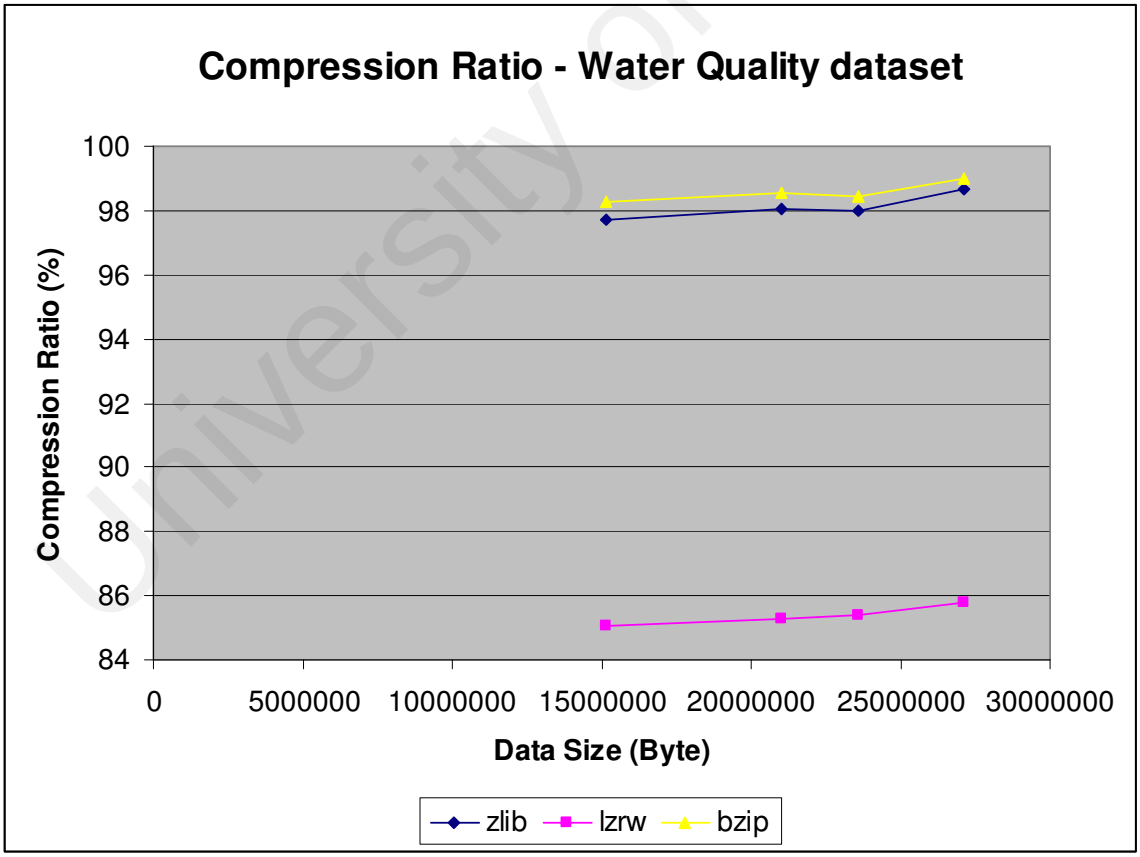


Table 6.12 Compression Ratio – Water Quality dataset

Compression ratio

Data Size	zlib	LZRW	bzip
15148619	97.73	85.08	98.25
21004413	98.07	85.26	98.55
23609061	97.99	85.42	98.46
27149542	98.67	85.79	98.99

The Figure 6.12 and Table 6.12 show result on Water Quality dataset of the test perform on three algorithms for compression ratio. Based on Water Quality dataset, bzip algorithm shows an impressive result average 98% compression ratio.

This test shows the compression ratio of each algorithm. This shows that even if the dataset size increases, the performance are not degraded, in fact all 3 algorithms shows an improvement of performance over the increment of dataset size. This is evidence in the logic of compression technology. Recall the idea of compression is based on reducing the redundancy data or pattern in a dataset, therefore with a larger size of dataset, statistically speaking; the reoccurrences of a data pattern will be increased. So with much more redundancy occurrences, the algorithm will be able to reduce even more data from the dataset.

6.5 Data Rate

These table and graph compare the Data Transmission Rate of each algorithm, in Byte per second, for each algorithm, against the Data Size of the input file, in number of Byte. The Raw Data transmission measurement is included as a reference. The Data Rate is calculated using the following formula:

Data Rate = [Total Transmitted Zipped Data (Byte)] / [Total Transmission Time (second)]

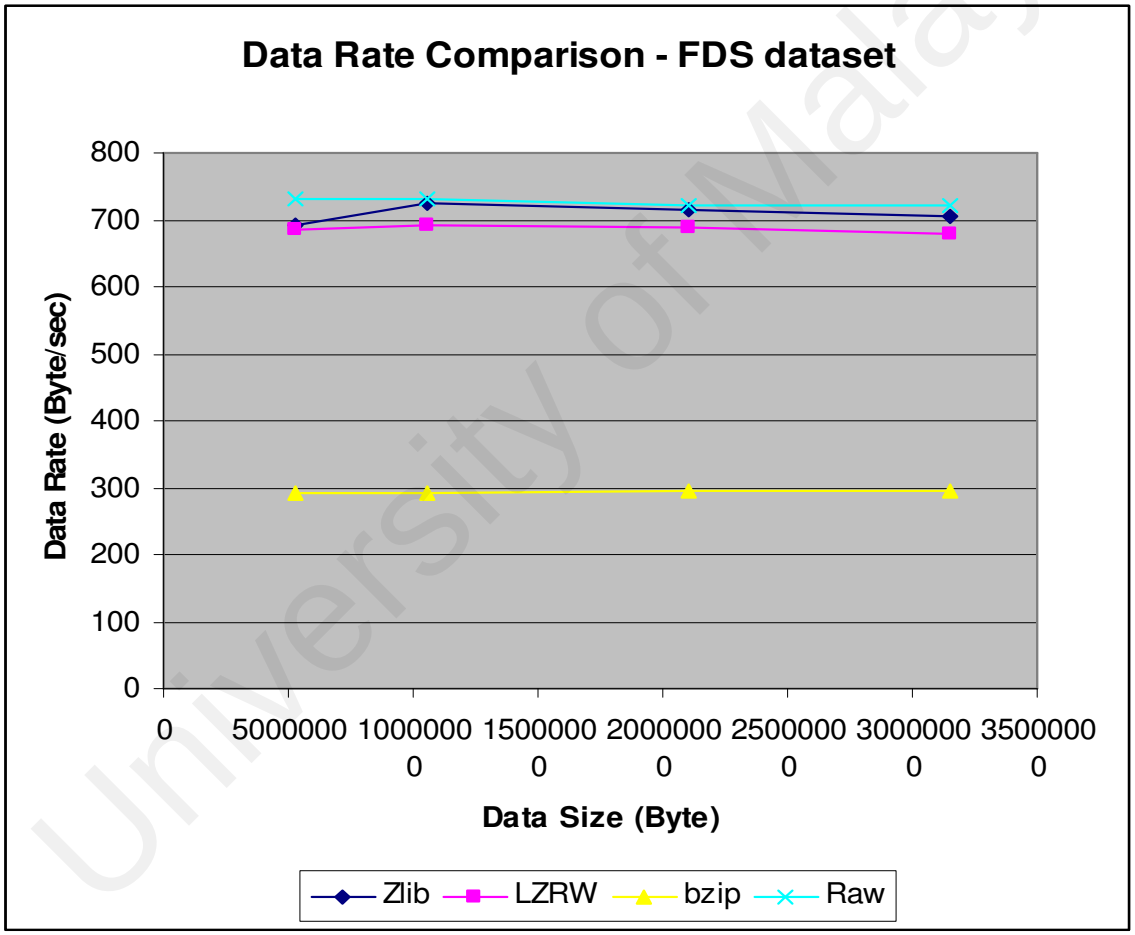


Figure 6.13 Data Rate - FDS dataset

Table 6.13 Data Rate - FDS dataset

Data Rate

Data Size	Zlib	LZRW	bzip	Raw
5275466	690.56	686.78	292.04	731.49
10546978	725.09	691.63	292.92	730.00
21023274	714.85	688.11	294.02	722.06
31499570	706.13	679.72	294.12	722.78

The Figure 6.13 and Table 6.13 show result on FDS dataset of the test perform on three algorithms for data rate.

This test shows that although with dataset compressed and transmitted, the performance is almost the same as transmitting a raw dataset without any compression. This is because the time taken to compress the dataset is bringing down the performance. This test also shows that for all 3 algorithms, they perform at optimum with the 10Mb sized datasets size.

Figure 6.14 Data Rate - NCBI dataset

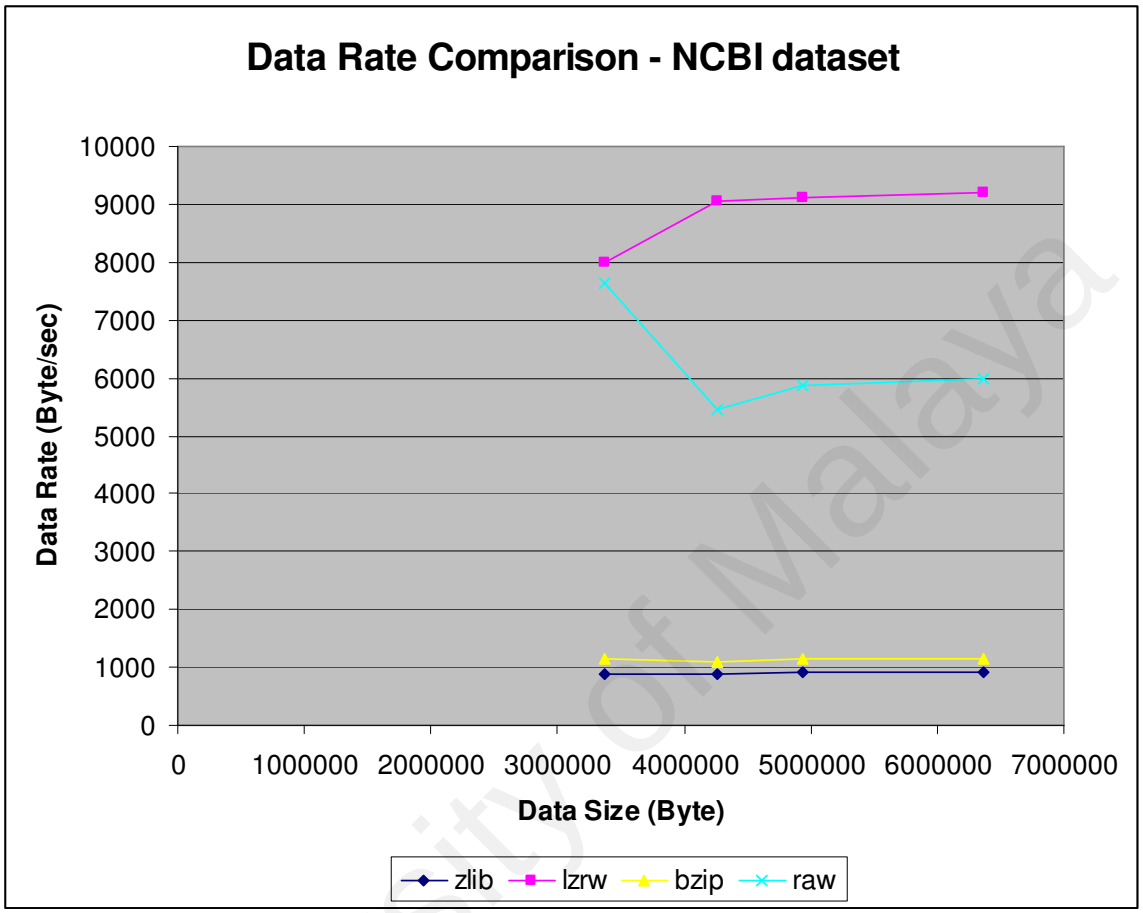


Table 6.14 Data Rate - NCBI dataset

Data Rate

Data Size	Zlib	LZRW	bzip	raw
3365257	870.48	7993.48	1143.09	7630.97
4265064	898.66	9055.34	1092.21	5461.03
4933488	907.39	9119.2	1148.39	5866.22
6360548	912.59	9204.85	1163.44	5989.22

The Figure 6.14 and Table 6.14 show result on NCBI dataset of the test perform on three algorithms for data rate.

Figure 6.15 Data Rate – Water Quality dataset

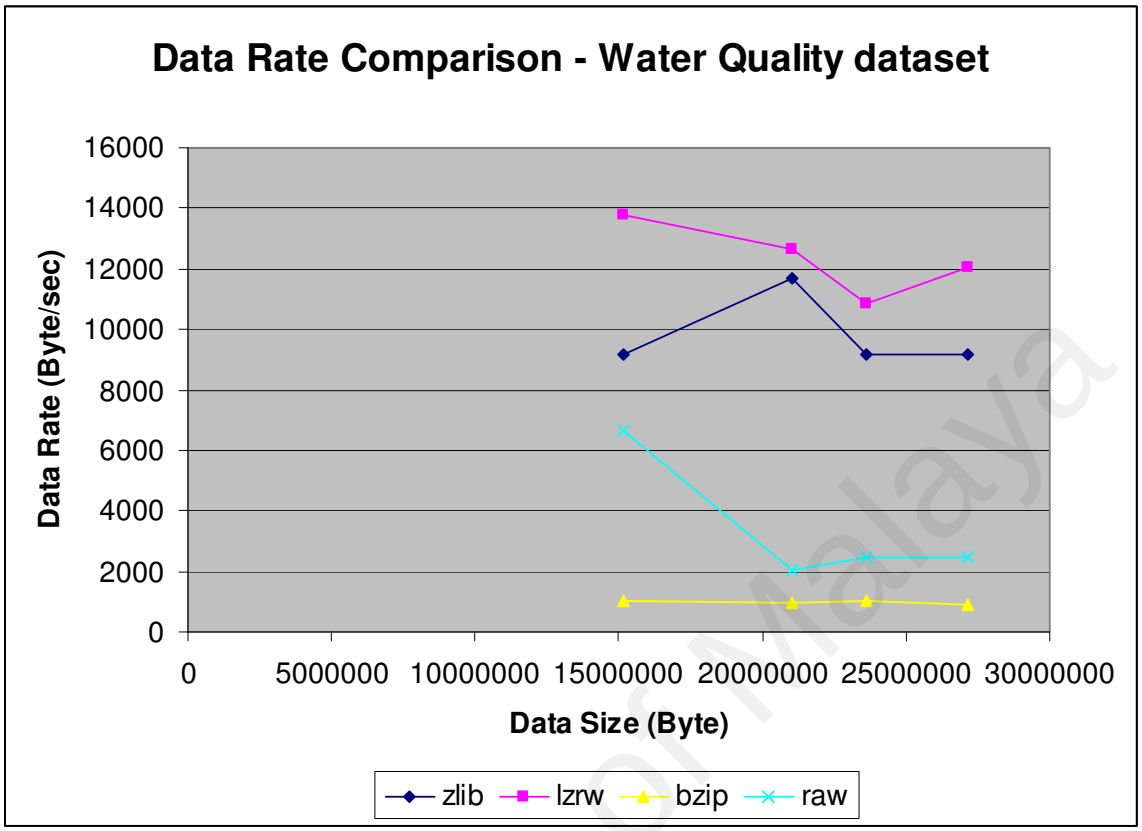


Table 6.15 Data Rate – Water Quality dataset

Data Rate

Data Size	zlib	LZRW	bzip	raw
15148619	9180.98	13758.96	1038.93	6664.59
21004413	11656.17	12638.03	980.09	2024.52
23609061	9186.41	10864.73	995.99	2463.38
27149542	9197	12050.4	913.73	2473.54

The Figure 6.15 and Table 6.15 show result on Water Quality dataset of the test perform on three algorithms for data rate.

This test shows the data rate over transmission. This test very much relies on the first test results. It measure the rate of data transmitted over a real time network environment.

6.6 Total Data Transmission Time

These table and graph compare the Total Data Transmission Time of each algorithm, in millisecond, for each algorithm, against the Zipped Data Size (the transferred data size), in number of Byte. The Raw Data transmission measurement is included as a reference. The Total Data Transmission Time is calculated using the following formula:

Total Data Transmission Time = ([Zipped Data Size (Byte)] / [Data Rate (Byte/second)]) * 1000 (ms)

The unit for Total Data Transmission Time is in millisecond to for ease of comparison with other measurement made, i.e., the Total Transmission Time, Compression Speed, and Delay Time.

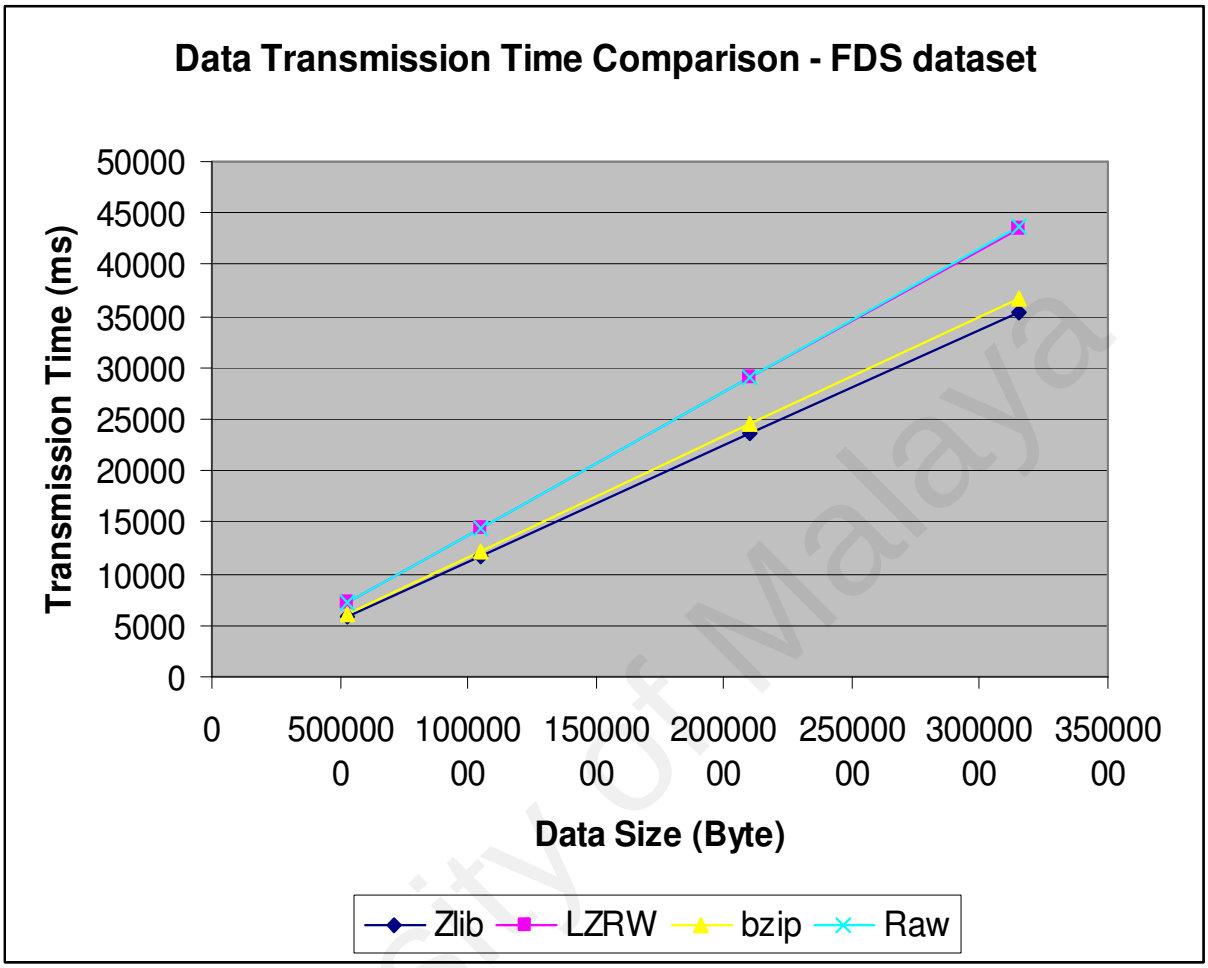


Figure 6.16 Data Transmission Time - FDS dataset

Table 6.16 Data Transmission Time - FDS dataset

Data Transmission Time

Data Size	Zlib	LZRW	bzip	Raw
5275466	5903.92	7201.53	6115.67	7230.30
10546978	11798.32	14425.73	12222.41	14448.70
21023274	23728.31	29063.96	24564.92	29118.10
31499570	35467.05	43492.25	36710.88	43582.10

The Figure 6.16 and Table 6.16 show result on FDS dataset of the test perform on three algorithms for data transmission time.

This test proves that with datasets compressed it shorten the time needed to transferred the data over the real time network environment. With a comparison of identical transmission using the raw data, the test shows an improvement of time taken of about 18%. The test in Figure 6.7, we see that the entire time taken to compress and transmitting are about the same by mere transmitting the raw dataset. In this test, now we can see that if by just comparing the transmission time of compressed dataset against the raw dataset, it proves that by compressing the dataset, it will take lest time in transmission. This is because with smaller dataset size, the throughput of the transmission will be increased.

Figure 6.17 Data Transmission Time - NCBI dataset

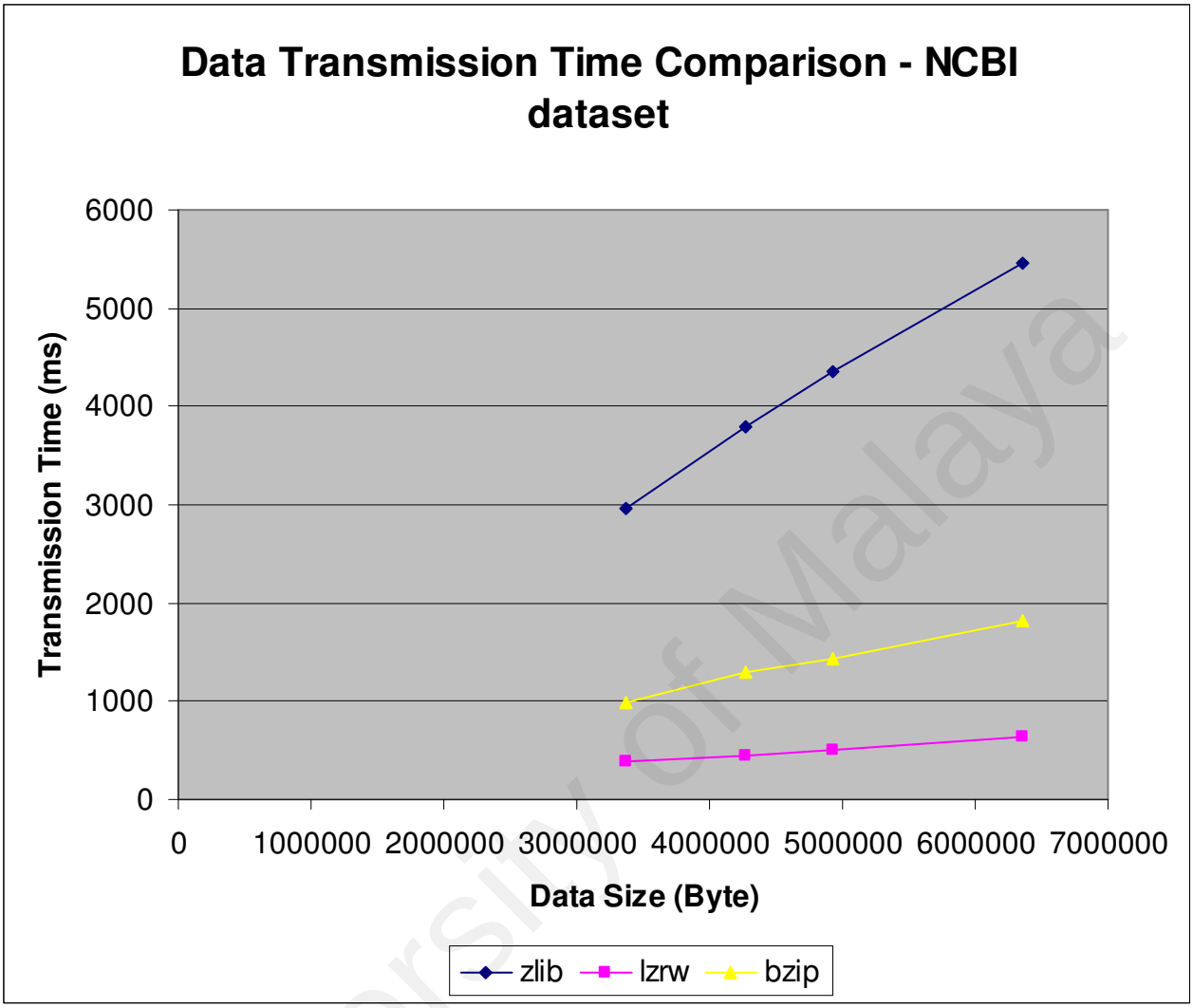


Table 6.17 Data Transmission Time - NCBI dataset

Data Transmission Time

Data Size	zlib	LZRW	bzip
3365257	2966.94	395.82	981.33
4265064	3796.8	443.95	1301.67
4933488	4349.6	511.22	1431.96
6360548	5454.59	647.81	1822.33

The Figure 6.17 and Table 6.17 show result on NCBI dataset of the test perform on three algorithms for data transmission time.

Figure 6.18 Data Transmission Time – Water Quality dataset

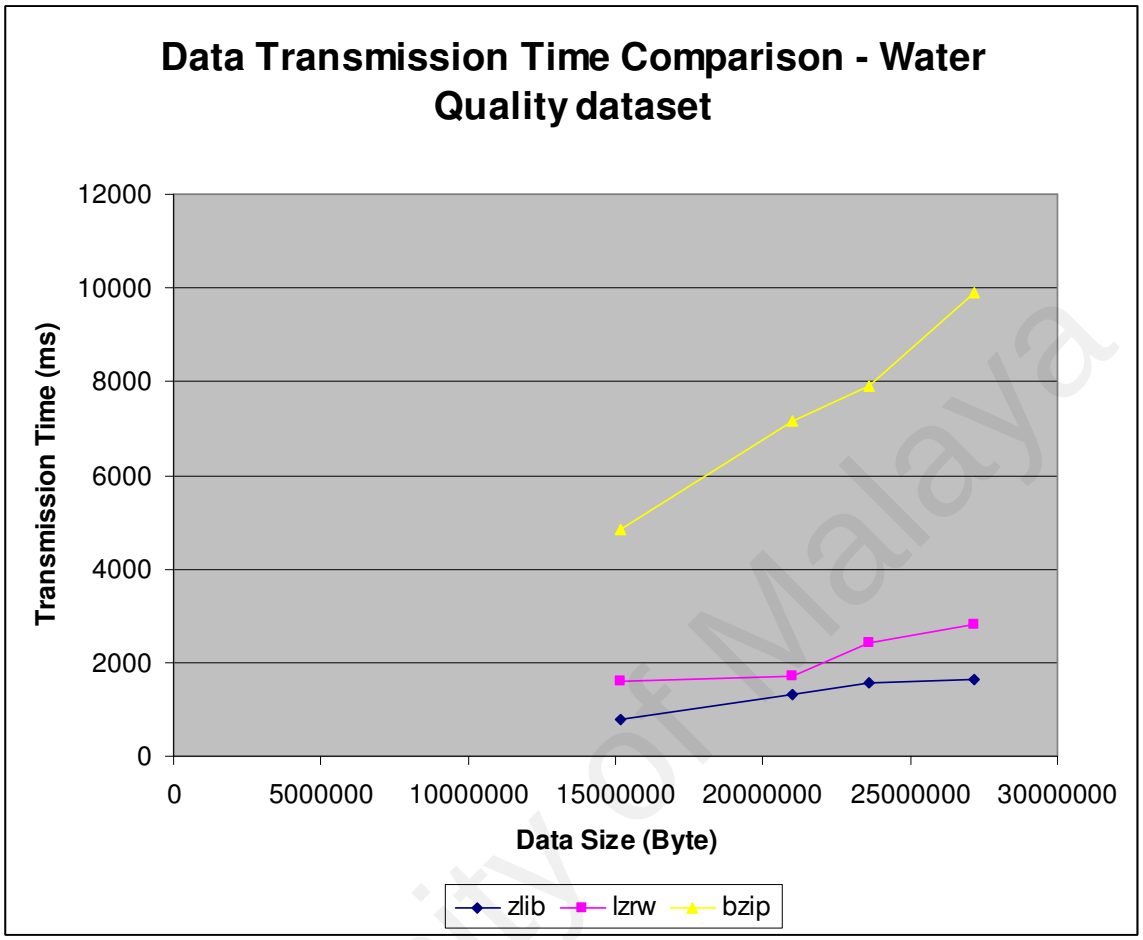


Table 6.18 Data Transmission Time – Water Quality dataset

Data Transmission Time

Data Size	zlib	LZRW	Bzip
15148619	770.49	1594.95	4860.33
21004413	1301.6	1709.67	7143.67
23609061	1575.33	2425.13	7901.33
27149542	1637.75	2811.19	9904.33

The Figure 6.18 and Table 6.18 show result on Water Quality dataset of the test perform on three algorithms for data transmission time.

6.7 Analysis

As shown in the Figure 6.16, the zlib algorithm for FDS dataset is not always the best among others. But it shows an optimal performance compare to other compression algorithms.

The LZRW algorithm on FDS dataset shows the best compression time on all data size. The compression time for LZRW algorithm is quite uniform and has little effect on the data size. The zlib algorithm shows slight increment in compression time in proportional to data size. The bzip algorithm shows dramatic increment in the compression time as the data size grows.

For zlib algorithm on FDS dataset, the lost of the time in performing compression is greatly recovered by its high compression ratio. Referring to Figure 6.10, the zlib algorithm achieves an 18 % compression ratio, and slight increment in the ratio as data size growth. This means it will need 82 % (and lesser) of the raw data transmission time. This makes the zlib algorithm able to transmit at the data rate closed to the data rate for raw data transmission, while saving the storage space for about 18%.

The LZRW algorithm on FDS dataset has little compression ratio on this data set. It only achieves less than 1 % of compression ratio, which means it is not suitable to work on this kind of data set.

The bzip algorithm on FDS dataset could achieve quite high compression ratio, which is about 15 %, and is increasing slightly as data size grows. However, due to the high compression time needed, it is not able to achieve a data rate that is close to the data rate for raw data transmission. It is only able to achieve about 40 % of the data rate for raw data transmission.

The large difference of the compression ratio between the zlib algorithm and the LZRW algorithm shows that, with the same basic of compression, it is able to achieve a higher compression ratio.

Dataset Category	Zlib	LZRW	Bzip
	Time (sec/Mb)	Time (sec/Mb)	Time (sec/Mb)
FDS 1	1.7758	1.4594	4.0380
FDS 2	1.6892	1.4482	4.0355
FDS 3	1.7167	1.4559	4.0312
FDS 4	1.7415	1.4742	4.0363
NCBI 1	3.8671	0.2683	3.1268
NCBI 2	3.4181	0.2325	3.2066
NCBI 3	3.7631	0.2372	3.1914
NCBI 4	3.4235	0.2309	3.0598
WQD 1	3.1970	0.7302	55.0179
WQD 2	4.0925	0.5820	70.1487
WQD 3	4.5724	0.7467	82.4068
WQD 4	6.2386	0.7653	108.3574

Table 6.19 Dataset category by Total Transmission Time/Compressed Size

Legend :
FDS = Fire Dynamic Simulator dataset
(binary format)
WQD = Water Quality Dataset (text format)
NCBI 1/3 = Fasta format
NCBI 2/4 = GenBank format

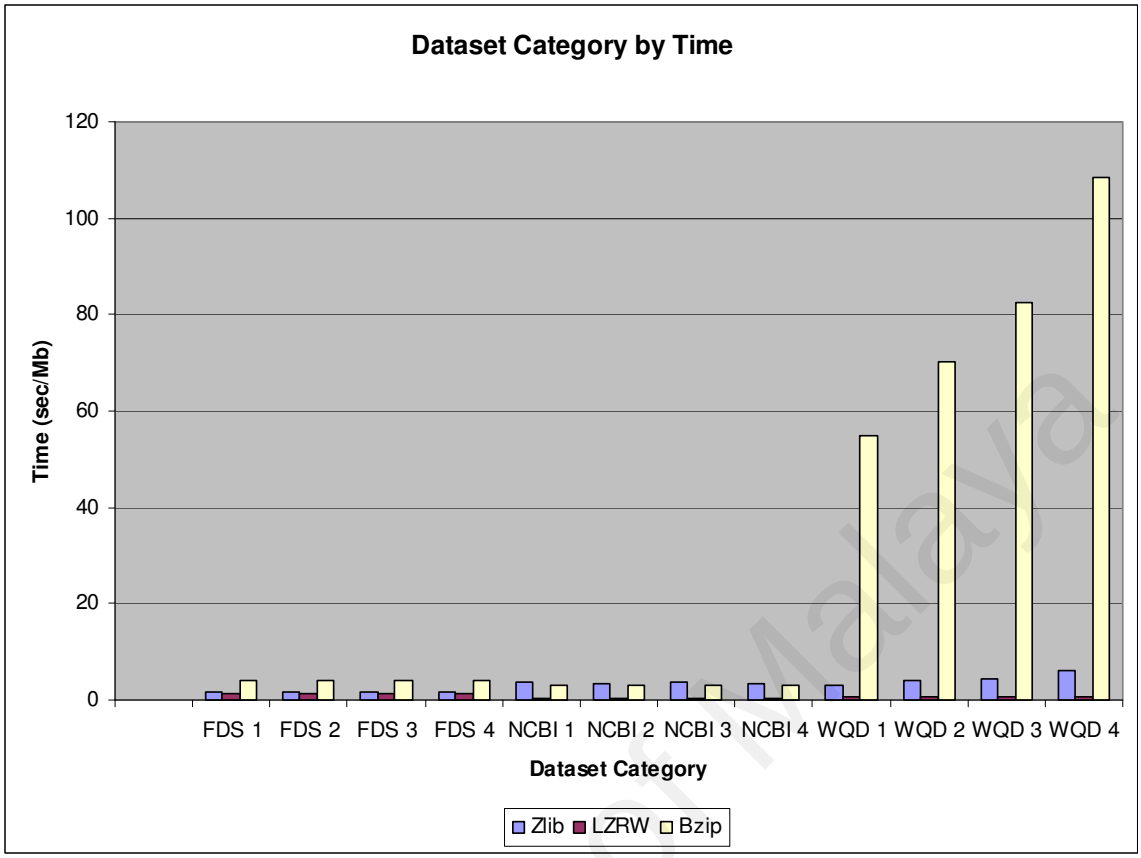


Figure 6.19 Dataset Category by Time

Legend :
FDS = Fire Dynamic Simulator dataset (binary format)
WQD = Water Quality Dataset (text format)
NCBI 1/3 = Fasta format
NCBI 2/4 = GenBank format

Based on above analysis on Figure 6.19, for text file format, Bzip compression is dependent on file size. The overall performance of Bzip compression drops tremendously with increasingly bigger file size. From the evaluation, Zlib and LZRW compression is not affected by file size or dataset type.

Dataset Category	Zlib	LZRW	Bzip
	Compression Ratio	Compression Ratio	Compression Ratio
FDS 1	18.14%	0.14%	15.20%
FDS 2	18.34%	0.15%	15.40%
FDS 3	18.50%	0.18%	15.63%
FDS 4	18.62%	0.20%	15.76%
NCBI 1	70.29%	53.37%	72.02%
NCBI 2	67.45%	52.49%	71.45%
NCBI 3	70.71%	53.78%	72.71%
NCBI 4	68.36%	52.95%	71.91%
WQD 1	97.73%	85.08%	98.25%
WQD 2	98.07%	85.26%	98.55%
WQD 3	97.99%	85.42%	98.46%
WQD 4	98.67%	85.79%	98.99%

Table 6.20 Dataset Category by Compression Ratio

Legend :
FDS = Fire Dynamic Simulator dataset (binary format)
WQD = Water Quality Dataset (text format)
NCBI 1/3 = Fasta format
NCBI 2/4 = GenBank format

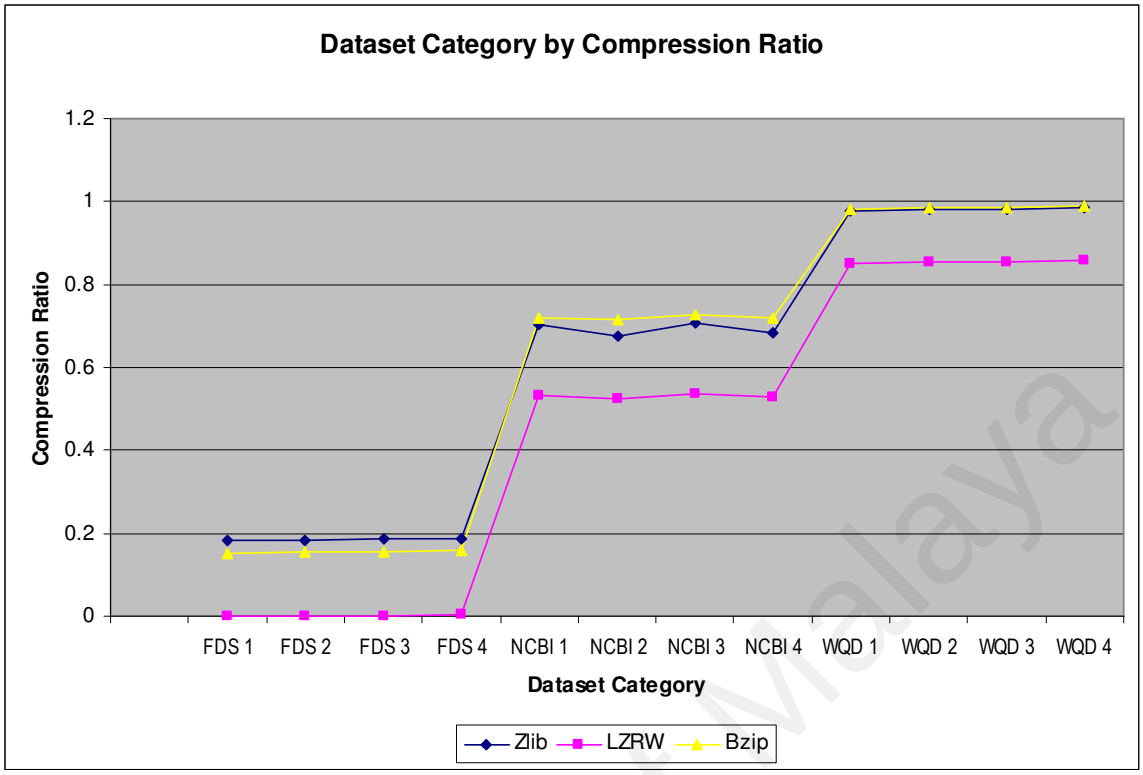


Figure 6.20 Dataset Category by Compression Ratio

Legend :
FDS = Fire Dynamic Simulator dataset
(binary format)
WQD = Water Quality Dataset (text format)
NCBI 1/3 = Fasta format
NCBI 2/4 = GenBank format

From the analysis of the Figure 6.20, different algorithms able to achieve different ratios for different data types or file format. Moreover, as for LZRW show as the lowest performing ratios given by all three dataset types.

6.8 Summary

This chapter covers the idea on how the implementation process on the compression evaluation simulator is carried out. This chapter also covers the compression testing and evaluation.

For FDS dataset, zlib algorithm works best compare to LZRW and bzip algorithm. Zlib algorithm achieve high compression ratio for FDS dataset with saving of storage space

For NCBI dataset, the file size was not affected by LZRW and zlib algorithms. Refering to Figure B.1, NCBI dataset works well on LZRW algorithm for best time performance in transmission time and compression time.

For Water Quality dataset, the three algorithms are able to achieve different compression ratio percentage. However, bzip algorithm performs best compression ratio performance compare to other algorithms but the drawback is its takes longer time for data transmission.

From the evaluation and the result obtained, the performance and behaviours of compression algorithms are based on time which include compression time, data transmission time and decompression time. The data transmission time is linear to the file size. The compression algorithms are dependent or independent of the file format or dataset types. With the effort of scientists and mathematicians, there is space for the growth of compression algorithm.

CHAPTER 7: Conclusion

The development of the compression evaluation benchmarking components has provided several valuable insights into the idea behind the compression techniques and behaviours that affects their performance, as well as a journey of research and development. This chapter will begin with the discussion of the objective and goals achieved during the process of completing this project.

The third section will discuss the evaluation outcome. This section will detail the knowledge gained from this compression evaluation benchmarking. It is then followed by a discussion on future enhancement for the simulator. This section will describe some of the new functionalities that can be implemented for future purposes.

7.1 Objectives and Goals Achieved

From this project we are now able to identify and evaluate the unique characteristic of a scientific dataset. A scientific dataset not just outstand the other dataset types in strength, but it is also provide the opportunity to vigorously test the compression algorithm.

Through the evaluation testing on the selected compression algorithm, we were able to identify and obtain the algorithm that best suit the unique characteristic of the scientific dataset. Also from the result analysis, we now understand how the size of datasets will affects the speeds, the ratio and performance as a whole on the compressed dataset.

By researching the current available compression techniques and the ongoing research on compression techniques, we were now able to identify the significance of compression performance and the various compression techniques behind.

7.2 Analysis Conclusion

The analysis results prove that, although the compression algorithm test subject derives from the same parent algorithm and theory, its performance varies when it comes to implement on different sized datasets. Some has a uniform performance through out the tests, while others either show superior performance or a performance dropped over large datasets.

From this analysis, we could deduce that, the technique of reducing the data redundancy in a datasets, play a very important role in its performance. How to reduce the redundancy depends greatly on the developer and the intended targeted datasets. Different developer may decide to work on the issue applying different method. Therefore it is clear and wise that specific dataset should targeted using different or specialised compression algorithm to maximize the ratios and performance.

7.3 Evaluation Outcome

According to the benchmarks conducted, we learned that, with the wide range of compression techniques and algorithms available, it is not easy to determine which one is more superior to the other. In fact, there is not one that is the most superior among the same category of algorithm and technique. A proper study of the compression algorithm specification should be done before selecting it as the subject compression algorithm or technique. And specific compression technique should be performed on a specific type of data. By doing this, the performance of the compression algorithm would be able to be maximized. In our case, the scientific dataset were used due to its random characteristic, which truly tested the compression algorithm, whereas if we were to choose a text datasets or numerical datasets, the result would be too good to simulate actual world scenario. These remind us of the background theory of the compression technology based on, “the information theory” [24], which postulate that a message contains redundancy.

As we can see, a compression algorithm takes advantages on the specific pattern in a datasets, which according to statistic laws [24], the algorithm would be able to take out redundant data and thus compressing the datasets. Therefore, we should be able to see in the very near future, that the new generation of compression technique and algorithm will be focusing even more in detail on a specific dataset, thus enhancing the compression performance to a greater height. And the more general type of compression algorithm and technique would be phase out.

7.4 Future Enhancement

Currently, this project only involves test on 3 different scientific datasets of variety file sizes and formats with 3 different types of algorithms. Therefore to obtain a wider coverage of results of different types, all the tests performed should be done in a lower specification test machine, to be able to truly evaluate the performance of a compression algorithm and diverse source of datasets from various applications.

Moving forward, the study should be able to test on streaming scientific datasets to allow remote monitoring of experiments on real time.

7.5 Summary

This chapter concludes this project as a whole, and shows that how the performance of compression algorithms can be evaluated and the importance of choosing suitable compression techniques against the intended datasets. It also shows how various properties of a datasets could affect the performance of the compression algorithms.

References

[1] Stephen Wolfram, 2002. *A New Kind of Science*, Wolfram Media, Place: Champaign, IL.

[2] Harringon, H. J., and James S. Harrington, 1996. *High Performance Benchmarking: 20 Steps To Success.* McGraw-Hill Publication.

[3] Camp, Robert C., 1989. *Benchmarking*. Milwaukee: Quality Press.

[4] Ziv, J., and Lempel, A., 1978. *Compression of Individual Sequences via Variable-Rate Coding*. IEEE Trans. Inform. Theory 24, 5 (Sept.), 530-536.

[5] Ziv, J., and Lempel, A., 1977. *A Universal Algorithm for Sequential Data Compression.* IEEE Trans. Inform. Theory 23, 3 (May), 337-343.

[6] Werner Bergmans, 2003. *The Compression Programs* [online]. Available from: http://www.maximumcompression.com/programs.php
[Accessed 4 November 2005].

[7] National Center for Biotechnology Information, 2005. *NCBI Map Viwer* [online]. Available from:
http://www.ncbi.nlm.nih.gov/mapview/
[Accessed 27 August 2005].

[8] National Institute of Standards and Technology, 2005. NIST Fire Dynamics Simulator (FDS) and Smokeview [online]. Available from:
http://fire.nist.gov/fds/
[Accessed 15 August 2005].

[9] U.S. Environmental Protection Agency, 2005. STORET Database Access [online]. Available from: http://www.epa.gov/storet/dbtop.html [Accessed 20 August 2005].

[10] G. Shavit, M. F. Ringenburg, J. West, R. E. Ladner, and E. A. Riskin, 2005. *Group testing for video compression*. In IEEE Data Compression Conference. Mar 2004.

[11] A.C. den Brinker and F. Riera-Palou., 2002. *Quantisation and interpolation of Laguerre prediction***.** Philips Research Eindhoven.

[12] David Salomon, 2000. *Data Compression: The Complete Reference. 2nd Edition.* Springer, Video compression: pp. 593-604, Mu-Law and A-Law Computing: pp. 644-649, H.261: pp. 627-630, MPEG: pp. 605-626.

[13] Stremler, F. G., 1990. *Introduction to Communication Systems, 3rd Ed*. Addison-Wesley Publishing Co., New York, pp.402-412, 541-547.

[14] Hambley, A.R., 1990. *An Introduction to Communication Systems*. Computer Science Press, New York, pp. 239-251.

[15] David J.C. MacKay, 2003. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, 2001. *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill. Section 16.3, pp.385–392, Section 30.2: The DFT and FFT, pp. 830-838.

[17] Paul E. Black, 2005. "Shannon-Fano coding", in *Dictionary of Algorithms and Data Structures* [online]. U.S. National Institute of Standards and Technology. Available from:
http://www.nist.gov/dads/HTML/shannonFano.html
[Accessed 15 July 2005]

[18] Mark Nelson, 1989. LZW Data Compression. *Dr. Dobb's Journal*.

[19] Terry Welch, 1984. *A Technique for High-Performance Data Compression*. IEEE Computer, 17(6):8-19.

[20] Mark Nelson, 1995. *The Data Compression Book*. 2nd ed., M&T Books.

[21] Khalid Sayood, 1996. *Introduction to Data Compression*. Morgan Kaufmann.

[22] Syed Ali Khayam, 2003. *The Discrete Cosine Transform (DCT): Theory and Application* [online]. Department of Elecrical & Computer Engineering, Michigan State University. Available from:
http://www.egr.msu.edu/waves/people/Ali_files/DCT_TR802.pdf
[Accessed 5 May 2005]

[23] Raymond W. Yeung., 2002. *A First Course in Information Theory*. Kluwer Academic/Plenum Publishers.

[24] Stanford Goldman, 2005. *Information Theory*. New York: Dover.

[25] Thomas M. Cover, Joy A. Thomas., 2006. *Elements of information theory*. 2nd Edition. New York: Wiley-Interscience.

[26] Arturo San Emeterio Campos, 1999. *LZ77 the basics of compression (2nd ed.)* [online]. Available from:
http://www.arturocampos.com/ac_lz77.html
[Accessed 26 July 2005]

[27] M. Purat, T. Liebchen, P. Noll., 1997. *Lossless Transform Coding of Audio Signals*. 102nd AES Convention, Munich.

[28] CDP Digital Audio Working Group, 2005. *Digital Audio Best Practices Version 2.0* [online]. Available from: http://www.cdpheritage.org/digital/audio/documents/CDPDABP_1-2.pdf [Accessed 4 December 2005]

[29] Miano, John., 1999. *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*. Boston: Addison-Wesley Professional.

[30] Ifeachor, Emmanuel C., and Jervis, Barrie W., 2002. *Digital Signal Processing: A Practical Approac.* Harlow, England: Pearson Education Limited.

[31] Jonathan (Y) Stein, 2000. *Digital Signal Processing, a Computer Science Perspective*. Wiley.

[32] Roger D. Smith, 1999.*Encyclopedia of Computer Science* [online]. Nature Publishing Group. Available from: http://www.modelbenders.com/encyclopedia/encyclopedia.html [Accessed 15 November 2005]

[33] P. Humphreys, 2004. *Extending Ourselves: Computational Science, Empiricism, and Scientific Method*. Oxford: Oxford University Press.

[34] *Common Steps in Benchmarking Models*. International Benchmarking Clearinghouse, Houston, TX., 1992.

[35] Jocelyn Dabeau, 2000. *An Inroduction to MP3* [online]. Available from: http://www.law.harvard.edu/faculty/tfisher/music/MP3.html [Accessed 20 Oct 2005]

[36] Xiph.org, 2000. *Vorbis audio compression* [online].Available from: http://www.xiph.org/vorbis/
{Accessed 20 Oct 2005]

[37] Matt Pharr and Greg Humphreys, 2004. *Physically Based Rendering: From Theory to Implementation* [online]. Morgan Kaufmann. Available from: http://graphics.stanford.edu/~mmp/chapters/pbrt_chapter7.pdf
[Accessed 20 Oct 2005]

[38] Hans Dieter Lüke, 1999. *The Origins of the Sampling Theorem*. IEEE Communications Magazine, pp.106–108, April 1999.

[39] Creative Labs, 2000. *History and Milestones* [online]. Available from: http://www.creative.com/corporate/about/
[Accessed 25 Oct 2005]

[40] M. H. Johnson and A. Alwan, 2002. *Speech Coding: Fundamentals and Applications*. Encyclopedia of Telecommunications, Wiley.

[41] M. R. Schroeder and B. S. Atal, 1985. *Code-excited linear prediction (CELP): high-quality speech at very low bit rates*. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 10, pp. 937-940.

[42] Sun Young Lee, Yong Ho Cho, Whoiyul Kim and Euee S. Jang, 2004. *Advances in Multimedia Information Processing* .PCM 2004, Volume 3333/2004 Selective Motion Estimation for Fast Video Encoding: pp. 630-638

[43] Murray, James D., and William van Ryper, 1996. *Encyclopedia of Graphics File Formats, Second Edition*. Sebastopol, Calif.: O'Reilly.

[44] Mertz, David, 2003. *Text Processing in Python*. Addison-Wesley Professional – PEARSON.

[45] Joint Video Team of ITU-T and ISO/IEC JTC 1, 2003 *Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC*), document JVT-G050r1, May 2003; technical corrigendum 1 documents JVT-K050r1 (non-integrated form) and JVT-K051r1 (integrated form), March 2004; and Fidelity Range Extensions documents JVT-L047 (non-integrated form) and JVT-L050 (integrated form), July 2004.