COMPARATIVE STUDY ON LOSSY AND LOSSLESS IMAGE COMPRESSION ALGORITHMS

by

Moumita Bose
Abhishek Mitra
Rick Majumder
Suman Das

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY
in the Department of Radio Physics and Electronics

CALCUTTA UNIVERSITY,
Kolkata

May,2016

COMPARATIVE STUDY ON LOSSY AND LOSSLESS IMAGE COMPRESSION ALGORITHMS

By

MOUMITA BOSE
UNIVERSITY REGISTRATION NO: 211-1121-1264-10
EXAM ROLL: NO 91/INT/131007

ABHISHEK MITRA
UNIVERSITY REGISTRATION NO: 023-1121-0505-10
EXAM ROLL NO: 91/INT/131013

RICK MAJUMDER
UNIVERSITY REGISTRATION NO: 211-1121-0907-09
EXAM ROLL NO:91/INT/131016

SUMAN DAS
UNIVERSITY REGISTRATION NO: 0007660
EXAM ROLL NO: 91/INT/131014

Under the supervision of

Dr. Partha P. Goswami

Assosiate Professor

Institute of Radio Physics and Electronics

University of Calcutta

Department of Radiophysics & Electronics,

University College of Technology, University of Calcutta

92,Acharaya Prafulla Chandra Road

Kolkata 700 009

Institute of Radio Physics and Electronics

University of Calcutta

Tel:+91-33-2350-9115/9116/9413

Fax: +91-33-2351-5828

SISIR MITRA BHAVAN

92,ACHARAYA PRAFULLA CHANDRA BHAVAN

KOLKATA -700 009

CERTIFICATE FROM SUPERVISOR

This is to certify that dissertation entitled "COMPARATIVE STUDY ON LOSSY AND LOSSLESS IMAGE COMPRESSION ALGORITHMS" has been carried out by Moumita Bose, Abhishek Mitra, Rick Majumder, Suman Das under my guidance and supervision and be accepted for Project work for final year in Bachelor of Technology in Information Technology.

Dr. Partha P Goswami

Supervisor, (Dept. of RPE)

University of Calcutta

Kolkata- 700009

Institute of Radio Physics and Electronics

University of Calcutta

Tel:+91-33-2350-9115/9116/9413

Fax: +91-33-2351-5828

SISIR MITRA BHAVAN

92,ACHARAYA PRAFULLA CHANDRA ROAD

KOLKATA -700 009

CERTIFICATE

This is to certify that Moumita Bose (Roll No – 91/INT/131007), Abhishek Mitra (Roll No – 91/INT/131013), Rick Majumder (Roll No – 91/INT/131016), Suman Das (Roll No – 91/INT/131014), Institute of Radiophysics and Electronics, University College of Science and Technology, Calcutta University have completed to satisfaction a project on "COMPARATIVE STUDY ON LOSSY AND LOSSLESS IMAGE COMPRESSION ALGORITHMS" during the academic year 2015-2016.

The dissertation has been prepared on the same to fulfill the requirements for the Project work for Final year in 'Bachelor of Technology' in Radiophysics and Electronics from Calcutta University, Kolkata.

Dr. Partha P Goswami
Supervisor, (Dept. of RPE)
University of Calcutta,
Kolkata

Dr. Nikhil Rajan Das
Head of the Department (Dept. of RPE),
University of Calcutta,Kolkata – 700 009

Acknowledgements

It is our privilege to express our sincerest regards to our project mentor, **Dr Prof. Partha P. Goswami** of ***Department of Radio Physics and Electronics, Calcutta University, Kolkata***, for his valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of our project work.

We deeply express our sincere thanks to our Head of Department **Dr Prof. Nikhil Ranjan Das** of ***Department of Radio Physics and Electronics, Calcutta University, Kolkata*** for encouraging and allowing us to present the project work on this topic "***A Comparative Study of Lossy and Lossless Image Compression Algorithms*** "at our department premises.

We wish to express our profound gratitude to **Mr. Ravindra Babu Ravula**, ***Ex Senior Software Engineer, CISCO System India Pvt. Ltd, Alumnus of IISc, Bangalore***, for constantly motivating us to work harder and without his help we have stood no chance at all what we have made on this Project Work.

We take this opportunity to thank all our Professors who have directly or indirectly helped our project work. We pay our respects and love to our parents and all other family members for their love and encouragement throughout our career.

Last but not least, my sincere thanks to all our teammates who have patiently extended all sorts of help for accomplishing this undertaking.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

BMP	Bit Map Picture
BWT	Burrows Wheeler Transform
GIF	Graphics Interchange Format
JPEG	Joint Photographic Experts Group
LZW	Lempel Ziv Wlech
MSE	Mean Square Error
PCX	Picture Exchange
PSNR	Peak-Signal to noise Ratio
BR	Bit-rate
RLD	Run Length Decoder
RLE	Run Length Encoding
TIFF	Tagged Image File Format

CHAPTER 1

1.1 Introduction

Image compression is technique through which we can compress the original image with lesser memory size, without degrading the quality of the image to an unacceptable level. Standing on these 21st Century, the development and demand of multimedia product grows rapidly which is contributing to ineffective bandwidth utilization and storage of memory devices. For this reason the art of data compression becomes more appreciably.

1.2 Problem Definition

Our objective of this dissertation is to study the different types of Image Compression algorithms and do the comparisons between them. There are generally two types of image compression algorithms. Image Compression may be LOSSY or LOSSLESS. In a Lossless image compression algorithm the original image to be perfectly reconstructed from the compressed file, while on the other hand lossy image compression algorithm uses partial data discarding to represent the content upto an acceptable level.

The amount of data compression possible using lossy compression is often much higher than through lossless techniques.

In our project we have chosen three different types of Lossless image compression algorithms as follows RLE (Run Length Encoding), LZW (Lemple Ziv Welch), BWT (Burrows Wheeler transform) and one Lossy image compression technique BTC (Block Truncation Coding).

All Lossless Image Compression, algorithm will be compared with each other in terms of their compression ratio and the Lossy Image Compression BTC (Block Truncation Coding) will be compared with the original image using Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR), MSE and PSNR are the two error metrics used to compare image compression quality.

Compression ratio with respect to Original image size-

The ratio is measured by uncompressed image size with respect to the compressed image size.

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

Percentage of Compression [1]-

To calculate it first percentage of the compressed image size is computed with

respect to the uncompressed image size:

$$\text{compressed file size}(\%) = \frac{100 \times \textit{compressed filesize}}{\textit{uncompressed filesize}}$$

1.3 Input File Format[2]

The format definition is as follows. You can use the libnetpbm C subroutine library to conveniently and accurately read and interpret the format.

A PGM file consists of a sequence of one or more PGM images. There are no data, delimiters, or padding before, after, or between images.

Each PGM image consists of the following:

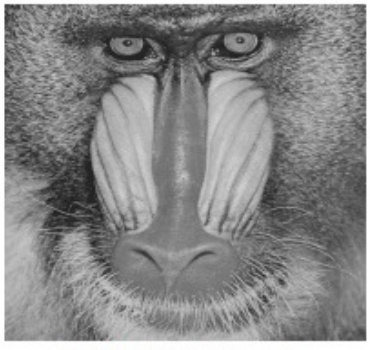
1. A "magic number" for identifying the file type. A pgm image's magic number is the two characters "P5".
2. Whitespace (blanks, TABs, CRs, LFs).
3. A width, formatted as ASCII characters in decimal.
   Whitespace.
4. A height, again in ASCII decimal.
5. Whitespace.
6. The maximum gray value (Maxval), again in ASCII decimal. Must be less than 65536 , and more than zero.
7. A single whitespace character (usually a newline).
8. A raster of Height rows, in order from top to bottom. Each row consists of Width gray values, in order from left to right. Each gray value is a number from 0 through Maxval, with 0 being black and Maxval being white. Each gray value is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte.
9. A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.

Here is an example of a small image in the plain PGM format.

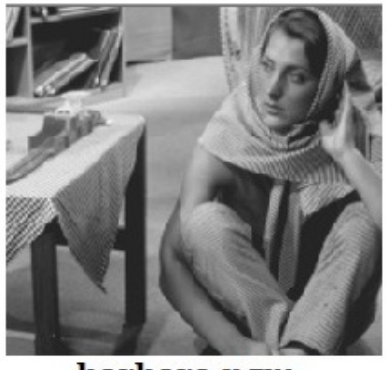
```
P2
# comment line
24 7
255
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  3
3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0  0  3  0
0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0  0  3  3  3  0
0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0  0  3  0  0  0  0
0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0  0  3  0  0  0  0  0  7
7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

1.3.1 [3]

BENCHMARKS IMAGES

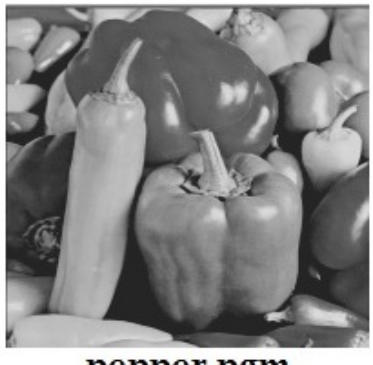


baboon.pgm

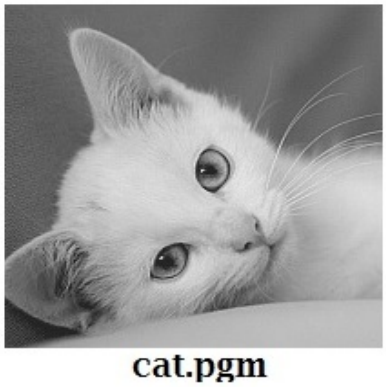


barbara.pgm

leena.pgm



pepper.pgm

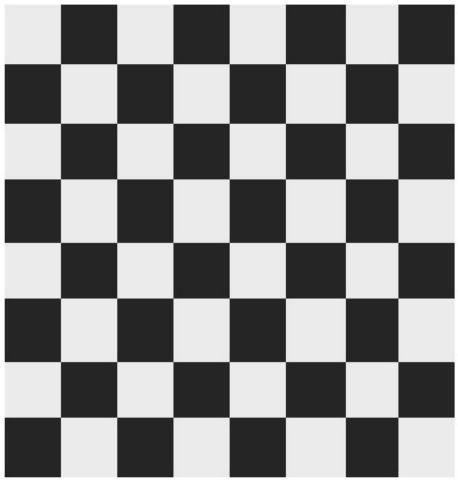


cat.pgm

flower.pgm

1.3.2

SOLID-COLOR IMAGES/COMPUTER GENERATED IMAGES



chess board.pgm

sky.pgm

color bar.pgm

google.pgm

CHAPTER 2

Run Length Encoding

2.1 Introduction Of Run Length Encoding

Run Length Encoding is a simple and popular data compression algorithm is frequently applied to image (or pixel in a scan line). Run Length Encoding (RLE) algorithm performs a lossless compression of input data based on sequence of identical consecutive data elements are stored as a single data value and count, rather than as the original run.

If we apply RLE algorithm on short length sequence of data then total size of data bits for short sequence data may increase. This is most useful on data that contains many such runs. In worst case for random noise encoding is heavier than original noise.

It is simple form of statistical encoding. Here we substitute a frequently repeating pattern with a code. The code is shorter than pattern gives us compression.

Main Approach of RLE is:

- Count the occurrence of tokens.
- Sort in descending order.
- Assign some symbols to highest count.

The flowchart of the Run Length Encoding algorithm is depicted in fig-2.1

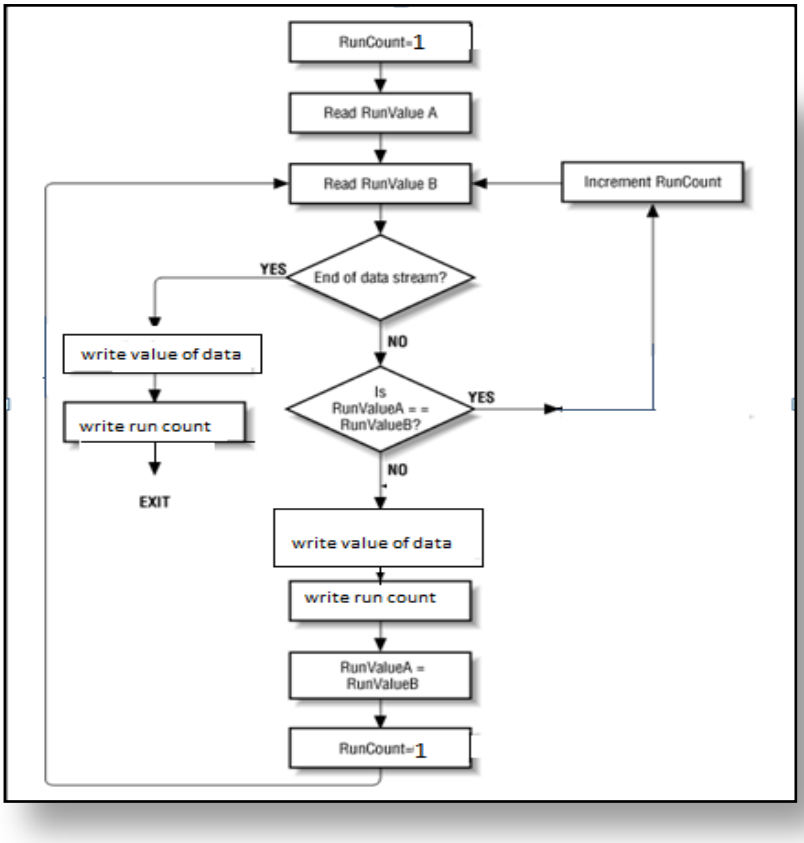


Fig- 2.1

2.1.1 Process steps for generating Run Length Encoding:

1. Start with long sequence of string and assign run count value with 1.
2. Read first bit of the string.
3. Read second bit of the string.
4. Check if the string end or not if yes then go to step 5 else go to step 6
5. Write value of first bit and write count value of same bit then exit.
6. Check 1st bit value equal to 2nd bit value of the string if yes increment same bit run count value by 1 and go to step 3. Else write the 1st bit value and its count value.
7. Assign the value of 2nd bit into 1st bit.
8. Assign run count value with 1 and go to step 3.

In order to clarify this algorithm, we give an example. We take a long sequence, i.e.

A	A	B	B	B	B	B	B	B	B	C	C	C	C	A	A	A	A	B

Total Bytes= 19 Bytes

After compression,

A	2	B	8	C	4	A	4	B	1

Total Bytes=10 bytes

Compression rate= (19-10)/19*100=47%

2.1.2 Process steps for generating decompression of Run Length Encoding:

1. Start on the first element of the data input.

2. Read the data and store it in a register A and initialize C with '1'.

3. Print the data which is in register A.

4. Take the second data from input and store it on register B.

5. If C=B then go to step 2 else print A and increment C and repeat.

In order to clarify this algorithm, we give an example. We take encoded sequence, i.e

A	2	B	8	C	4	A	4	B	1

In decompression , it will first read A symbol and initialize counter c with 1 and also store A into register A. After that it will print A. Take the next data i.e 2 which is nothing but the run length count of symbol A and put it register B.In this case, content of register B and content register c is not same so it will print the symbol A and counter c will be incremented. Now the content of register c and content of register B is same then it will again read next symbol B and store it into register A as well as assign counter c with 1.This process will continue until we will reach to end of the string.

After decompression it will print,

A	A	B	B	B	B	B	B	B	B	C	C	C	C	A	A	A	A	B

Coming to the time complexity for RLE it can be clearly stated that we need to traverse the horizontal pixels formation and required to count the occurrence of each pixel compare to the preceding pixels which requires O(n) time and there can be at most 'c_n' (c is constant) no. of individual horizontal pixels formation which will required

O(n)	** assuming that the resolution of image is(c_1 X c_2)n,c_1 andc_2 are the constants

2.1.3 APPLICATIONS

It is a small compression component used in JPEG* compression. It is used mostly for TIFF*, BMP* and PCX* files[4].

* *Word expansions have been mentioned in **Abbreviation** section*

2.1.4 RLE Compression Comparisons

RLE COMPRESSION COMPARISON ON BENCHMARK

IMAGE NAME	RESOLUTION	UNCOMPRESSED SIZE	COMPRESSED SIZE	% of COMPRESSION
baboon.pgm	200 x 200	151KB	218KB	-44.37
barbara.pgm	200 x 200	145KB	195KB	-38.48
leena.pgm	200 x 200	139KB	190KB	-36.69
peppers.pgm	200 x 200	143KB	187KB	-30.77
cat.pgm	200 X 200	159KB	214KB	-34.59
flower.pgm	200 X 200	156KB	116KB	25.64

Table 2.1:RLE compression comparison on benchmark images

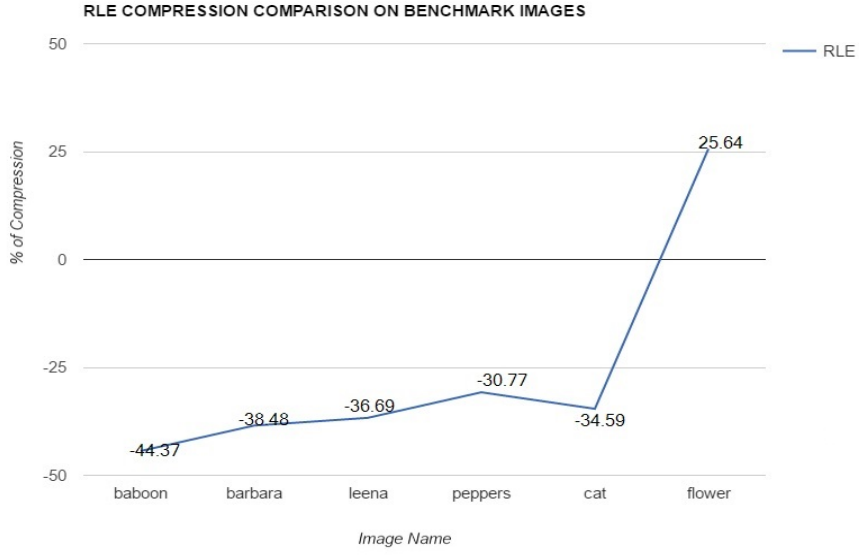


Fig 2.2 RLE compression comparison on benchmark images

RLE COMPRESSION COMPARISON ON SOLID IMAGE

IMAGE NAME	RESOLUTION	UNCOMPRESSED SIZE	COMPRESSED SIZE	RLE COMPRESSION %	JPEG COMPRSSION %
chess8X8.pgm	256 x 256	224KB	13KB	94.22	99.65
sky.pgm	250 x 156	153KB	103KB	33.12	90.16
color_bar	250 x 167	138KB	14KB	89.93	98.61
google.pgm	350 x 233	266KB	29KB	89.1	96.48

Table 2.2 :RLE compression comparison on solid images

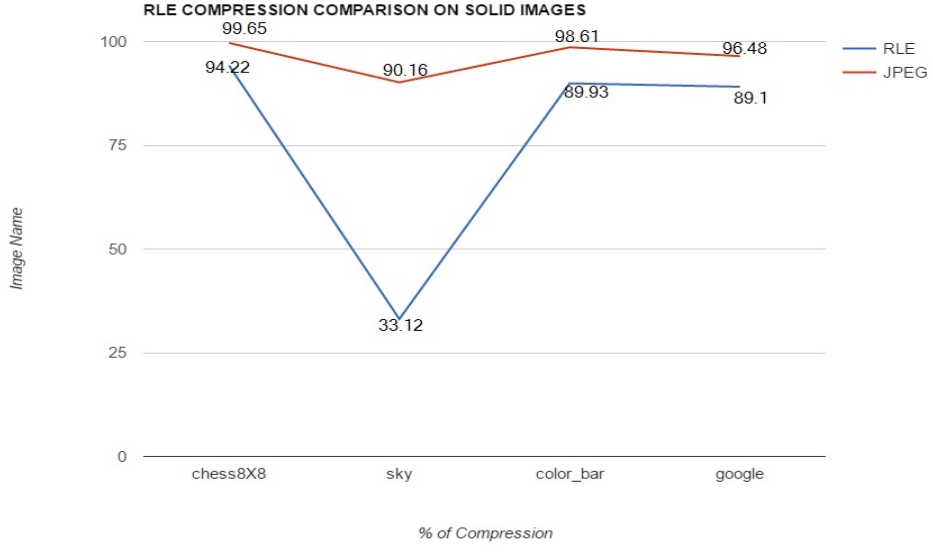


Fig 2.3 RLE compression comparison on solid images

2.1.5 Advantages and Drawbacks

This algorithm is very easy to implement and does not require much CPU horsepower. Execution rate of RLE is very fast. The time complexity and space complexity reduced rapidly compare to the conventional method.

But there is some restriction, i.e. it will work efficiently on redundant sequential data and long run length of input string . In case of short run length sequence, if we apply RLE then space complexity will increase. So, RLE compression only efficient with files that contains lots of repetitive data. These can be text files if they contain large lots of spaces for indenting but line-art images that contain white or black areas are far more suitable. Computer generates color images (e.g. architectural drawings) can also give fair compression ratios (it has been discussed in chapter 1). Compression ratio of RLE is low as compared to other algorithms.

CHAPTER 3

Burrows-Wheeler Transform

3.1 Introduction

Burrows-Wheeler Transform (BWT) is used in Data Compression techniques such as bzip. It was invented by Michael Burrows and David Wheeler in 1994 at DEC System Research Center in Palo, Alto, and California. It is not only used to compressed raw data, but widely used in Genome compression also.

BWT is a Pre-compression Algorithm, it transform the string in such a way that it is more amenable for compression using Run-Length Encoding (RLE) Data Compression Algorithm. The BWT rearranges the String which forms a cluster consisting of repeated character in order to increase the spatial locality, but we also need to assure that doing such transformation will not hurt the original string formation, and this criteria is assured by the BWT algorithm.

The given image in which BWT will work and gives the effective result only when the image consisting of repeated pixels throughout the image, it need not to be one after another. Generally, the size of an image is consisting of several pixels ranging from 500pixels – 20Mega pixels and if a image consisting of pixels more than 256 pixels, it can be assured that more than one pixel will have same pixel values as per PIGEON HOLE principal, if you consider the all PIGEON as total no pixels on a image and the color space(8bit-256 Colors) as HOLE, then more than one PIGEON will accumulated each HOLE and this phenomenon of an image is exploit by the BWT Algorithm. BWT try to recollect all the same value pixels into independent overlapping cluster, and thus RLE Compression Algorithm can be applied efficiently[5].

The main heart of the BWT is to rearrange a Random Run Length of Pixel into Similar Run Length of Pixels, keeping in mind that we can get back the initial Pixel formation at the time of the decompression.

The Transform is done by sorting all rotation of the horizontal Pixel formation into Lexicographical Order, consider this example given further:

* *Word expansions have been mentioned in* ***Abbreviation*** *section*

(For simplicity of understanding and explanation we considering the English String consisting of English Alphabets instead of Pixel values)

TABLE-3.1 Burrow-Wheeler Forward-Transofrmation

INPUT	ALL ROTATION	SORTING ALL ROWS INTO LEXIOGRAPHICAL ORDER	BWT CODE
$ B A N A N A	$ B A N A N A A $ B A N A N N A $ B A N A A N A $ B A N N A N A $ B A A N A N A $ B B A N A N A $	$ B A N A N **A** A $ B A N A **N** A N A $ B A **N** A N A N A $ **B** B A N A N A **$** N A $ B A N **A** N A N A $ B **A**	**A N N B $ A A**

*** $ is an augmented character used for alignment purpose, $ have highest precedence.*

Here we can observe the String **[B A N A N A]** is not applicable for RLE compression Algorithm because it doesn't have running of similarity character thus RLE fail to works efficiently on **[B A N A N A]**, but after applying BWT we get **[A N N B $ A A],** here we can apply RLE efficiently and after applying RLE we get **[A 2N B $ 2A],** for this example we were not getting any compressed string, but our goal is to explain that a String which is not working efficiently on RLE, BWT transform that same String for RLE compatible string, BWT work more efficiently for String of longer length comparing with our example for explanation.

3.2 Structure of the Burrows-Wheeler Compressor

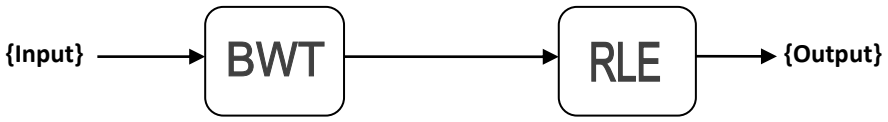


Figure 3.1: Structure of the Burrows-Wheeler Algorithm

The compressor algorithm consists of 2 stages. The first stage of the algorithm is the Burrows-Wheeler Transform (BWT). The main aim of the 1st stage is to sort the pixels of the input image which results that identical pixels are close to each other, or better to sequence of identical pixels.

The 2nd stage of the algorithm is the Run Length Encoding (RLE). In this stage, the sequence of the identical pixels are decoded in such a way that redundant pixels values will be omitted by the single pixels value with their frequency of their occurrence in respect to their run- length matching.[6] It must be mentioned at this point that first

* *Word expansions have been mentioned in **Abbreviation** section*

of the compressor algorithm don't compress the data, the main compressor is done only by the 2nd stage(RLE), 1st stage is just a pre-compressing adjustment of the data set which can be amenable for the compression by the second stage. Below we discuss more detailed about the first stage (BWT) and for second stage we already discussed earlier (RLE).

First stage: Forward Burrows-Wheeler Transform

The aim of the BWT is to sort the pixels value of the input image in such a way in results that identical pixels are close to each other in order to increase the spatial locality as per locality of reference concerns. Now we look detailed at the techniques of the BWT:

Process steps:

1. ***Order the input n (n is the length of the input), augment '$' at the start of the input, rotate each row one character to the right compared to the previous row till '$' reaches the end.***
2. ***Preserve all the rows in their respective orders.***
3. ***Sort all the rows lexicographically.***
4. ***The output of this stage is the Last Column from each row we get after Step 3.***

The detailed explanation of the above algorithm was explained earlier in tabular form on **Table-3.1** after getting the BWT code we apply RLE algorithm which was discussed earlier on this dissertation.

The Space complexity is time complexity $\mathbf{O(n^2)}$ to store all posible rotation, the time complexity for the Stage 1(BWT) it can be clearly stated that we need to sort the each possible rotation string of the horizontal pixels with respect in terms of their lexicographical order on $\mathbf{O(n^2)}$ times quick sort as applicable and the total time complexity can be given as :

For QUICK SORT ON LIST OF STRING recursive equation as follows[5]:
$\mathbf{T(n)=2T(n/2)+\theta(n^2)}$ where $\theta(n^2)$ is for partitioning,and after partitioning the pivot is placed at the n/2 location and QUICK SORT call recursively, Solving we get $\mathbf{\theta(n^2)}$.

$$\underset{rotation}{\mathbf{O(n^2)}}+\underset{sorting}{\mathbf{\theta(n^2)}}+\underset{BWT\ code}{\mathbf{O(n)}}+\underset{rle}{\mathbf{O(n)}}\approx\mathbf{\theta(n^2)}$$

where n is the no of pixels on raw image

Second Stage: Run Length Encoding (RLE)

Run Length Encoding (RLE) was discussed earlier on **SECTION 1.**

3.3 Structure of the Burrows-Wheeler De-compressor

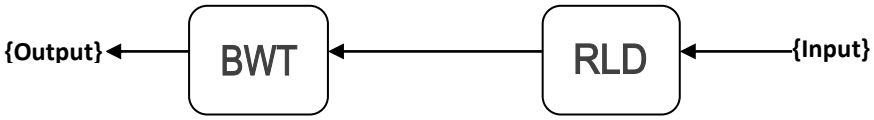


Figure 3.2: Decompression of the Burrows-Wheeler Algorithm

The de-compressor consists of again two steps just like the compressor but here the we do all the operation in reverse order. Now we look detailed of the techniques from the Burrows-Wheeler Back transform. As Run Length Decoding (RLD) has been discussed earlier on **SECTON 1.**

3.3.1 Burrows-Wheeler Back transform

The Burrows-Wheeler Back transform is the most complicated method of the Burrows-Wheeler Algorithm. We get the BWT code as **[A N N B $ A A]** after decoding from the Run-Length Decoder (RLD). We labeled this BWT code as R-Column and assigned numerical suffix value to the alphabets to identify each alphabet with their proper positioning. Now we look detailed of the technique from the Burrows-Wheeler Back transform step by step that how we get back the original string[6] **[B A N A N A].**

Process steps:
1. Sort the R-column of the input alphabetically => L-column
2. Assigned the precedence suffix in strictly increasing order on each alphabet.
3. Set the initial Start Symbol to '$' from F-Column. 3. From the standing symbol of L-Column, save its respective R-Column alphabet into the save order list serially.
4. From the standing R-Column alphabets visits its respective L-column alphabets.
5. Repeat Step 3 until we reach '$' from R-Column.

The output of this stage consists of all saved alphabets on from bottom to up on the save order list.

STEP 1: Table 3.2.1 Burrow-Wheeler Back-Transformation Table-I

Save Order	L-Column(Sorted BWT code)	R-Column(BWT Code)
-	$	A_1
-	A_1	N_1
-	A_2	N_2
-	A_3	B_1
-	B_1	$
-	N_1	A_2
-	N_2	A_3

STEP 2: Table 3.2.2 Burrow-Wheeler Back-Transformation Table-II

Save Order	L-Column(Sorted BWT code)	R-Column(BWT Code)
A_1	***Start*** → $ →	A_1
-	A_1	N_1
-	A_2	N_2
-	A_3	B_1
-	B_1	$
-	N_1	A_2
-	N_2	A_3

STEP 3: Table 3.2.3 Burrow-Wheeler Back-Transformation Table-III

Save Order	L-Column(Sorted BWT code)	R-Column(BWT Code)
A_1	***Start*** → $ →	A_1
N_1	A_1 →	N_1
-	A_2	N_2
-	A_3	B_1
-	B_1	$
-	N_1	A_2
-	N_2	A_3

STEP 4: Table 3.2.4 Burrow-Wheeler Back-Transformation Table-IV

Save Order	L-Column(Sorted BWT code)	R-Column(BWT Code)
A_1	*Start* → $	A_1
N_1	A_1	N_1
A_2	A_2	N_2
-	A_3	B_1
-	B_1	$
-	N_1	A_2
-	N_2	A_3

We do this iteration up to 7-times in total, so the final table after 7th steps is given bellow:

STEP 7: Table 3.2.5 Burrow-Wheeler Back-Transformation Table-V

Save Order	L-Column(Sorted BWT code)	R-Column(BWT Code)
A_1	*Start* → $	A_1
N_1	A_1	N_1
A_2	A_2	N_2
N_2	A_3	B_1
A_3	B_1	$ ← *Stop*
B_1	N_1	A_2
$	N_2	A_3

After completion of **Step 7** we get the String from Bottom to up is **[$ B A N A N A]** from **Save Order list** and after excluding **'$'** we get **[B A N A N A]** and by this we successfully reconstruct the original string from the compress RLE code.

Coming to time complexity for the Burrows-Wheeler Back transform it can be clearly stated that we need to sort the BWT code of R-Column to L-Column which is of length $C_1 n$ which will be **$\approx\theta(n2)$** times.

The Storage complexity is **O(n)** to store the sorted BWT code.

$$\underset{\text{sorting}}{O(n \log n)} + \underset{\text{rev BWT CODE}}{O(n^2)} + \underset{\substack{\text{RLE} \\ \text{decompression}}}{O(n)} \approx \theta(n^2)$$

where n is the no of pixels

3.4 BWT Compression Comparisons

BWT COMPRESSION COMPARISON ON BENCHMARK IMAGES

IMAGE NAME	RESOLUTION	UNCOMPRESSED SIZE	COMPRESSED SIZE	% of COMPRESSION
baboon.pgm	200 x 200	151KB	77KB	49.01
barbara.pgm	200 x 200	145KB	71KB	51.03
leena.pgm	200 x 200	139KB	72KB	48.2
peppers.pgm	200 x 200	143KB	69KB	51.75
cat.pgm	200 X 200	159KB	77KB	51.75
flower.pgm	200 X 200	156KB	42KB	73.08

Table 3.3:BWT compression comparison on benchmark images

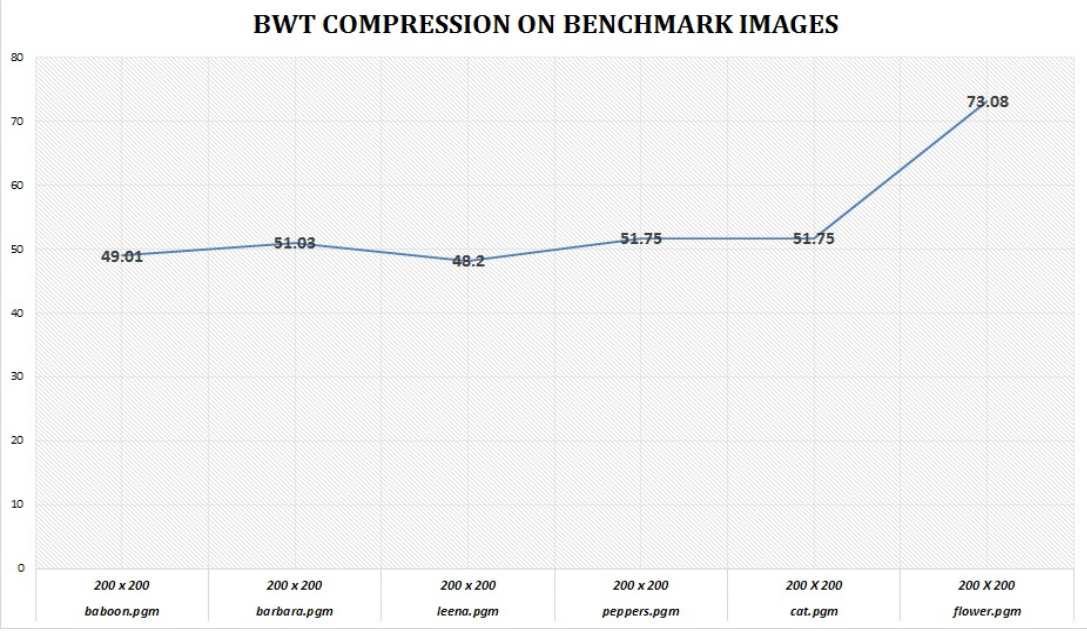


Fig 3.3 RLE compression comparison on benchmark images

BWT COMPRESSION COMPARISON ON SOLID IMAGE

IMAGE NAME	RESOLUTION	UNCOMPRESSED SIZE	COMPRESSED SIZE	BWT COMPRESSION %	JPEG COMPRESSION* %
chess8X8.pgm	256 x 256	224KB	1.02KB	99.54	99.65
sky.pgm	250 x 156	153KB	46KB	70.2	90.16
color_bar	250 x 167	138KB	707B	99.48	98.61
google.pgm	350 x 233	266KB	10.7KB	95.97	96.48

Table 3.4 :RLE compression comparison on solid images

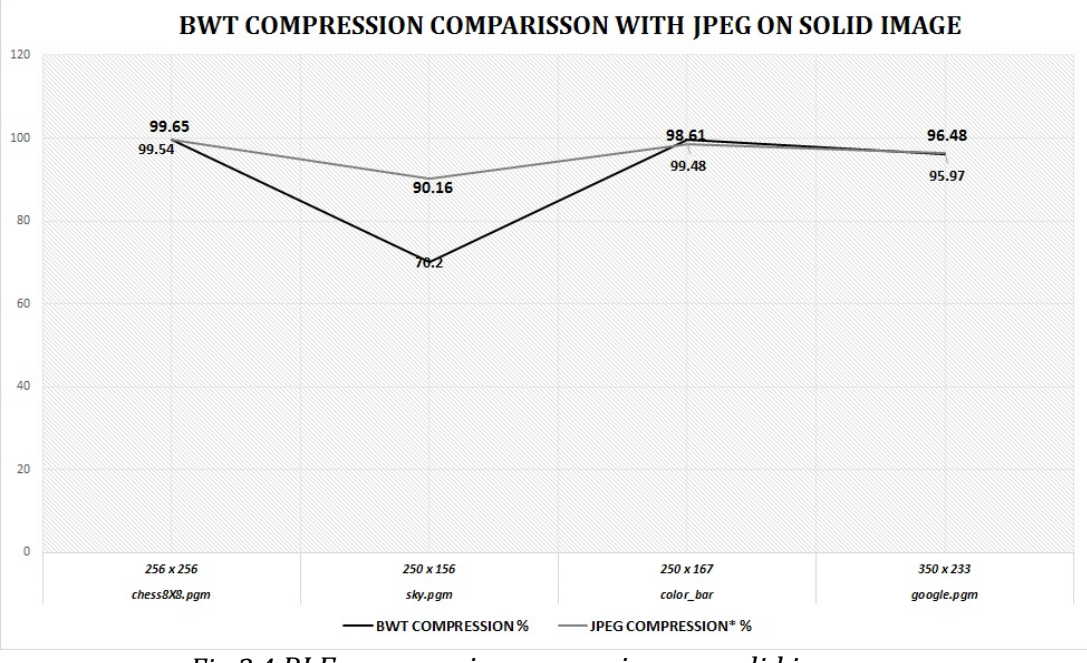


Fig 3.4 RLE compression comparison on solid images

3.5 Drawbacks

Twenty one years have now passed since the introduction of the BWT. In these twenty five years our understanding of several theoretical and practical issue related to the BWT has significantly increased. We have found some interesting concepts on decompression from BWT code, which uses the storage efficiently and also the time complexity and space complexity reduced rapidly compare to the conventional method, various variant of BWT has been published by the several researcher throughout the world on this twenty one years.

The biggest drawbacks of any variant of BWT-based algorithm is that they are not on-line algorithm, i.e. they required all the dataset at the time of the computational without seeing the extreme end of the data, BWT will not able to compute any possible outcome, not only this they must process a large portion of the data before a single output bit can be produced, which is still a storage complexity issue as earlier discussed[7]. Many researchers try to developing on-line counterpart of BWT-based compressors which will able to work on on-line data set.

CHAPTER 4

LEMPEL-ZIV-WELCH

4.1 Introduction

LZ77 and LZ78 are the two lossless data compression algorithms which were published in papers by Abraham Lempel and Jacob Ziv in 1977 and 1978. These two algorithms are also known as LZ1 and LZ2 respectively. In the year 1984 Terry Welch published an improved implementation of LZ78. This improvised algorithm is known as Lempel-Ziv-Welch (LZW). LZW is a universal lossless data compression algorithm. It provides a fast symmetric way of compressing and decompressing of data.

LZW is a dictionary-based compression algorithm. It means that instead of tabulating character counts and building trees (as for Huffman encoding which has been described in chapter 2), LZW encodes data by referencing a dictionary. Thus to encode a substring, only a single code number, corresponding to that substring index in the dictionary and needs to be written to the output file.

This is the foremost technique for general purpose data compression due to its simplicity and versatility. We can compress text, executable code, and similar data files to about one-half their original size. LZW algorithm also performs well when presented extremely redundant data files, such as tabulated numbers computer source code etc. Compression ratios (it has been discussed in chapter 1) of 5:1 for these cases.

LZW compression is always used in GIF* image files, and offered as an option TIFF* and PostScript.[8]

* *Word expansions have been mentioned in* ***Abbreviation*** *section.*

LZW is the basis of several computer utilities that claim "*double the capacity of your hard drive*"[9]. This algorithm is simple to implement, and has the very high throughput in hardware implementations.

4.2 Algorithm Description

LZW compression replaces strings of characters with single codes. It does not do any analysis of the incoming text. Instead, it just adds every new strings of character it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters.

The scenario described by Welch's paper in 1984 encodes sequences of 8 bit data as fixed length 12 bit codes. The codes from 0 to 255 represent one character sequences consisting of the corresponding 8-bit character. The codes 256 through 4095 are created in dictionary for sequences encountered in the data as it is encoded. At each stage in compression input bytes are gathered into sequence until the next character would make sequence for which there is no code yet in the dictionary. The code for the sequence (without the character) is added to the output, and a new code (for the sequence with that character) is added to the dictionary. As the encoding continues, LZW identifies repeated sequences in the data and adds them to the code table.

The key point is that the sequence from the input file is not added to the code table until it has already been placed in the compressed file as individual characters (codes 0 to 255). This is important because it allows the decompression program to reconstruct the code table directly from the compressed data, without having the code table separately.[8]

The decoding algorithm works by reading a value from the encoded input and outputting the corresponding string from the initialized dictionary. In order to rebuild the dictionary in the same way it was built in encoding. It also obtains the next value from the input and adds to the dictionary. The concatenation of the current string and the first character of the string obtained by decoding the next input value, or the first character of the string is just output if the next value can't be decoded. If the next value is unknown to the decoder, then it must be the value that will be added to the dictionary this iteration, and so its first character must be the same as the first character of the current string being sent to decoded output. The decoder then proceeds to the next input value (which was already read in as the "next value" in the previous pass) and repeats the process until there is no more input, at which point the final input value is decoded without any more additions to the dictionary.[8]

4.2.1 LZW Compression

High Level view

Process steps:

1. ***Initialize the string table with all strings of length one.***
2. ***Find the longest string P in that dictionary that matches with the current input.***
3. ***Emit the dictionary index for P to the output and remove P from the input.***
4. ***Add P followed by the next symbol in the input to the dictionary.***
5. ***Go to step 2.***

The following example will explain the Encoding process.

Example: The LZW algorithm to compress the string

BABAABAAA

Table 4.1 : Results of compression

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

The output sequence : <66><65><256><257><65><260>

4.2.2 LZW Decompression

LZW creates the string table during decompression. It starts with the first 256 table entries initialized to single characters. The string table is updated for each character in the input stream, except the first one. Decoding is achieved by reading codes and translating them through the code table being built.

Process steps:

1. ***Initialize the string table with all strings of length one.***
2. ***Get the first input code.***
3. ***Output the translation code.***
4. ***Initialize a character variable C with NULL value.***
5. ***Get the next input code.***
6. ***Check either the input code is present in the string table or not.***
7. ***If the input code is not in the table then get the translation of the previous input code and concatenate with a variable 'C'.***
8. ***Otherwise get the translation code for next input code.***
9. ***Output the translation code.***
10. ***Assign the variable C with the first character of the translation code.***
11. ***Concatenate the previous input code with the variable C and add it to the table.***
12. ***Assign the previous input code with the new input code.***
13. ***Go to step 5.***

Example:

The following example will explain the decompression logic

LZW to decompress the previous compressed output sequence

<66><65><256><257><65><260>

Table 4.2: results of decompression

DECODER OUTPUT	STRING TABLE	
String	codeword	string
B		
A	256	BA
BA	257	AB
AB	258	BAA
A	259	ABA
AA	260	AA

The decompressed output sequence: BABAABAAA

As the dictionary size is fixed and independent of the length , LZW is O(n) as each byte is only read once and the complexity of the operation of each character is constant.[10]

4.3 LZW COMPRESSION COMPARISONS

LZW COMPRESSION COMPARISON ON BENCHMARK IMAGES

IMAGE NAME	RESOLUTION	UNCOMPRESSED SIZE	COMPRESSED SIZE	% of COMPRESSION
baboon.pgm	200 x 200	151KB	109KB	27.81
barbara.pgm	200 x 200	145KB	102KB	29.66
leena.pgm	200 x 200	139KB	95KB	31.65
peppers.pgm	200 x 200	143KB	143KB	33.57
cat.pgm	200 X 200	159KB	100KB	37.41
flower.pgm	200 X 200	156KB	59KB	62.18

Table 4.3:LZW compression comparison on benchmark images

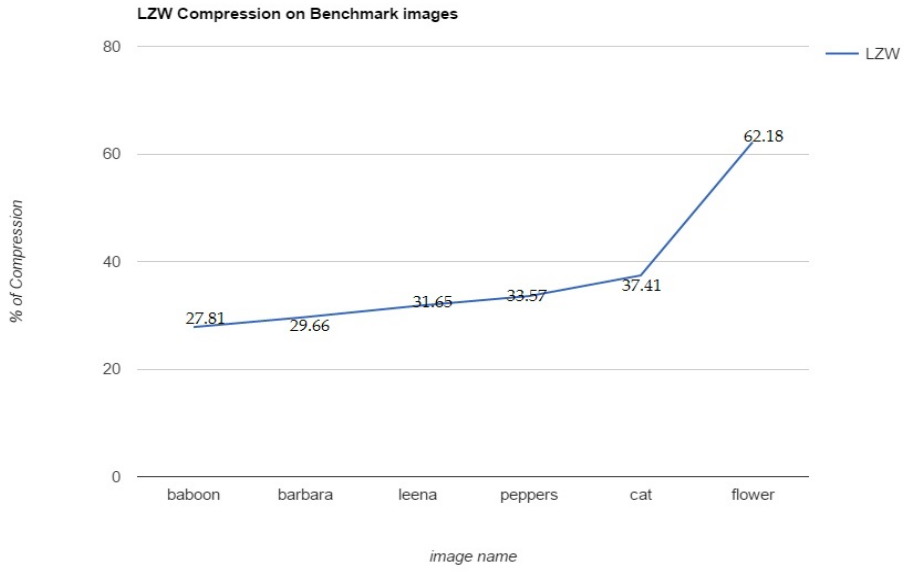


Fig 4.1: LZW compression comparison on benchmark images

LZW COMPRESSION COMPARISSON ON SOLID IMAGE

IMAGE NAME	RESOLUTION	UNCOMPRESSED SIZE	COMPRESSED SIZE	BWT COMPRESSION %	JPEG COMPRESSION* %
chess8X8.pgm	256 x 256	224KB	13KB	94.22	99.65
sky.pgm	250 x 156	153KB	57KB	62.99	90.16
color_bar	250 x 167	138KB	9KB	93.53	98.61
google.pgm	350 x 233	266KB	10.7KB	90.98	96.48

Table 4.4:LZW compression comparison on solid images

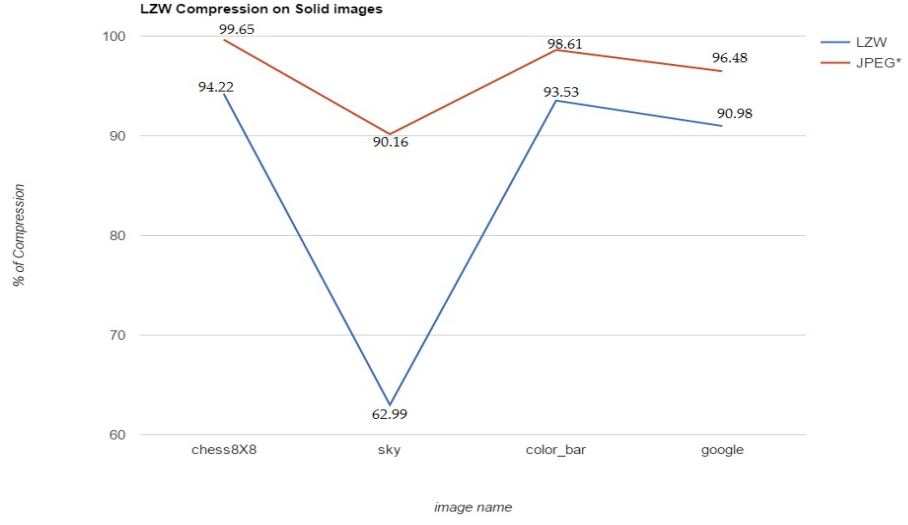


Fig 4.2: LZW compression comparison on benchmark images

4.3 ADVATEGES AND DRAWBACKS

The LZW algorithm is very fast and very simple. There is no need to analyze the incoming text. The size of files usually increases to a great extent when it includes lots of repetitive data or monochrome images. LZW compression is the best technique for reducing the size of files containing more repetitive data. LZW compression is fast and simple to apply. Since this is a lossless compression technique, none of the contents in the file are lost during or after compression. The decompression algorithm always follows the compression algorithm. LZW algorithm is efficient because it does not need to pass the string table to the decompression code. The table can be recreated as it was during compression, using the input stream as data. This avoids insertion of large string translation table with the compression data. [11]

Although the algorithm is very simple but implementation of this algorithm is complicated mainly because of the management of the string table. Files that do not contain any repetitive at all cannot be compressed much. The method is good at text files but not as good as the other types of file. The amount of storage needed is indeterminate as it depends on the total length of all the strings. Also problem involves while searching the strings. Each time a new character is read in the algorithm has to search for the new string formed by (string + character). Each and every time a new character is read in a string table has to be searched for a match. If a match is not found then a new string has to be added to the string table. This causes two problems. Firstly string table can get very large. Secondly if string length average even as low as three or four character each the overhead of storing a variable length string and its code could easily reach seven or eight bytes per code.

CHAPTER 5

COMPARATIVE ANALYSIS OF LOSSLESS IMAGE COMRESSION

5.1 Conclusion

Run-length Encoding image compression gives a very high rate of compression for computer generated images like architechtural drawings. But when it comes to practical images instead of compressing the actual image it increases the compressed file size. For almost practical images it yeilds a negetive compression ratio which makes Run length Encoding not a non effective compression algorithm. Though RLE requires less CPU time and memory.

As we discussed earlier RLE compatible images can get a decent compression ratio. Using a variance of Block sorting algorithm, **Burrows-Wheeler Transform** algorithm can be judged as a pre-compression algorithm which will try to sort the pixel values into different blocks containing same pixel value adjacent with each other and then we can apply **RLE** to get a better compression ratio. The analysis we have done on benchmark images RLE was yeilding averagely negetive comression ratio whereas, **BWT** algorithm was giving averagely 50% compression. For solid color images RLE is giving averagely more than 90% compression. But when wy apply BWT on same images we are getting above 99% compression, which is more than JPEG compression *[QUALITY: Good]*. But the main drawback of BWT algorithm is time and storage complexity.

The above mentioned drawback of BWT algorithm can be minimized by dictionary based algorithm **Lemple-Ziv-Welch.** Though the resource utilization and the time complexity get reduced but when it comes to compression ratio BWT gives a better compression than LZW. But in case of practical image LZW gives better compression than RLE.The comparitive study has been given below.

Comparison of lossless algorithms[12]

ALGORITHM	RLE	BWT	LZW
ADVANTAGES	1.Easy to implement 2.Produce optimal and compact understandable code 3. Does not require much CPU and memory resources 4.Work good on Computer-generated images	1.Gives phenomenal compression ratio on practical images as a lossless algorithm. 2.For Computer-generated images it give average better compression than JPEG. 3.When dataset is large it give comparable more compression ratio.	1.Easy to implement 2. Fast compression. 3. Dictionary based technique. 4.Work with different data format like flat file,normal data file etc.
DISADVANTAGES	1.For an RLE uncompatable images RLE can increase the output file size. 2.Doesn't work on text file unless use any predictive algorithm.	1.BWT alone is not a compression algorithm it use RLE to do compression. 2.It have a huge time and storage complexity.	1. Management of string table is difficult. 2. Amount of storage needed is indeterminate. 3. Royalties have to be paid to use LZW compression for commercial purpose.
APPLICATION	Used in JPEG,ZIP.	Used in BZIP2,genome compression in Bio-informatics. Bowtie,BWA and SOAP2.	Used in TIFF and GIF files.

5.2 Results

LOSSLESS COMPRESSION COMPARISSON ON BENCHMARK IMAGES

IMAGE NAME	RESOLUTION	RLE Compression %	BWT Compression %	LZW Compression %
baboon.pgm	200 x 200	-44.37	49.01	27.81
barbara.pgm	200 x 200	-38.48	51.03	29.66
leena.pgm	200 x 200	-36.69	48.2	31.65
peppers.pgm	200 x 200	-30.77	51.75	33.57
cat.pgm	200 X 200	-34.59	51.75	37.41
flower.pgm	200 X 200	25.64	73.08	62.18

Table 51:Lossless compression comparison on benchmark images

GRAPHICAL REPRESENTAION OF COMPARITIVE OF LOSSLESS ALGORITHMS

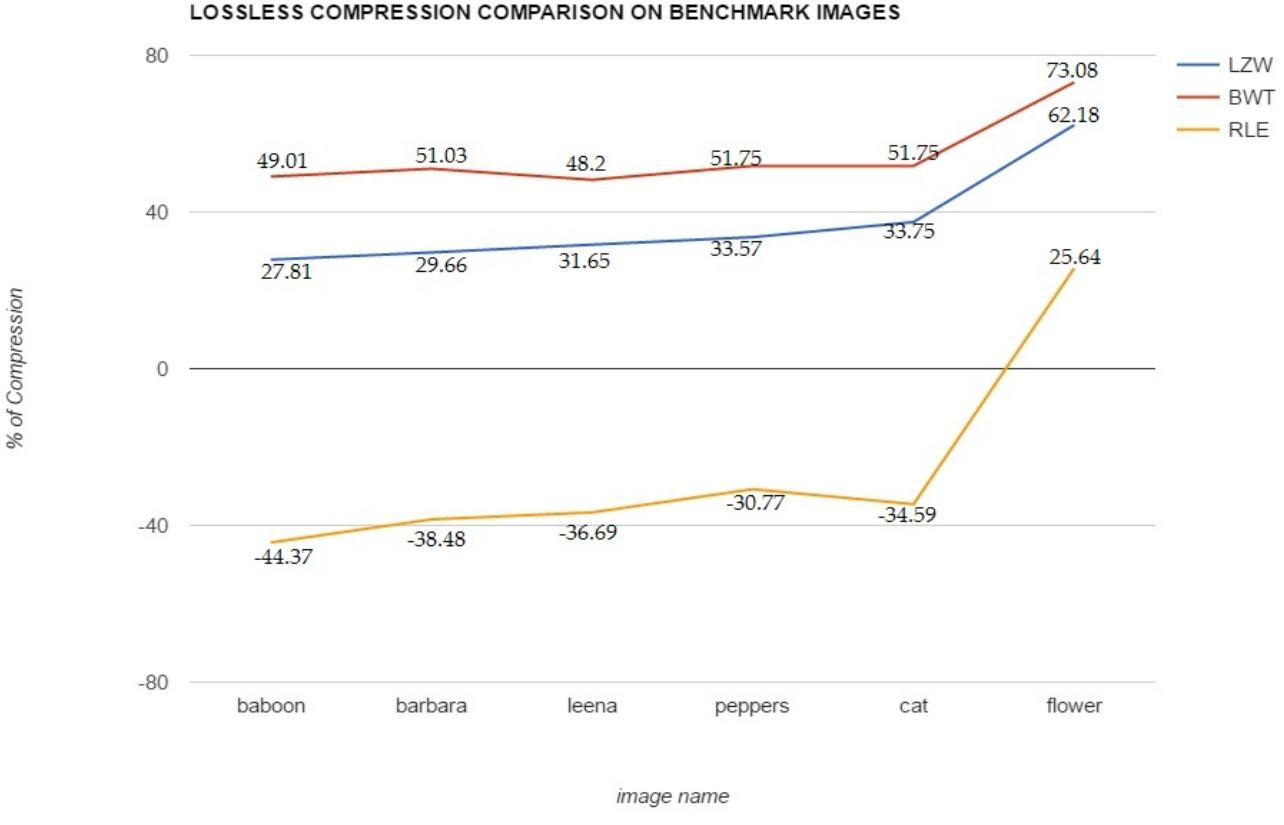


Fig 5.2: Lossless compression comparison on benchmark images

CHAPTER 6

Block Truncation Coding

6.1 Introduction

Block truncation coding[13] is an Image compression technique that effectively works on 8bit grey-scale images. From the beginning days storing digital images with their resolution quality in reduced size being a most challenging job for us. BTC was first proposed by e.g. and O.R. Mitchell at Purdue University. Since the data of a gray-scale image is 8-bit so that the values can range from 0-255 in a grey-scale image depending upon the pixel intensity. So this varying range matrix of an image increases the size of the image, we know that an image pixel value is stored in file with matrix formation. This technique divides the original images into blocks and then uses a quantiser to reduce the number of grey levels in each block whilst maintaining the same mean and standard deviation. For compressing the image a m x m sub-block matrix will be

retrieved each time here m is a positive integer from the series of 2^n for example M=2,4,8,16 etc. Choosing block size of 4x4 gives a compression ratio of 4:1.

From a 4x4 block mean and standard deviation is calculated for the pixel values this have to be calculated for each of the block, this is the main statistical calculation of BTC. The two level quantization is for an image compression is like -

y(i,j) = 1 for x(i,j)> x mean,

y(i,j)=0 for x(i,j)< x mean,

here x(i,j) is the pixel value of original image and y(i,j) is the pixel value of the compressed image. The 16 bit block of 0's and 1's is transmitted along with the value of mean and standard deviation of that sub-block. The sub-block reconstruction is made depending upon the values of mean and standard deviation which are preserved.

$$a = \overline{X} - \overline{\sigma} \cdot \sqrt{\frac{q}{m-q}}$$

$$b = \overline{X} + \overline{\sigma} \cdot \sqrt{\frac{m-q}{q}}$$

To reconstruct the image calculated approximate value a is placed for 0 and b is placed for 1 and hence by repeating this process for each 4x4 block the image is reconstructed.

x(i,j) = a, for y(i,j) = 0

x(i,j) = b, for y(i,j) = 1

For an image where pixel values in 4x4 sub-block are all same this algorithm work effectively best in this type of images. The BTC algorithm was used for compressing Mars Pathfinder's rover images.

6.2 Block Truncation Coding Algorithm :

Process steps:
1. Divide the Image in block of 4x4 pixels.
2. Now for each block calculate the mean(and the standard deviation (σ).
3. Now the compression is performed as.

$$y(i,j)=\begin{cases}1 & x(i,j)>\overline{X}\\ 0 & x(i,j)<\overline{X}\end{cases}$$

here x(i,j) is the pixel element of original image and y(i,j)is the pixel element of compressed image corresponding to the original image.

4. The block is now to be transmitted with the value of mean($\overline{X}$) standard deviation(σ), the reconstruction is made on depending upon the two values a,b.

$$a=\overline{X}-\overline{\sigma}\cdot\sqrt{\frac{q}{m-q}}$$

$$b=\overline{X}+\overline{\sigma}\cdot\sqrt{\frac{m-q}{q}}$$

here m is the total no. of pixels and q is the no. of pixel greater than mean. Now image is reconstructed and original pixel values get as

x(i,j) = a for y(i,j) = 0

x(i,j) = b for y(i,j) = 1

Note : This process have to repeated for each of the 4x4 sub-block.

Complexity of the algorithm

The Complexity of this algorithm is O(n^2).

For this algorithm a 8x8 or a 16x16 block can also be taken by which the compression will become faster and we can achieve more better compression but the computation complexity in this case will increase more and the image quality will fall i.e. after reconstruction we will get poor quality image.

Example for testing[14] -

A 4x4 sub-block is taken as

$$\begin{matrix} 245 & 239 & 249 & 239 \\ 245 & 245 & 239 & 235 \\ 245 & 245 & 245 & 245 \\ 245 & 235 & 235 & 239 \end{matrix}$$

Now we need to calculate the mean and the standard deviation for this data and here the result are

$$\overline{X} = 241.875$$

$$\sigma = 4.36$$

So, the decoded marix will look like -

$$\begin{matrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{matrix}$$

To reconstruct the original matrix we need to preserve the value 241.875 and 4.36 here we can see that the pixel values are replaced with '0' or '1'.
The reconstruction is done by calculating the value of a and b and the values are

$$a = 236.93$$
$$b = 245.71$$

and here ‘0’s are replaced by a and ‘1’s are replaced by b and now the decoded matrix looks like

$$\begin{matrix} 245 & 236 & 245 & 236 \\ 245 & 245 & 236 & 236 \\ 245 & 245 & 245 & 245 \\ 245 & 236 & 236 & 236 \end{matrix}$$

6.3 IMAGE QUALITY MEASUREMENT

Image quality measurement[15] is an important role for measuring the output quality of the algorithm, here we are taking some benchmark images for this testing these are Leena, Peppers, Baboon, Barbara, Flower, Cat.The measurements are the compression ratio measurement, PSNR measurement and the measurement of Mean Square Error(MSE). The input raw images and compressed one will be compared here with the above mentioned measurements. All the images below are of order 200x200 and 8bit per pixel i.e. max value 255 for each pixel.

Peak Signal To Noise Ratio (PSNR)[14] :

Peak signal-to-noise ratio, often abbreviated PSNR, is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. PSNR is usually expressed in terms of the logarithmic decibel scale.PSNR is mostly used to measure the quality of compressed lossy image after reconstruction with respect to the original image. It calculates the error introduced in a pixel, by taking original image pixel value and reconstructed image pixel value then some error calculation. Higher PSNR indicates higher image quality after reconstruction.PSNR is easily calculated using mean squared error, for any image of order mxn image mean squared error calculated as

$$MSE = \frac{1}{m\,n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

where we indicates for the noise free image and K indicates the noisy approximation of that image. Now to compute PSNR we need two things.

one is MSE which we know another is maximum pixel(MAX) value in this case it is 255, and here the PSNR calculation procedure is -

PSNR is deifned as -

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

$$= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

Compression Ratio with respect to Original image size

Another important measurement is the compression ratio. The ratio is measured by uncompressed image size with respect to the compressed image size.

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

Percentage of Compression -

To calculate it first percentage of the compressed image size is computed with respect to the uncompressed image size:

$$\text{compressed file size(\%)} = \frac{100 \times compressed\ filesize}{uncompressed\ filesize}$$

Amount of file size compression is :

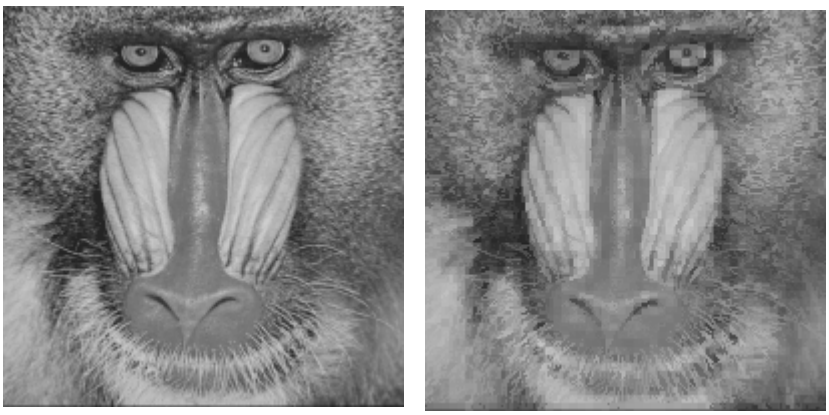
amount of compression(%) = 100 - compressed file size(%)

Bit Rate :

Bit rate and compression ratio is related, let b be the bit value per pixel(bit depth) of an uncompressed image and the compression ratio is as mentioned above then the bit rate (BR) is given by-

$$\text{BR} = \frac{b}{Compression\ Ratio}$$

Comparison of Sample Images :



(a) (b)

Figure 5.1: Baboon (a) original image. (b) compressed image

(a) (b)

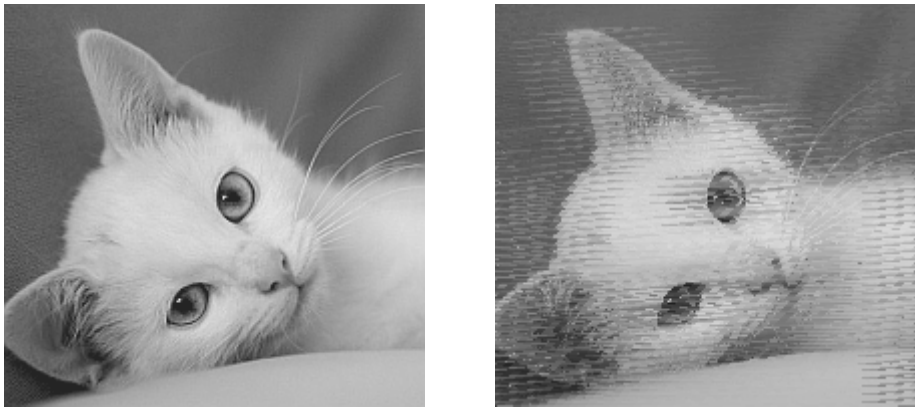
Figure 5.2: Leena (a) original image. (b) compressed image

(a) (b)

Figure 5.3: Peppers (a)original image. (b) compressed image

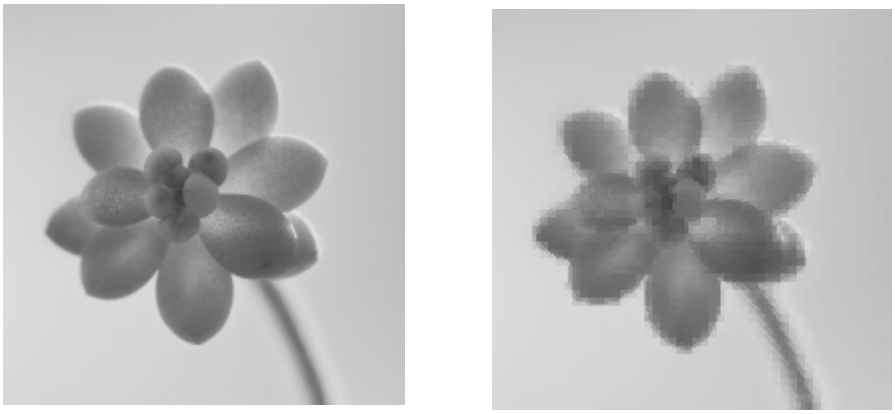
(a) (b)

Figure 5.4: Barbara (a)original image. (b) compressed image



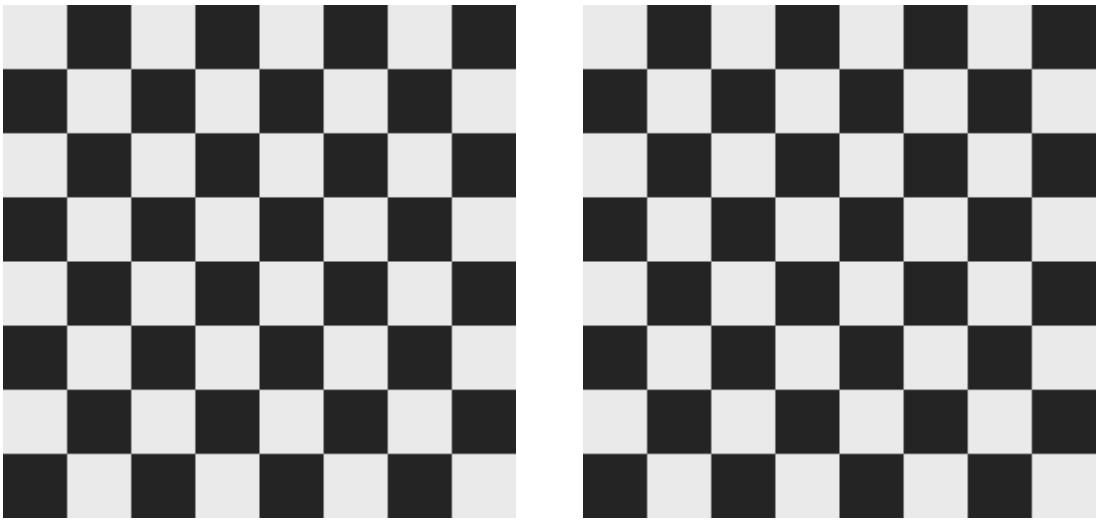
(a) (b)

Figure 5.5: Cat (a)original image. (b) compressed image



(a) (b)

Figure 5.6: Flower (a)original image. (b) compressed image



(a) (b)

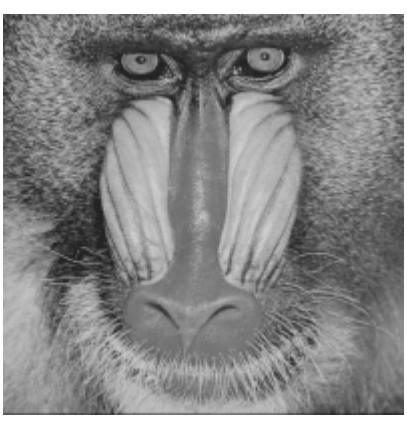
Figure 5.7 : Chessboard (a)original image. (b) compressed image

Here we are comparing the images it is easily seen that after the compression the images are losing some pixels and hence the quality is degrading for the images. If we run compression for a particular image for 2-3 times then more degrading quality will be observed and around the sharp edges.

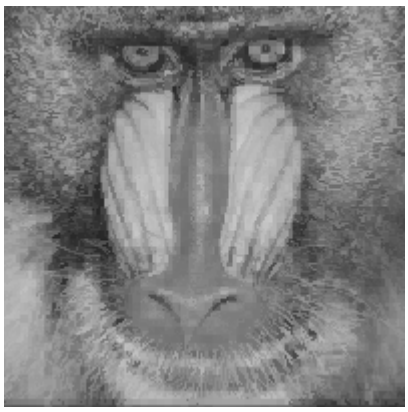
Around the sharp edges of an image the high frequency pixel values will lost and so the result comes to us, but the pixel loss is not always same it purely depends upon the image, as seen in the above images. If we see the Baboon image we can easily observe that there pixel are lost after compression but not so much, and in the chess board image the overall all quality of uncompressed image is preserved in the compressed image.

Degradation of Image Quality after repetitive Compression

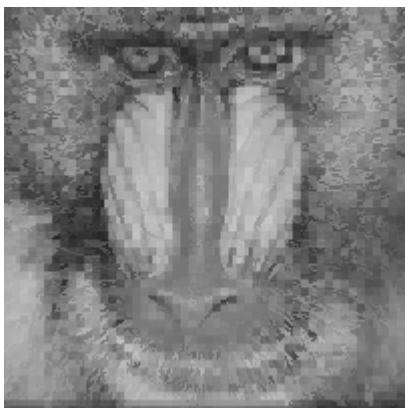
To understand the degradation of image quality compression is performed on Leena and Baboon image for 3 times and so we get some result like this-



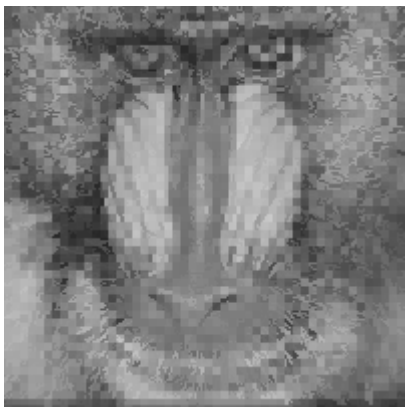
(a)



(b)



(c)



(d)

Figure 5.8: (a)original image. (b) after 1st compression (c) after 2nd compression (d) after 3rd compression

(a)

(b)

(c)

(d)

Figure 5.9 : (a)original image. (b) after 1st compression (c) after 2nd compression (c) after 3rd compression

Evaluation of Perceptual Quality [15]:

These are measurement result of PSNR, BR and compression.

	Compression(%)	PSNR	BR
Baboon	66.47	22.546	2.682
Barbara	65.743	22.707	2.740
Cat	65.574	23.083	2.754
Chessboard	65.00	17.265	2.8
Flower	68.246	31.290	2.5
Leena	64.834	24.097	2.81
Peppers	65.665	23.440	2.746

Table 5.1: Measurement parameters of BTC

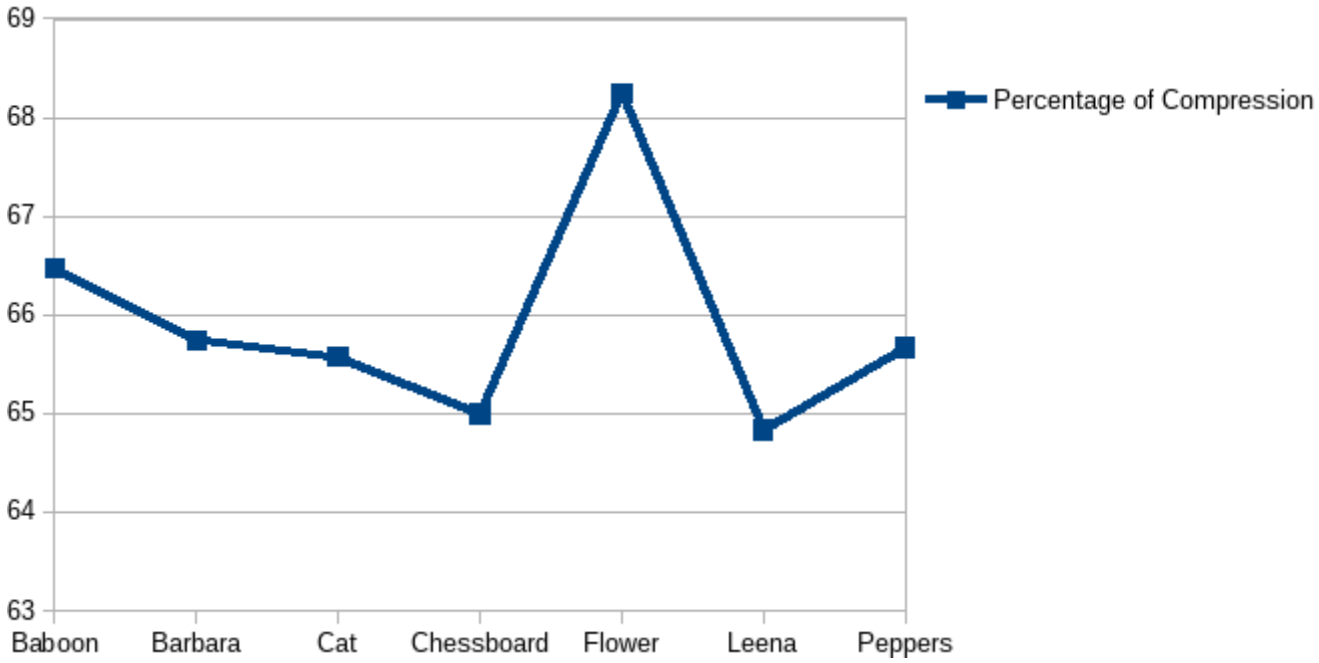


Figure 5.10: Graph for the percentage of compression

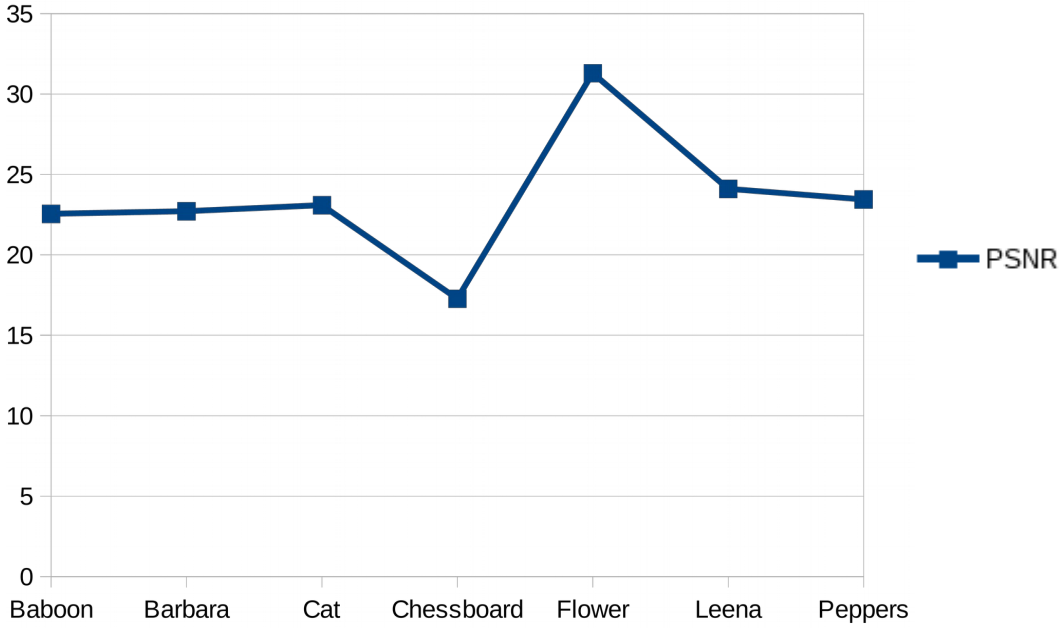


Figure 5.11: Graph of PSNR

Perceptual Quality Comparison with another paper:

We found a paper on this algorithm by Doaa Mohammed, Fatma Abou-Chadi from Man-soura University Egypt in 2004. Here we given the perceptual quality measurement of their work to comparison with our works.

	PSNR	BR
Baboon	24.7720	1.25
Barbara	28.2149	1.25
Leena	29.5552	1.25
Peppers	29.0598	1.25

Table5.2: Image Compression using BTC

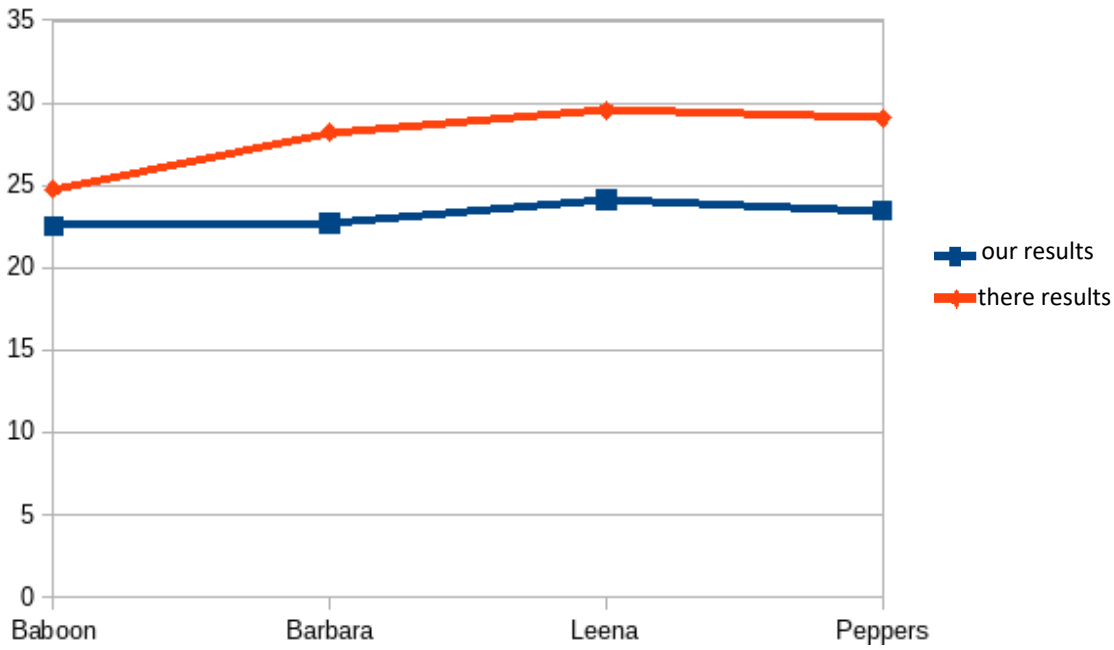


Figure 5.12: Comparison graph of PSNR

Comparison of Perceptual Quality of BTC With JPEG Lossy Image Compression Algorithm :

The comparison of BTC with JPEG[16] lossy compression for measurement as we know that JPEG is also another available standard image compression technique, so here the comparison results are

	JPEG		BTC	
	Compression(%)	BR	Compression(%)	BR
Baboon	93.5	0.521	66.47	2.682
Barbara	94.6	0.432	65.743	2.740
Cat	96.4	0.292	65.574	2.754
Chessboard	99.4	0.05	65.00	2.8
Flower	98.1	0.154	68.246	2.5
Leena	95.4	0.370	64.834	2.81
Peppers	95.2	0.386	65.665	2.746

Table 5.3: Comparison of BTC with JPEG

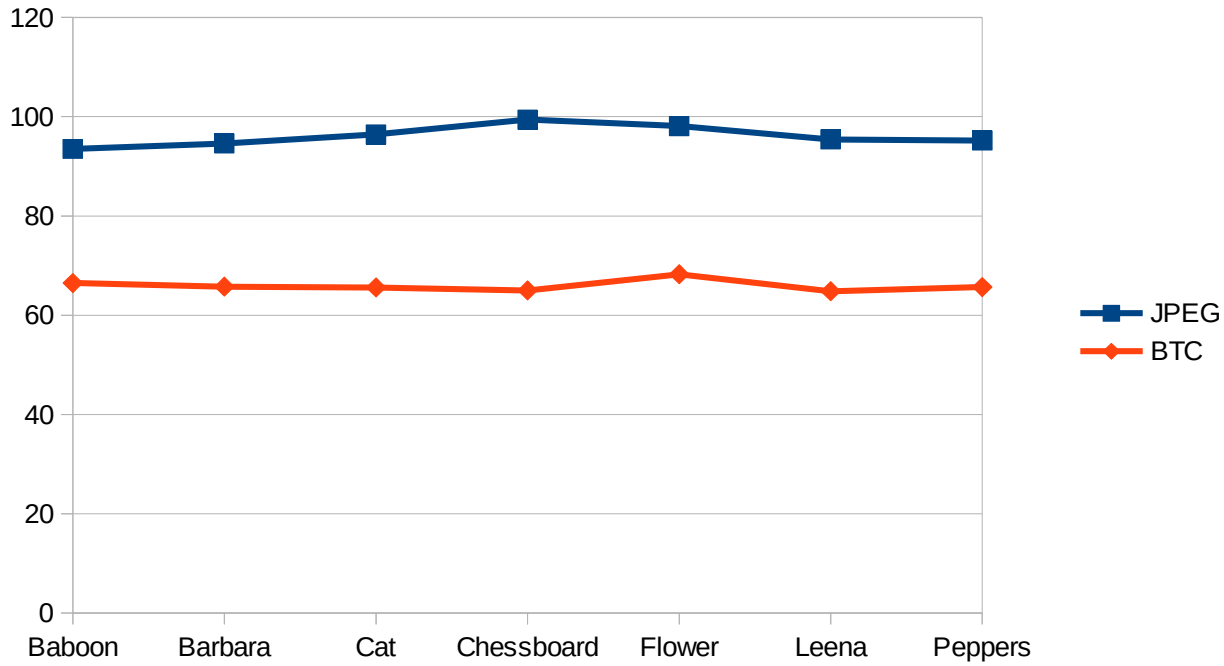


Figure 5.13: Compression comparison of BTC and JPEG

6.4 Advantages and Drawbacks :

Advantage of this algorithm is that it is very much space efficient, takes minimum space because for a mxn sub-block all data can be stored in a bit '0' or '1'. For that reason file size becomes minimum and this type of compression is ideal when a large image is to be transmitted through a slower transmission medium.

This image compression is very useful for that type of image where pixel values of a mxn sub-block cells are quite nearest. This technique is not suitable for colour images.

Some Case Study :

Here we discuss some case for this algorithm what we found during the analysis and implementation

Case 1: For a mxn sub-block all the pixel values are same but not zero for example a sub-block like

$$\begin{matrix} 192 & 192 & 192 & 192 \\ 192 & 192 & 192 & 192 \\ 192 & 192 & 192 & 192 \\ 192 & 192 & 192 & 192 \end{matrix}$$

in this case mean = 192 and σ=0 the compressed sub-block is as below -

$$\begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix}$$

For reconstruction here computation of ‘a’ should have to avoid, because there is a m-q term in the denominator which will cause a floating point exception error. All the values ‘1’ should be replaced by 192(mean) or b=mean is enough.

Case 2: For a mxn sub-block all the pixel values are same and are ‘0’ for example a sub-block like

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

Here in this case value of mean = 0, σ=0 so the compressed sub-block remains same. But a problem may arise during reconstruction, here it is easily seen that we don't need to calculate value of 'b' and it should be avoided. Because there is a term 'q' in the denominator in the equation for 'b' as we know 'q' is the number of count for pixels value greater than mean which is '0' for this case. So, computation of 'b' should be avoided otherwise the floating point exception error will occur, here just a= 0 or a = mean is enough.

These are two strong error prone cases which we found during implementation, and may become the cause of arising some other serious error during reconstruction procedure which are some times hard to be diagnosed.

6.5 Conclusion :

Once when disk space is not so avalilable like now days, the lossy image compression plays a key role for storing images on disk with compressing the size of the image. Though the image quality falls for the lossy image compression. The lossless techniques stores all the data except some cases, so the quality is preserved but the file size isn't decreased so much. The BTC algorithm have some historical value of lossy compression, in some cases where disk space is not so available or using to much disk space will cause for extra overhead, these lossy image compression also in demand in present days.

FUTURESCOPE

In this dissertation, many of important image have been presented and analyzed. The best way of fast and secure transmission is by using compression of multi-media data like images.

The dissertation works have been categorized in the following two categories viz. LOSSLESS and LOSSY Image Compression algorithm. The compression technique observed is either lossy or lossless. Always lossless compression is preferred but to achieve secrecy some image quality degradation is accepted.

The performance evaluation factors are PSNR ratio and coding decoding time for compression and compression percentage.

As a future work more focus can be on improvement of compression ratio using the new techniques,optimizing the implementation to get the optimal results in term of utilizing time and space resources. The proposed technique can be experimented on different kinds of data sets like audio, video, text as till now it is restricted to images. The experimental dataset in this project work is somehow limited; so applying the developed methods on a larger dataset could be a subject for future research which may lead to new observations and conclusions.Another apporach can be stress on future work that is to use a combination of lossless and lossy algorithm to get optimal compression ratio without lossing information to an acceptable level but getting sufficent compressing ratio.

References :

[1]. Pixel grids , bit rate and compression ratio". Broadcast Engineering. 2007 - 12 - 01. Retrieved 2013-06-05.

[2]. James D. Murray, William vanRyper (1996 - 04). "Encyclopedia of Graphics File Formats, Second Edition". O'Reilly. ISBN 1-56592-161-5. Retrieved.

[3]. "SIPI Image Database-MISC". USC University of Southern California , Signal and Image processing Institute.

[4]. Recommendation T .45 (02 / 00) : Run-length colour encoding. Internationaltional Telecommunication Union. 2000. Retrieved 2015-12-06.

[5]. Burrows, Michael ; Wheeler , David J. (1994), A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation.

[6]. Duval, Jean-Pierre (1983), "Factorizing words over an ordered alphabet", Journal of Algorithms 4 (4): 363–381.

[7]. G Manzini,Mathematical Foundations of Computer Science 1999, 1999 - Springer, "The Burrows-Wheeler transform: theory and practice",www.unipmn.it

[8]. Ziv, J.; Lempel, A. (1978). "Compression of individual sequences via variable-rate coding". IEEE Transactions on Information Theory 24 (5): 530

[9]. Steven Smith,Nwenes :(2013)."Digital Signal Processing: A Practical Guide for Engineers and Scientists":CH3:488,ISBN-9780080477329.

[10] . David Salomon, Data Compression – The complete reference, 4th ed., page 212.

[11]. Welch, Terry (1984). "A Technique for High-Performance Data Compression" (PDF). Com puter 17 (6): 8–19. doi:10.1109/MC.1984.1659158.

[12]. Suresh Yerva, Smita Nair and Krishnan Kutty, Lossless Image Compression based on Data Folding,IEEE, pp. 999-1004, 2011.

[13]. R. Mitchell and E. J. Delp, "Image Compression Using Block Truncation," Purdue University - Purdue Research Foundation, Record and Disclosure of Invention, dated April 8, 1977.

[14]. WIKIPEDIA :Block truncation coding :www.en.wikipedia.org/wiki/Block_Truncation_Coding.

[15]. CYBERJOURNALS :PSNR.MSEhttp://www.cyberjournals.com/Papers/Feb2011/02.pdf.

[16]. JPEG - Wikipedia,www.jpeg.org/, https://en.wikipedia.org/wiki/JPEG.