Lossless Image Compression by incorporating Histogram Processing stage in Burrows Wheeler Compression Algorithm (BWCA).

By
Muhammad Adnan Yousaf
[2009-NUST-MSPhd-ComE-01]

Submitted to the Department of Computer Engineering
In partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

Advisor
Dr. Muhammad SaadRehman

College of Electrical & Mechanical Engineering
National University of Sciences and Technology
2011

DECLARATION

We hereby declare that no portion of the work referred to in this Project Thesis has been submitted in support of an application for another degree or qualification of this of any other university or other institute of learning. If any act of plagiarism found, we are fully responsible for every disciplinary action taken against us depending upon the seriousness of the proven offence, even the cancellation of our degree.

COPYRIGHT STATEMENT


بسم الله الرحمن الرحيم

This thesis is dedicated to my parents

Acknowledgements

All praise to the Almighty Allah, the most Merciful and the most gracious one. Without whose help and blessings, I would not have been able to complete this research. Many thanks to my project supervisor, **Dr. Muhammad Saad** whose constant motivation, unflagging efforts and uninvolved words of wisdom ever proved a lighthouse for me, it was earnestly felt whenever we swayed. Despite his never ending assignments of university management, student counseling, project supervision and teaching, he did never mind whenever we went for an advice, within or without the time slot allocated for us.

Acknowledgement is also due to my teachers for dedicatedly instilling and imparting enlightenment to me during the course of studies and afterwards for our project. I am also very thankful to my parents for their tacit and avowed support, patience and understanding.

I would like to thank my friends who gave me confidence to face the difficulties of life. They all gave me good company and everlasting memories.

ABSTRACT

Lossless Image Compression by incorporating Histogram Processing stage in Burrows Wheeler Compression Algorithm (BWCA)

Lossless Image compression reduces the amount of data required to represent an image without losing any information.Burrows-Wheeler compression is a four stage process in which the data is transformed with the Burrows-Wheeler Transform, and then transformed with Move-To-Front, and then Run Length Encoding is performed and finally encoded with an entropy coder.BWT is a reversible context sorting algorithm and compression is achieved because sorting increase the spatial and temporal redundancy.

In this report a new stage is proposed before BWT to enhance compression ratio i.e. histogram processing stage and several other improvements have been proposed in each stage of Burrow wheeler compression algorithm. The proposed scheme is also compared with LZW Lempel Ziv Welch algorithm which is a universal dictionary based algorithm while BWCA is lexographical transform .The focus of this report is to improve the compression ratio of BWCA by introducing histogram processing stage as well as improving GST and RLE stages.

Table of Contents

CHAPTER 3

PROPOSED SCHEME

CHAPTER 4:

RESULTS

CHAPTER 5

CONCLUSION AND FUTURE WORK

Chapter 1

Introduction

1.1 Introduction to Lossless Image Compression

Lossless image compression is the technique of compressing the image in such a way that no information is lost when image is decompressed, unlike the most commonly known image compression techniques such as JPEG (Joint Photography Experts Group) in which the quality of the image and the details of the image are compromised in order to save more space and bandwidth. Lossless image compression is used widely in sensitive data such as the medical imaging and astronomical imaging.In Lossless compression the digital image is represented with reduced amounts of data. The basic principle of the reduction process is elimination of the redundant data.

In many organizations related to the field of astronomy, where most of the information comes in the form of digital images, it is not feasible to store all the information as is. Similar in medical databases where records of the patients with their respective scans are stored, it is not feasible to use lossy compression techniques to reduce the size of images because of loss of vital information when these images are decompressed for this purpose lossless image compression techniques are used.

The technique which is improved in this thesis is based on the research of Burrows and Wheeler in 1994 [1] although this technique was designed for text compression, but many variants of this technique can be found which were then modified for image compression.

Fenwick [3] proposed improvements to increase efficiency of Burrows-Wheeler Transform (BWT) based compression and to reduce the complexity of overall scheme. Michael Schindler proposed a variant of BWT which reduces the time taken by BWT by using limited sorting method[4]. Sadakane introduced an idea of improving the speed of BWT in [5]. Many improvements on Post-BWT stages are proposed by J. Abel in [2].Before discussing the technique which is proposed in this thesis, fundamental of the Image compression is discussed very briefly.

1.2 Digital Image and Digital Image Processing

Digital images are formed when light illuminated on an object is reflected back and is captured by the camera light sensors which convert the amount of light on the sensing area to finite and quantized electrical voltages and the resulting voltage arrayform digital image.

In the process of image generation, the values of the image are directly proportional to the energy radiation reflected from the target object.

$$0 < f(x, y) < \infty.$$

The values of the function *f* (x, y) can be characterized by two main components

i. The amount of source luminosity on the object at the time the object is viewed by the camera.
ii. The amount of reflected light from object, which depends upon mainly the reflectance of the object and the color of that object.

The above two components are usually referred to as *Illumination* and *Reflectance* component and can be denoted by *i*(x, y) and *r*(x, y) and the combinational result of the two of them is the function value of *f*(x, y) that is,

$$f(x, y) = i(x, y)r(x, y)$$

Where

$$0 < i(x, y) < \infty$$

And

$$0 < r(x, y) < 1.$$

Luminance on the surface of the object from the source can vary from zero to infinity, however the amount of light reflected by the object can vary from zero to one, this reflectance of the object can be referred to as multiplying co-efficient which determines the pixel values, the resultant of both factors is the amplitude value of the function *f*(x, y).

Figure 1.1 illustrates the process of image acquisition, where the reflectance and luminance of the light are also shown, the digital image formed by the image acquisition process produce the amplitude values for the function which are called as pixels, the pixels describe the gray-levels and can be denoted as,

$$\ell = f(x_0, y_0)$$

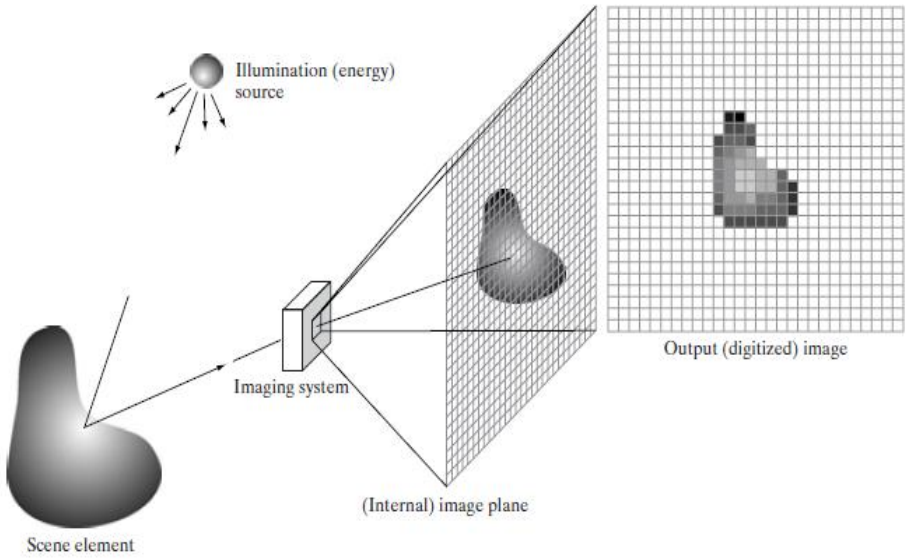
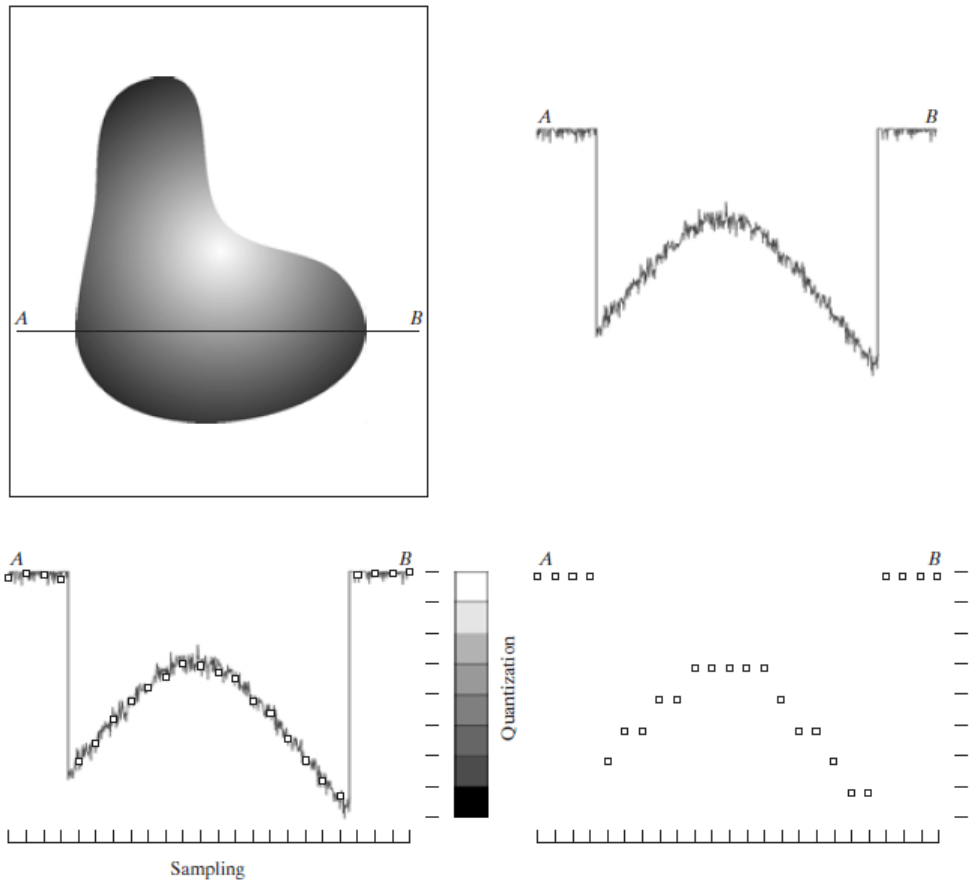


Figure 1.1: Process of image acquisition

The output of the sensor arrays is usually the voltage waveform which reflects the input of the sensors and is continuous in nature. To create the digital image from the acquired continuous data two processes are done which are sampling and quantization.

The main principle of the sampling is to convert the continuous signal into discrete intervals of time. For this purposethe output of the sensor arraysis multiplied with train of impulses and the output of the sampling is discrete in nature, but the amplitude of the signal or image is still in continuous domain.

For quantization the amplitude values of the image the sampled signal is passed through the quantization stage which maps the continuous output of the each sampled value to some finite values, the process of quantization is not reversible because no information is kept that what changes are made upon the amplitudes.



1.3 Image Compression

1.3.1 Image and Data Compression fundamentals

Lossless compression of data is the removal of redundancies in the data while preserving the ability to return the data to its original form. Lossy compression involves the discarding of less useful information from the data which then cannot be recovered

The amount of data compression achieved is measured by the term compression ratio, which is,

$$C_R = \frac{n_1}{n_2}.$$

Where n_1 represent the number of bits (or information carrying units) required for representing in original data and n_2 represent the number of bits in compressed data.

And in real life examples the compression ratio of the data depends upon the redundancy of the original data, the more the redundancy the more the compression ratio and the redundancy is the feature which we are targeting to exploit and accomplish the additional data compression, the Redundancy of the data is defined as,

$$R_D = 1 - \frac{1}{C_R} \qquad \text{(Equation: 1.7)}$$

Redundancy of the data and the compression ratio both are related terms, because in the data set when there is more redundancy then it means that data can be compressed more so that compression ratio is also increased.

In images there are coding redundancies as well as spatial and temporal redundancies which can be exploited to compress the images.

1.3.2 Coding Redundancy

A code can be defined as the set of symbols (for digital information symbols are bits) by which the information can be represented. The coding redundancy can be described as in any

image the probability of occurrence of some gray-levels is more than others.Then the amount of data symbols assigned to that high probability gray level can be reduced, so that the fewer symbols are used. For example in a gray scale digital image of 512x512 consider that gray-levels are in range of 0 – 255 and each gray-level is expressed by the single byte in memory (which is 8-Bits).

Variable length coding exploits the probabilities of the gray-levels and the lowest symbol code are assigned to the gray-level having maximum probability as in *Huffman Coding*. So simply by varying the amount of bits to represent the pixels the overall data can be compressed.

1.3.3 Spatial and Temporal Redundancy

In most images the pixels are correlated spatially and pixel intensities can be predicted reasonably well from neighboring intensities,in images the group of gray-levels can occur in between the image in many locations. That type of redundancy is known as Interpixel redundancy which indicate that some sequence of pixels are behaving as a group and that group can be found repeatedly. So the information carried by a single pixel is small because its value can be inferred from its neighboring pixels.

The entropy coding technique such as the Huffman coding does not take advantage of such redundancy.

There are several data compression techniques which take account of spatial redundancy, such as *lossless predictive coding* and *Lempel-Ziv and Lempel Ziv Welch (LZW)*[7]

1.4 Summary

In this chapter brief introduction regarding fundamentals of image compression are described and the idea of lossless image compression is also mentioned briefly, the basic idea of image data and other types of data are also briefly expressed. The fundamental ideas of compression are discussed and on what factors the compression ratios increase were also illustrated.

Chapter 2

Literature Review

2.1 Burrow wheeler transform.

M. Burrows and D. J. Wheeler in their publication gave the idea of lossless data compression[1]. The Burrows Wheeler Compression Algorithm is based upon burrow wheeler transform which is reversible sorting transform.

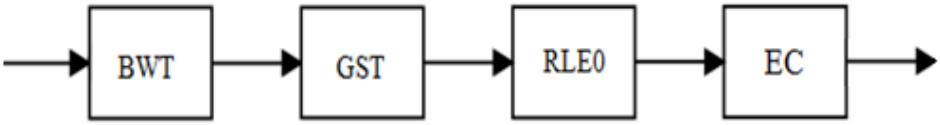


Figure 2.1 Different stages in BWCA

Burrow wheeler transform is lexographical transform which don't sort the data exactly in ascending or descending order exactly, but it sorts in such a way that data contain more runs of similar data after transformation. And this can be exploited by GST to increase the compression ratio.

A	S	S	A	S	S	I	N	A	T	I	O	N
N	A	S	S	A	S	S	I	N	A	T	I	O
O	N	A	S	S	A	S	S	I	N	A	T	I
I	O	N	A	S	S	A	S	S	I	N	A	T
T	I	O	N	A	S	S	A	S	S	I	N	A
A	T	I	O	N	A	S	S	A	S	S	I	N
N	A	T	I	O	N	A	S	S	A	S	S	I
I	N	A	T	I	O	N	A	S	S	A	S	S
S	I	N	A	T	I	O	N	A	S	S	A	S
S	S	I	N	A	T	I	O	N	A	S	S	A
A	S	S	I	N	A	T	I	O	N	A	S	S
S	A	S	S	I	N	A	T	I	O	N	A	S
S	S	A	S	S	I	N	A	T	I	O	N	A

A	S	S	A	S	S	I	N	A	T	I	O	N
A	S	S	I	N	A	T	I	O	N	A	S	S
A	T	I	O	N	A	S	S	A	S	S	I	N
I	N	A	T	I	O	N	A	S	S	A	S	S
I	O	N	A	S	S	A	S	S	I	N	A	T
N	A	S	S	A	S	S	I	N	A	T	I	O
N	A	T	I	O	N	A	S	S	A	S	S	I
O	N	A	S	S	A	S	S	I	N	A	T	I
S	A	S	S	I	N	A	T	I	O	N	A	S
S	I	N	A	T	I	O	N	A	S	S	A	S
S	S	A	S	S	I	N	A	T	I	O	N	A
S	S	I	N	A	T	I	O	N	A	S	S	A
T	I	O	N	A	S	S	A	S	S	I	N	A

Table 2.1 BWT performed on string "ASSASSINATION"

N	N
S	S
N	N
S	S
T	T
O	O
I	I
I	I
S	S
S	S
A	A
A	A
A	A

Table 2.2 "ASSASSINATION after transformation by BWT"

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

A	B	A	N	A	N	N
A	N	A	B	A	N	N
A	N	A	N	A	B	B
B	A	N	A	N	A	A
N	A	B	A	N	A	A
N	A	N	A	B	A	A

Table 2.3 BWT of string "BANANA"

In Table 2.1"ASSASSINATION" and in Table 2.3"BANANA" both strings are transformed according to burrow wheeler transform. the first step in BWT is to circularly shift the string by one step and form a N*N block of circularly shifted strings.

In case of string "ASSASSINATION" n=13 and for "BANANA" n=6.Lets call the circularly shifted matrix for "ASSASSINATION" and "BANANA" M and N respectively.

The next step is to sort the rows in M and N alphabetically as shown in table 2.1 and 2.3 respectively and call these sorted matrices M_sorted and N_sorted respectively.

Now the last column of M_sorted and N_sorted contains the Burrow wheeler transform for respective string.

The transformed string for “ASSASSINATION” is “NSNSTOIISSAAA” and for “BANANA” it is “NNBAAA”. It can be clearly seen that the transformed strings contain more repetition of words than original strings. This can be used by GST stage and RLE stage to compress the data.

2.2 Inverse Burrows wheeler Transform

Inverse Burrow wheeler only requires the transformed string and the index of original string in sorted N_sorted and M_sorted matrix respectively. So in essence for every block of data to transform only one byte for determining the index is required.

The index of original string in M_sorted and N_sorted is “1” and “4” respectively.

So now we have two things index and transformed string,

Step 1: The transformed string contains all the characters that are present in the original string so if we sort the transformed string we will get the first column of M_sorted and N_sorted matrices because the first column contains the sorted version of original string as shown in Table 2.4.

N	A
S	A
N	A
S	I
T	I
O	N
I	N
I	O
S	S
S	S
A	S
A	S
A	T

Table 2.4 transformed string (1) and sorted string (2)

Step 2: Each row of the M_sorted and N_sorted matrix contains the circularly shifted version of the original string. And in each row the last element and the first element appear consecutively in original string. so elements in string (2) appear after string (1) of Table 2.4 (“NA”,”SA”,” NA”, “SI”, “TI”, “ON”, …)

N	A
S	A
N	A
S	I
T	I
O	N
I	N
I	O
S	S
S	S
A	S
A	S
A	T

Table 2.5

Step3: Now sort the above matrix as shown in table 2.5.

A	S
A	S
A	T
I	N
I	O
N	A
N	A
O	N
S	A
S	I
S	S
S	S
T	I

Table 2.6

Step4: Now add the transformed string (1) to the beginning of above table 2.6 and repeat step 3 and 4

A	S	S	A	S	S	A
A	S	S	A	S	S	I
A	T	I	A	T	I	O
I	N	A	I	N	A	T
I	O	N	I	O	N	A
N	A	S	N	A	S	S
N	A	T	N	A	T	I
O	N	A	O	N	A	S
S	A	S	S	A	S	S
S	I	N	S	I	N	A
S	S	A	S	S	A	S
S	S	I	S	S	I	N
T	I	O	T	I	O	N

Adding transformed string (1) and sortingAdding transformed string (1) and sorting

A	S	S	A	S	A	S	S	A	S	S
A	S	S	I	N	A	S	S	I	N	A
A	T	I	O	N	A	T	I	O	N	A
I	N	A	T	I	I	N	A	T	I	O
I	O	N	A	S	I	O	N	A	S	S
N	A	S	S	A	N	A	S	S	A	S
N	A	T	I	O	N	A	T	I	O	N
O	N	A	S	S	O	N	A	S	S	A
S	A	S	S	I	S	A	S	S	I	N
S	I	N	A	T	S	I	N	A	T	I
S	S	A	S	S	S	S	A	S	S	I
S	S	I	N	A	S	S	I	N	A	T
T	I	O	N	A	T	I	O	N	A	S

Adding transformed string (1) and sortingAdding transformed string (1) and sorting

A	S	S	A	S	S	I
A	S	S	I	N	A	T
A	T	I	O	N	A	S
I	N	A	T	I	O	N
I	O	N	A	S	S	A
N	A	S	S	A	S	S
N	A	T	I	O	N	A
O	N	A	S	S	A	S
S	A	S	S	I	N	A
S	I	N	A	T	I	O
S	S	A	S	S	I	N
S	S	I	N	A	T	I
T	I	O	N	A	S	S

Adding transformed string (1) and sorting

A	S	S	A	S	S	I	N
A	S	S	I	N	A	T	I
A	T	I	O	N	A	S	S
I	N	A	T	I	O	N	A
I	O	N	A	S	S	A	S
N	A	S	S	A	S	S	I
N	A	T	I	O	N	A	S
O	N	A	S	S	A	S	S
S	A	S	S	I	N	A	T
S	I	N	A	T	I	O	N
S	S	A	S	S	I	N	A
S	S	I	N	A	T	I	O
T	I	O	N	A	S	S	A

Adding transformed string (1) and sorting

A	S	S	A	S	S	I	N	A
A	S	S	I	N	A	T	I	O
A	T	I	O	N	A	S	S	A
I	N	A	T	I	O	N	A	S
I	O	N	A	S	S	A	S	S
N	A	S	S	A	S	S	I	N
N	A	T	I	O	N	A	S	S
O	N	A	S	S	A	S	S	I
S	A	S	S	I	N	A	T	I
S	I	N	A	T	I	O	N	A
S	S	A	S	S	I	N	A	T
S	S	I	N	A	T	I	O	N
T	I	O	N	A	S	S	A	S

Adding transformed string (1) and sorting

A	S	S	A	S	S	I	N	A	T
A	S	S	I	N	A	T	I	O	N
A	T	I	O	N	A	S	S	A	S
I	N	A	T	I	O	N	A	S	S
I	O	N	A	S	S	A	S	S	I
N	A	S	S	A	S	S	I	N	A
N	A	T	I	O	N	A	S	S	A
O	N	A	S	S	A	S	S	I	N
S	A	S	S	I	N	A	T	I	O
S	I	N	A	T	I	O	N	A	S
S	S	A	S	S	I	N	A	T	I
S	S	I	N	A	T	I	O	N	A
T	I	O	N	A	S	S	A	S	S

Adding transformed string (1) and sorting

A	S	S	A	S	S	I	N	A	T	I
A	S	S	I	N	A	T	I	O	N	A
A	T	I	O	N	A	S	S	A	S	S
I	N	A	T	I	O	N	A	S	S	A
I	O	N	A	S	S	A	S	S	I	N
N	A	S	S	A	S	S	I	N	A	T
N	A	T	I	O	N	A	S	S	A	S
O	N	A	S	S	A	S	S	I	N	A
S	A	S	S	I	N	A	T	I	O	N
S	I	N	A	T	I	O	N	A	S	S
S	S	A	S	S	I	N	A	T	I	O
S	S	I	N	A	T	I	O	N	A	S
T	I	O	N	A	S	S	A	S	S	I

A	S	S	A	S	S	I	N	A	T	I	O
A	S	S	I	N	A	T	I	O	N	A	S
A	T	I	O	N	A	S	S	A	S	S	I
I	N	A	T	I	O	N	A	S	S	A	S
I	O	N	A	S	S	A	S	S	I	N	A
N	A	S	S	A	S	S	I	N	A	T	I
N	A	T	I	O	N	A	S	S	A	S	S
O	N	A	S	S	A	S	S	I	N	A	T
S	A	S	S	I	N	A	T	I	O	N	A
S	I	N	A	T	I	O	N	A	S	S	A
S	S	A	S	S	I	N	A	T	I	O	N
S	S	I	N	A	T	I	O	N	A	S	S
T	I	O	N	A	S	S	A	S	S	I	N

Adding transformed string (1) and sorting

A	S	S	A	S	S	I	N	A	T	I	O	N
A	S	S	I	N	A	T	I	O	N	A	S	S
A	T	I	O	N	A	S	S	A	S	S	I	N
I	N	A	T	I	O	N	A	S	S	A	S	S
I	O	N	A	S	S	A	S	S	I	N	A	T
N	A	S	S	A	S	S	I	N	A	T	I	O
N	A	T	I	O	N	A	S	S	A	S	S	I
O	N	A	S	S	A	S	S	I	N	A	T	I
S	A	S	S	I	N	A	T	I	O	N	A	S
S	I	N	A	T	I	O	N	A	S	S	A	S
S	S	A	S	S	I	N	A	T	I	O	N	A
S	S	I	N	A	T	I	O	N	A	S	S	A
T	I	O	N	A	S	S	A	S	S	I	N	A

Table 2.7 read the original string at "0" index. decoding complete

N	A
N	A
B	A
A	B
A	N
A	N

Transformed string **Sorted string**

N	A	A	B
N	A	A	N
B	A	A	N
A	B	B	A
A	N	N	A
A	N	N	A

Adding transformed string **sorting**

N	A	B	A	B	A
N	A	N	A	N	A
B	A	N	A	N	A
A	B	A	B	A	N
A	N	A	N	A	B
A	N	A	N	A	N

Adding transformed string **sorting**

N	A	B	A	A	B	A	N
N	A	N	A	A	N	A	B
B	A	N	A	A	N	A	N
A	B	A	N	B	A	N	A
A	N	A	B	N	A	B	A
A	N	A	N	N	A	N	A

Adding transformed string **sorting**

N	A	B	A	N	A	B	A	N	A
N	A	N	A	B	A	N	A	B	A
B	A	N	A	N	A	N	A	N	A
A	B	A	N	A	B	A	N	A	N
A	N	A	B	A	N	A	B	A	N
A	N	A	N	A	N	A	N	A	B

Adding transformed string **sorting**

N	A	B	A	N	A
N	A	N	A	B	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

Adding transformed string

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

Sorting and reading the original string at index "4" (decoding process complete)

2.3 Move to front transform:

Burrows wheeler transform compresses textual data by sorting it lexographically and hence generating sequences of similar characters as shown in above examples. However the data sorted by BWT is not exactly sorted in ascending or descending order and it contain values which change abruptly, so to solve this problem move to the front was suggested by Burrow wheeler in

their publication [1].MTF transform is called Global Structure transform because it converts the local data into global context.

2.3.1 MTF encoding:

In mtf encoding first an indexed table is created for all the symbols that are present in the data in case of English text it will contain all the alphabets.

For encoding data is read character by character(in case of text) so first character is searched in the index table. The index at which the character is present is stored and then in index table that character is brought to the front or at zero index.

String	Decoded string	Index table
		a,b,c,d,e,f,g,h,I,j,k,l,m,n,o,p,….
N	13	n,a,b,c,d,e,f,g,h,I,j,k,l,m,o,p,….
N	0	n,a,b,c,d,e,f,g,h,I,j,k,l,m,o,p,….
B	2	b,n,a,c,d,e,f,g,h,I,j,k,l,m,o,p,….
A	2	a,b,n,c,d,e,f,g,h,I,j,k,l,m,o,p,….
A	0	a,b,n,c,d,e,f,g,h,I,j,k,l,m,o,p,….
A	0	a,b,n,c,d,e,f,g,h,I,j,k,l,m,o,p,….

In the above example the string ”NNBAAA” which is BWT of”BANANA” is decoded by mtf. The first character is “N” and its index (Location) in index table is “13” so we place “13” in decoded string and secondly we bring “N” to the start of Index table (at location zero). Next character is again “N” and it is now present at location ”0” so we place the”0” in decoded string. As “N” is present at zero location so there is no change in index table.

Next character is “B” and it is present at index 2 so we write 2 in decoded string and we bring the “B” to the zero index .next character is “A” and it is at index “2” so “A” is moved to front and “2” is written in decoded string. Next two characters are A and in index table it is at location “0” so “0” is written in decoded string for both “A”. it can be seen that the decoded string has converted the consecutive similar characters to zeros which can be compressed by RLE0 but another advantage of MTF transform is that it also shift the values that appear regularly in data to lower indexes and the probability of lower values will increase and due to this property when entropy encoders are used in later stage better compression ratio will be achieved.

2.3.2 MTF DECODING.

The string decoded in previous section (“NNBAAA” decoded to”13,0,2,2,0,0”) can be recovered easily. the only thing that is required for decoding is index table. And in most cases it is known(in this case it is alphabets in English).

		a,b,c,d,e,f,g,h,I,j,k,l,m,n,o,p,….
N	13	n,a,b,c,d,e,f,g,h,I,j,k,l,m,o,p,….
N	0	n,a,b,c,d,e,f,g,h,I,j,k,l,m,o,p,….
B	2	b,n,a,c,d,e,f,g,h,I,j,k,l,m,o,p,….
A	2	a,b,n,c,d,e,f,g,h,I,j,k,l,m,o,p,….
A	0	a,b,n,c,d,e,f,g,h,I,j,k,l,m,o,p,….
A	0	a,b,n,c,d,e,f,g,h,I,j,k,l,m,o,p,….

In this case the decoded string is (“13,0,2,2,0,0”), first character is read at the location “13” of index table which is”N”. So “N” is written in original string, and in index table “N” is shifted to

the zero location. The next character can be read at location “0” which is “N” again. The third character can be read at location 2 which is “B”. And “B” is shifted to location zero of index table. The forth character can be read at location 2 which is “A”. And “A” is shifted to location zero of index table. The next two characters can be read at “0” location and both are “A”.

So by only knowing the symbols table that was used in encoding the original data can be recovered.

2.4 Zero Run-Length Encoding (RLE-0)

Typical BWCA scheme consists of four basic steps for data compression in which after GST stage (which is MTF as suggested by M. Burrows and D. J. Wheeler) the *Zero Run-Length encoder* (RLE-0) is also suggested, from all the discussion of the MTF and RLE it seems at first that RLE is the extra step which is not necessary, but this is not the case. MTF is just the encoder of data and the MTF output data is not yet compressed, by analyzing the output of the MTF it is clearly visible that occurrences of zeros are very frequent as shown in Figure 2.14 in which the histogram of the output of the MTF clearly illustrates the high occurrences of zeros.

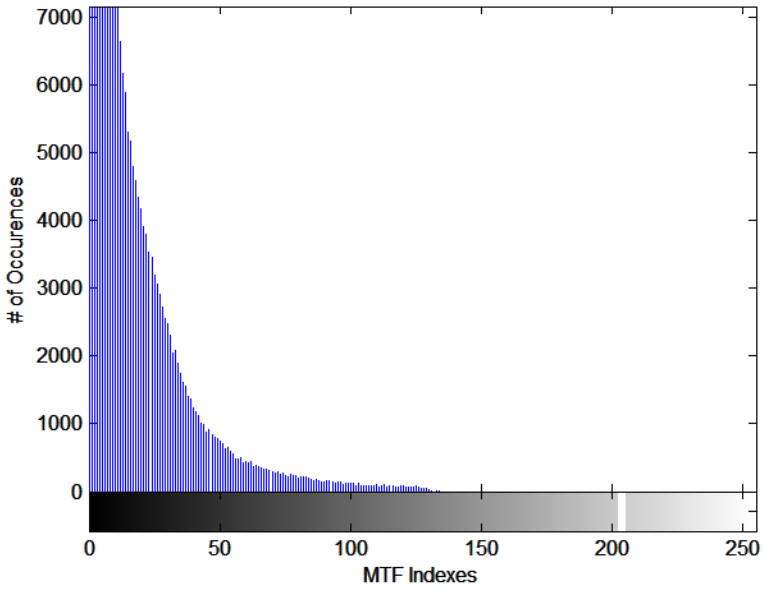


Figure 2.14: Histogram of the MTF transformed data

It is always an option to directly process the MTF output with entropy coders, but the problem with this procedure is that not only the present zeros are in high probabilities but most of the time these zeros are in stream, so one special Run-Length Encoder can be used to exploit this redundancy. Hence the RLE-0 is the run-length encoder which only transforms the zeros so that streams of the zeros can be reduced and the data can be compressed more.

2.5 Entropy Coder

Entropy coder is the final stage of BWCA, this is the stage in which the data is compressed in order to reduce its size to represent the information, as we have described all the stages of BWCA, RLE-0 is the stage in which the stream of 0s are removed, but the zeros are not the only type of redundancy in data, so an actual data compression algorithm is needed to reduce the whole data.

Entropy coders are used to eliminate the coding redundancy of data, and as described earlier coding redundancy is usually present in data, even in the binary codes which are used to represent the values of gray levels. To eliminate coding redundancy a *variable length code* can be used which can represent the data instead of original data.

2.5.1 Huffman coding

In order to remove coding redundancy Huffman coding is the most popular and widely used technique, this technique was proposed by Huffman in 1952 [8] and still is used due to the significant amount data reduction. Huffman coding works on the individual symbols of information source and yields the smallest possible code for every symbol. The codes assigned by Huffman coding are variable length codes, as the optimal fixed codes are generated for individual source symbol then the original data can be coded at a time by simple look up table approach.

The brief intro of Huffman encoding is presented here to describe the importance and working of entropy coder stage, the first step in Huffman coding is to generate the list of available symbols with respect to their probabilities, when the list of probabilities are available then the procedure of source reduction is performed, in this process the two lowest probabilities of the information source are added together and the same process is repeated attain until only two sources are remained.

The process of the source reduction is illustrated in Figure 2.15 in which the available symbols are represented in column named as symbols and the probabilities of their respective occurrences are shown next.

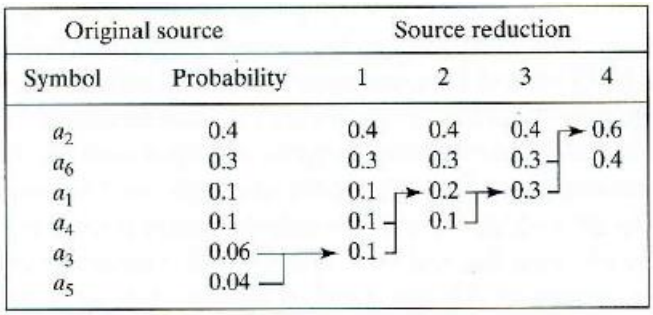


Figure 2.15: Process of source reduction in Huffman [6]

The second step of Huffman coding is to assign short codes to reduced sources, in this step the smallest codes are assigned to the final sources, for the rest of sources the appending of 0s and 1s is used, and the assignment procedure is performed on all the sources and hence all the sources have their own unique codes as shown in Figure 2.16.

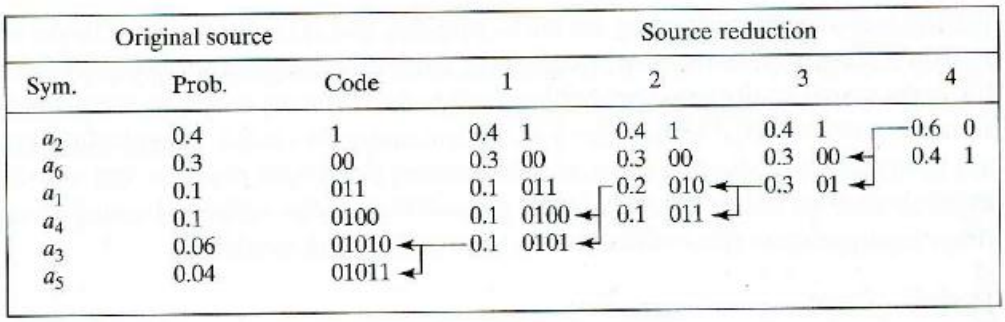


Figure 2.16: Code assignment process in Huffman [6]

Considering that for the above symbols the minimum amount of data bits required to represent all the symbols is 3-bits (since total symbols are 8, i.e. $a_0, a_1, \ldots a_7$ so $2^3=8$) and after Huffman coding the average code length can be calculated by as follows.

$$L_{avg} = (0.41)\,(1) + (0.3)\,(2) + (0.1)\,(3) + (0.1)\,(4) + (0.06)\,(5) + (0.04)\,(5)$$

$$L_{avg} = 2.2 \text{ bits/symbols}$$

Hence the average amount of information with the codes generated by Huffman is lower than original 3 bits/symbols.

2.5 Summary

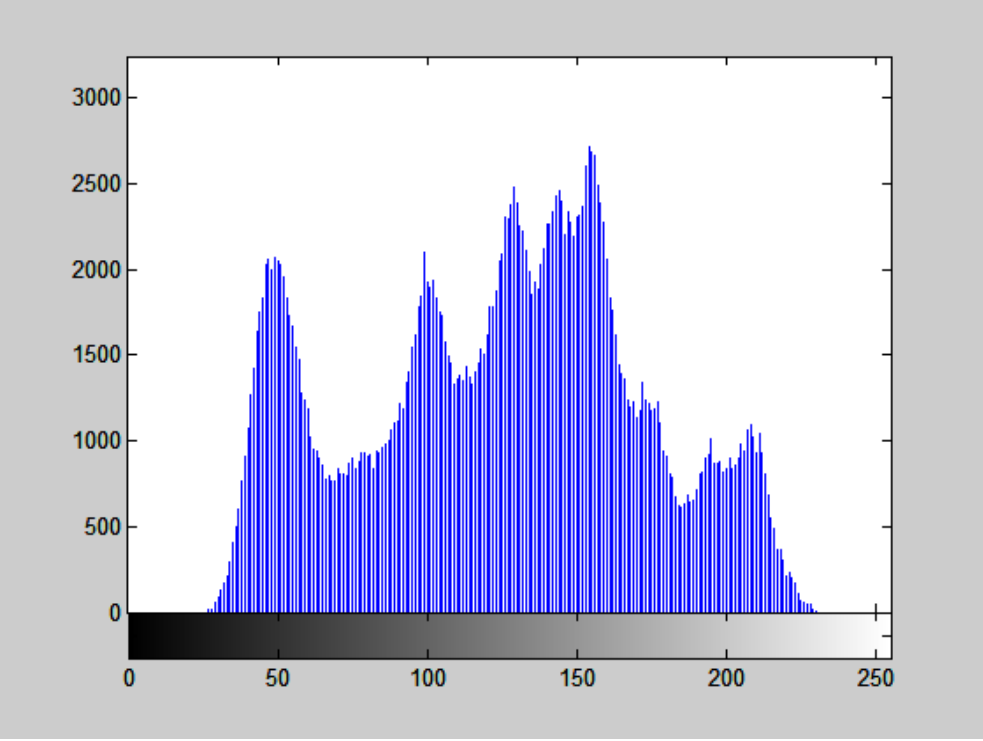
In this chapter the original scheme of BWCA is explained in detail, each stage of BWCA with adequate explanation is illustrated with short but related examples. Related works to lossless image compression techniques ware also discussed in brief detail; the other fields of BWT are also discussed momentarily

Proposed Scheme

Chapter 3

3.1 Histogram processing.

Histogram in figure 3.1 clearly shows that the pixels are distributed in wide range of intensity values. If one can redistribute the histogram to make the intensity values lie in smaller range then the further stages of burrow wheeler compression algorithm will be able to compress the image to higher compression ratios. There are several ways to shift the histogram of image in a specific range but for lossless image compression that method has to be reversible so that when we decode the compressed image it can be retrieved without losing any information.

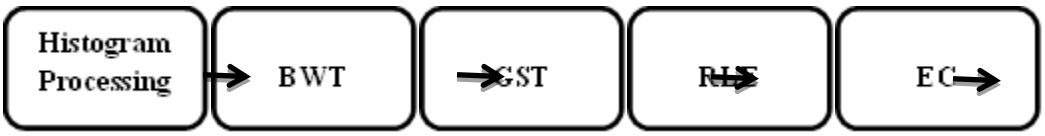


Fgure 3.1. Histogram of image 16.

This can be achieved by dividing the image into small blocks and finding the minimum value in each block. The minimum value for each block must be stored because it will be required for

decompression. The minimum value for each block is then subtracted from every value of that block respectively.

What this whole procedure does is that if the block size is quite small then the intensity value in that block will be similar, and by subtracting the minimum value from them the values will be shifted to the lower intensity values. Further if two blocks contain different value one contain smaller value and other contain very large values, after finding local minima and subtracting that from respective block, both blocks will then contain similar values. Thus this procedure shifts the histogram of image to lower intensity values. And by storing only the minimum values original image can be recovered without any loss of information.



16x16 block of original image.

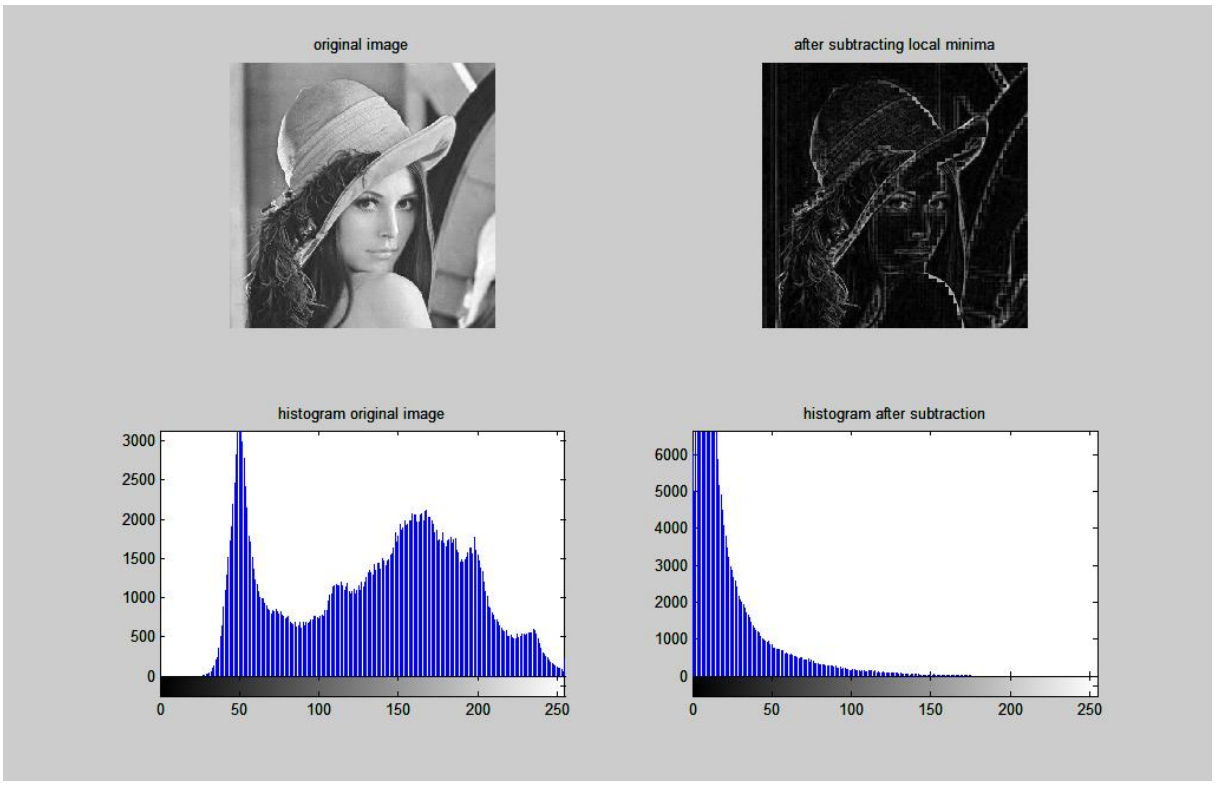
225	223	226	223	225	227	229	231	231	232	230	233	234	237	233	231
225	223	226	223	225	227	229	231	231	232	230	233	234	237	233	231
225	223	226	223	225	227	229	231	231	232	230	233	234	237	233	231
225	223	226	223	225	227	229	231	231	232	230	233	234	237	233	231
225	223	226	223	225	227	229	231	231	232	230	233	234	237	233	231
222	223	220	220	221	226	231	229	231	230	233	233	235	234	238	237
223	223	224	224	224	231	230	233	230	232	234	238	235	235	237	237
223	223	221	221	226	231	225	231	236	233	235	235	236	236	236	235
223	221	224	225	229	227	231	232	230	235	236	236	238	236	235	237
221	222	226	228	230	230	229	233	233	233	236	237	237	234	235	236
225	222	225	227	227	230	232	232	235	234	236	236	235	237	234	238
225	224	223	227	231	232	232	233	232	235	233	235	232	238	236	238
226	225	227	227	232	232	233	235	232	234	234	235	236	233	235	236
227	230	230	229	233	233	229	231	230	229	230	234	234	234	236	232
229	229	228	229	232	233	233	230	233	229	233	232	234	231	233	232
228	230	228	231	228	232	233	232	232	231	233	235	234	236	232	237

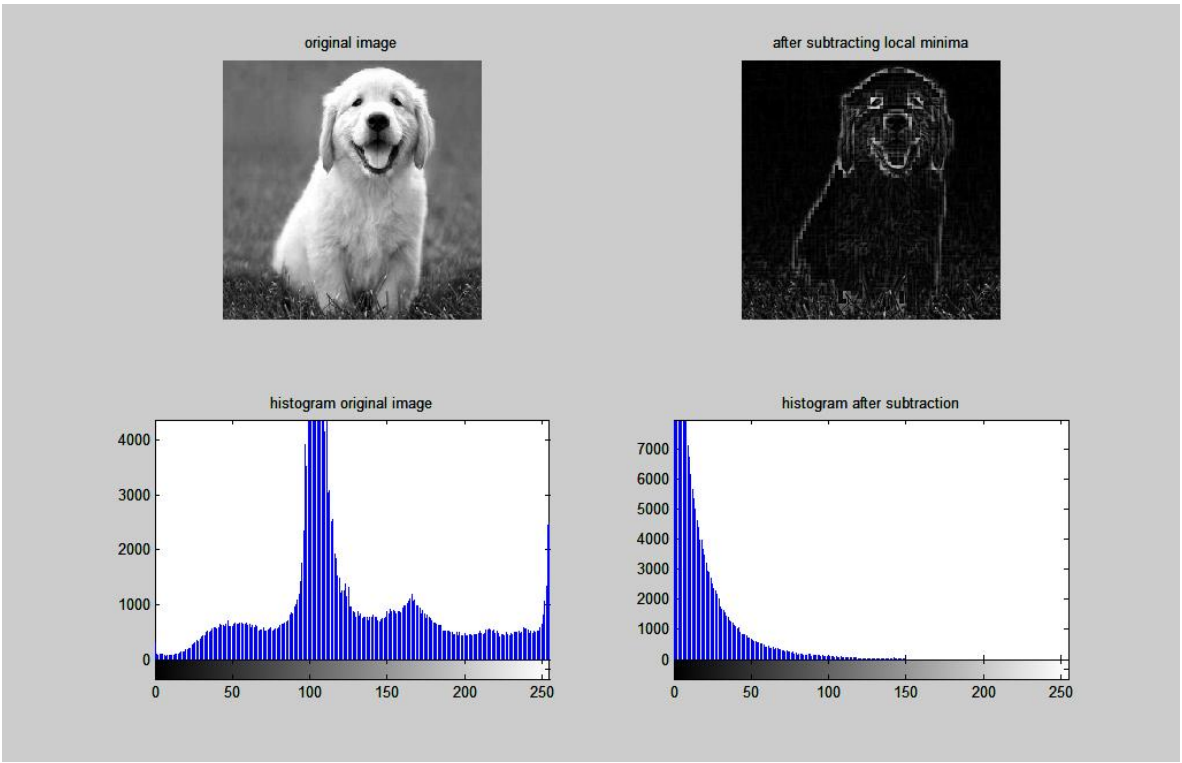
Minimum values

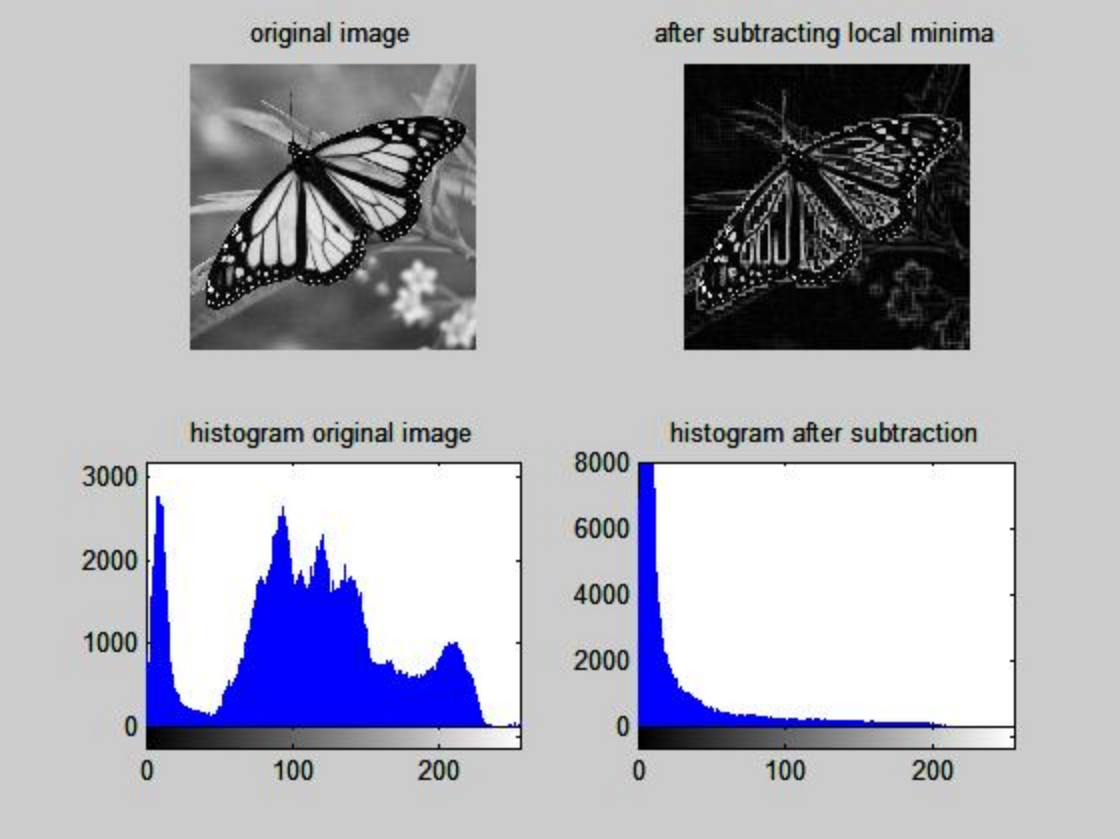
220	230
221	229

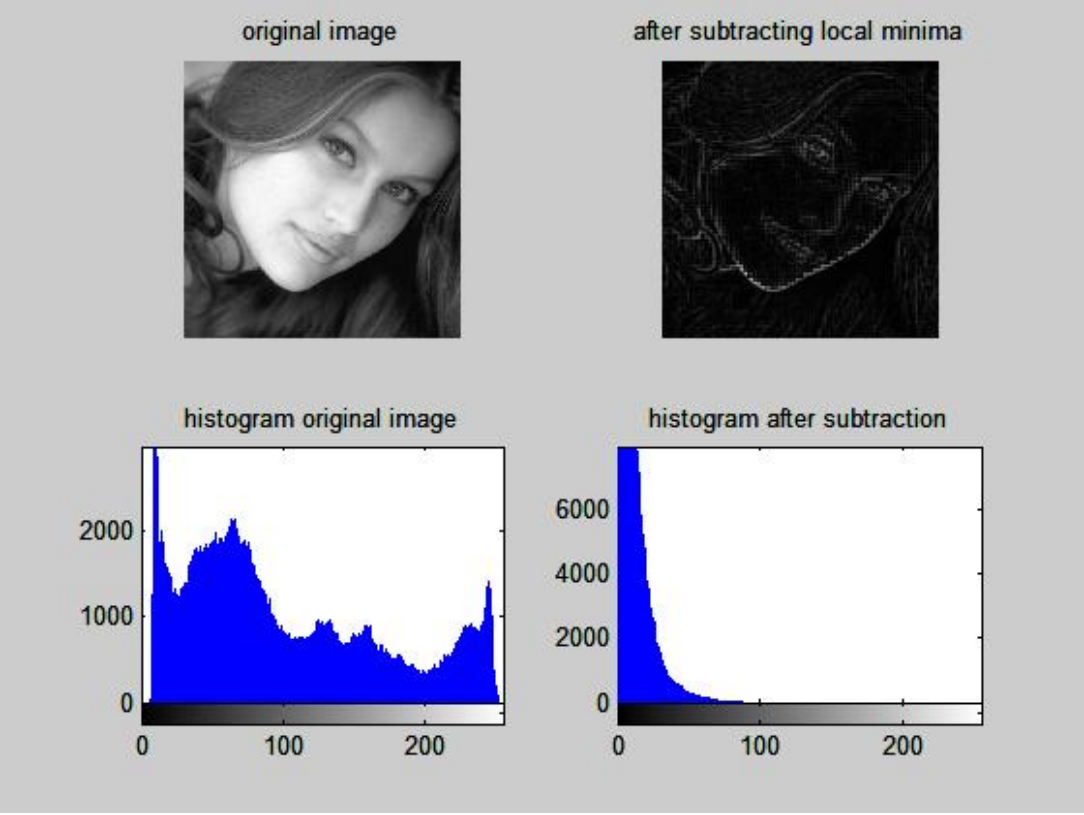
After subtracting minimum values from the image

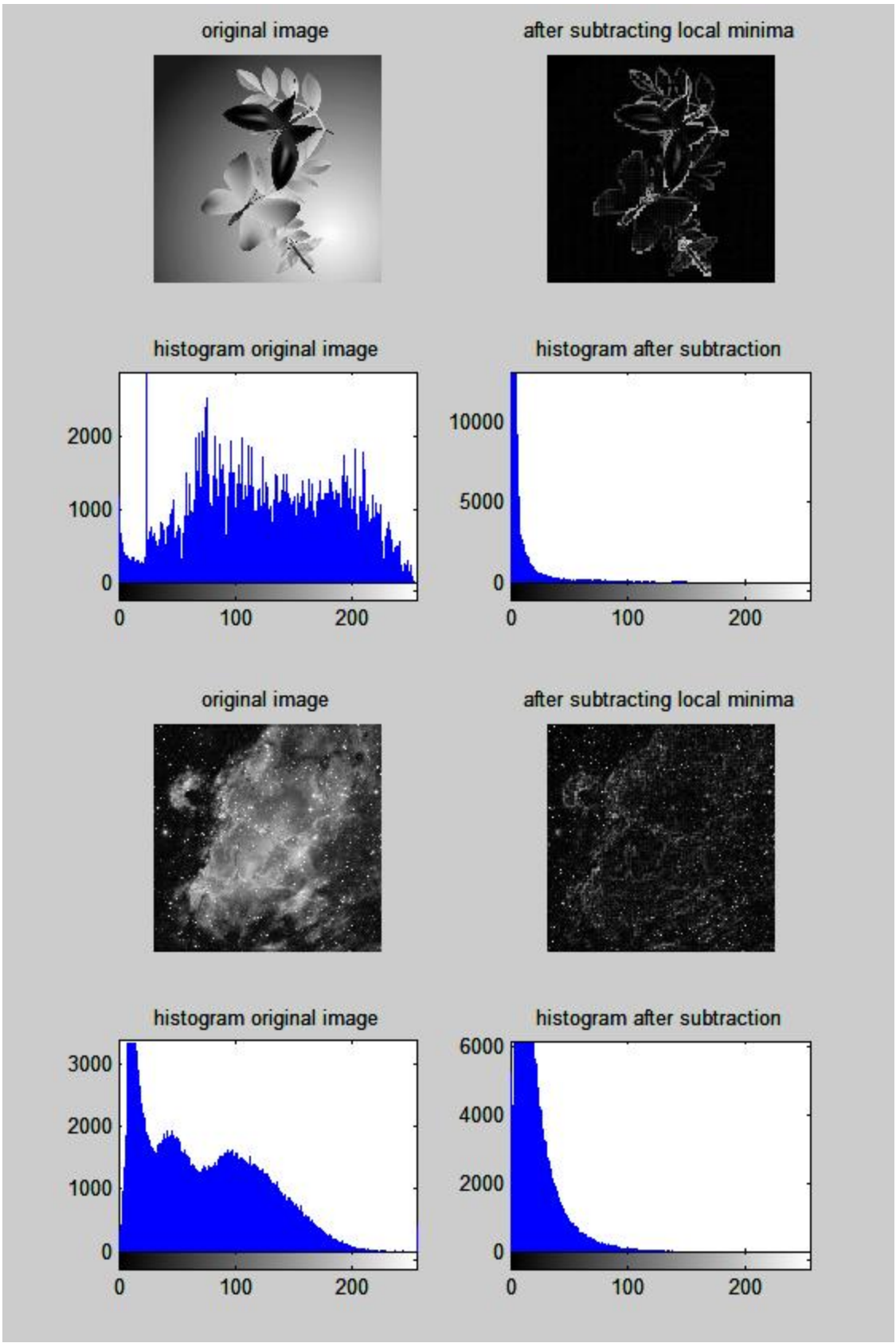
5	3	6	3	5	7	9	11	1	2	0	3	4	7	3	1
5	3	6	3	5	7	9	11	1	2	0	3	4	7	3	1
5	3	6	3	5	7	9	11	1	2	0	3	4	7	3	1
5	3	6	3	5	7	9	11	1	2	0	3	4	7	3	1
5	3	6	3	5	7	9	11	1	2	0	3	4	7	3	1
2	3	0	0	1	6	11	9	1	0	3	3	5	4	8	7
3	3	4	4	4	11	10	13	0	2	4	8	5	5	7	7
3	3	1	1	6	11	5	11	6	3	5	5	6	6	6	5
2	0	3	4	8	6	10	11	1	6	7	7	9	7	6	8
0	1	5	7	9	9	8	12	4	4	7	8	8	5	6	7
4	1	4	6	6	9	11	11	6	5	7	7	6	8	5	9
4	3	2	6	10	11	11	12	3	6	4	6	3	9	7	9
5	4	6	6	11	11	12	14	3	5	5	6	7	4	6	7
6	9	9	8	12	12	8	10	1	0	1	5	5	5	7	3
8	8	7	8	11	12	12	9	4	0	4	3	5	2	4	3
7	9	7	10	7	11	12	11	3	2	4	6	5	7	3	8


original image
after subtracting local minima
histogram original image
histogram after subtraction
3000
2500
2000
1500
1000
500
0
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250


original image
after subtracting local minima
histogram original image
histogram after subtraction
4000
3000
2000
1000
0
7000
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250


original image
after subtracting local minima
histogram original image
histogram after subtraction
3000
2000
1000
0
8000
6000
4000
2000
0
100
200

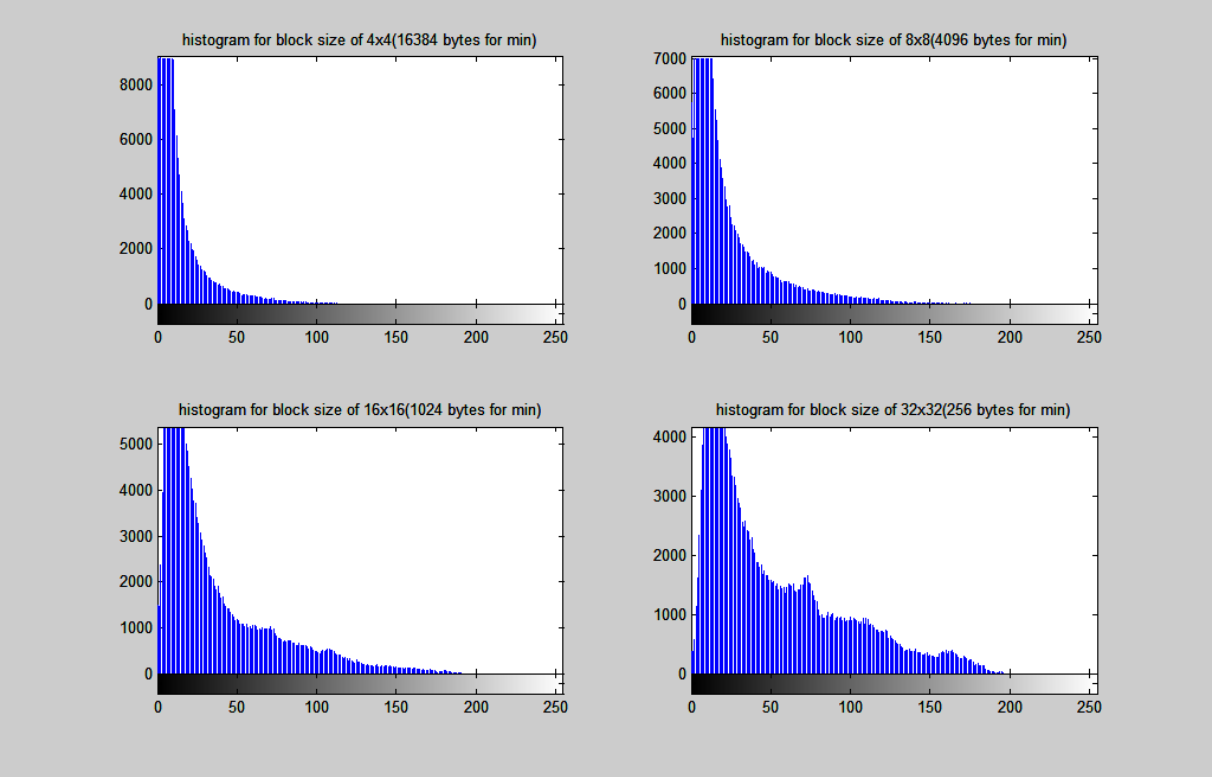

original image
after subtracting local minima
histogram original image
histogram after subtraction
2000
1000
0
6000
4000
2000
0
100
200

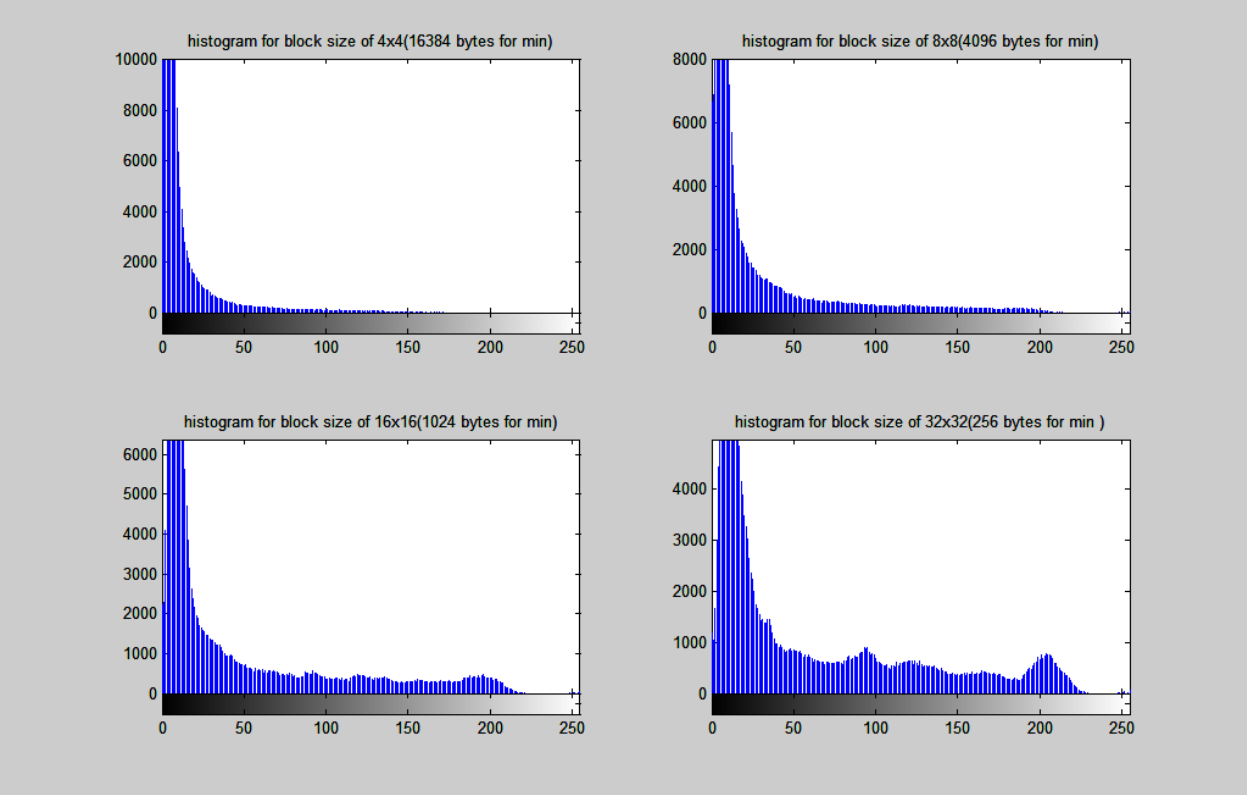

original image
after subtracting local minima
histogram original image
histogram after subtraction
2000
1000
0
10000
5000
0
100
200
original image
after subtracting local minima
histogram original image
histogram after subtraction
3000
2000
1000
0
6000
4000
2000
0
100
200

In these figure the block size for finding minimum value is taken as 8x8 and after subtraction histogram of these images is shifted to the lower intensity values, which will ultimately result in higher compression ratio. For 512x512 image with block size for local minima of 8x8 the minimum values will require 4096 bytes.

3.1.1 Size of block.

Size of block plays an important role in compression ratio. if the block is very small then though it will shift the histogram to a very narrow range as shown in figure. But additional overhead to store the minimum values will effect the compression ratio. For example for grey scale image of size 512x512 bytes if block size is of 4x4 then bytes required to store minimum values will be 16384 bytes. If block size is 32x32 then bytes required to store minimum values will be 256 bytes. So optimizing the block size can be very important in determining the compression ratio.


histogram for block size of 4x4(16384 bytes for min)
histogram for block size of 8x8(4096 bytes for min)
histogram for block size of 16x16(1024 bytes for min)
histogram for block size of 32x32(256 bytes for min)


histogram for block size of 4x4(16384 bytes for min)
histogram for block size of 8x8(4096 bytes for min)
histogram for block size of 16x16(1024 bytes for min)
histogram for block size of 32x32(256 bytes for min)

11.jpg

3.2 Improvements in MTF stage.

Burrow wheeler transform is performed on 32x32 or 64x64 blocks of image and then MTF is performed. What BWT does is that it sorts the data in roughly ascending order so after BWT each block contains relatively higher values at the end and lower values at the start as shown by highlighted row in figure below.so if MTF is performed on whole image then at the end of each block the mtf index table will contain high values and then next block will be processed from start. And that block will contain lower values in the beginning so this will result in higher values.

This problem can be solved by refreshing the index list after performing the mtf on one block of image. This will result in lower values in the index table at start of each block. This will shift the histogram of overall image to lower values further enhancing the compression ratio.

Burrows Wheeler Transform Performed on 16x16 block of lena img															
3	1	8	0	13	1	2	2	2	2	2	2	4	10	9	3
11	11	11	11	11	3	4	3	3	3	3	0	0	11	1	0
1	1	1	1	1	5	1	5	3	0	3	2	3	7	7	7
7	7	11	4	7	7	0	3	0	0	0	0	0	0	4	3
6	14	6	6	6	6	6	5	5	5	5	5	12	4	7	7
6	9	7	9	0	2	3	12	4	6	2	1	5	7	3	3
3	3	3	4	2	3	5	4	6	3	1	1	1	1	8	9
3	1	3	3	5	8	5	8	5	6	5	5	6	1	3	3
3	3	3	8	11	11	3	3	3	3	3	4	3	6	4	11
6	5	4	4	5	5	4	1	7	7	7	6	8	2	1	1
6	4	4	4	4	4	7	8	5	5	6	6	9	7	6	5
5	6	4	8	9	7	3	5	5	5	5	5	5	9	10	6
3	4	8	6	4	4	8	7	3	12	7	9	9	11	12	5
7	7	3	7	9	9	7	6	7	7	7	7	7	6	8	7
6	6	11	9	9	9	9	9	10	12	6	5	11	6	4	9
10	6	11	7	8	11	11	8	12	12	11	8	11	11	10	12

3.3 Improvements in RLE (Run Length Encoding).

In this stage of burrow wheeler compression algorithm all the consecutive values that are created due to histogram processing, BWT and MTF transforms are encoded.

4	3	9	4	14	4	6	1	1	1	1	1	7	12	12	9
13	1	1	1	1	2	5	2	1	1	1	9	1	4	8	3
2	1	1	1	1	11	2	2	5	4	2	9	2	13	1	1
1	1	7	8	3	1	6	5	2	1	1	1	1	1	4	3
13	15	2	1	1	1	1	9	1	1	1	1	15	6	8	1
5	12	3	2	9	11	10	8	8	8	5	12	10	10	8	1
1	1	1	7	6	3	5	4	7	4	7	1	1	1	15	11
4	4	2	1	7	5	2	2	2	6	2	1	2	5	5	1
1	1	1	5	13	1	3	1	1	1	1	8	2	6	3	4
3	7	4	1	2	1	2	7	10	1	1	5	8	10	5	1
4	6	1	1	1	1	6	6	7	1	5	1	10	5	3	4
1	2	6	6	6	6	10	7	1	1	1	1	1	4	14	8
5	8	8	4	3	1	3	8	5	12	3	8	1	12	4	10
5	1	6	2	6	1	2	9	2	1	1	1	1	2	8	3
3	1	8	5	1	1	1	1	10	9	5	9	6	3	10	7
7	4	5	8	9	3	1	2	9	1	3	3	2	1	6	4

Figure: After performing Histogram processing, BWT, and MTF transform.

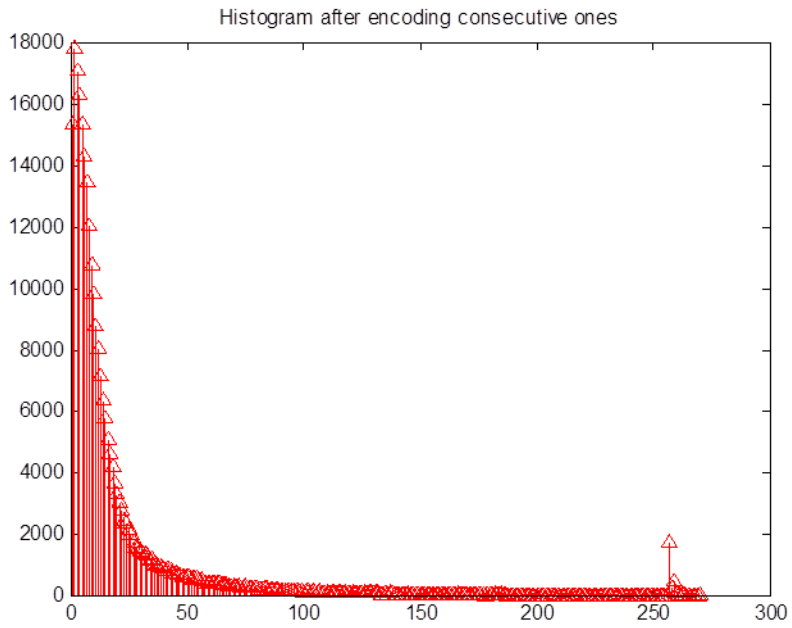
It can be clearly seen that there are consecutive runs of same characters in the sample 16x16 data block. RLE is implemented by creating new symbols for consecutive runs of data, as the image does not contain higher intensity values than 255. So for two consecutive 1's symbol 256 is assigned and for three consecutive 1's 257 is assigned and so on.

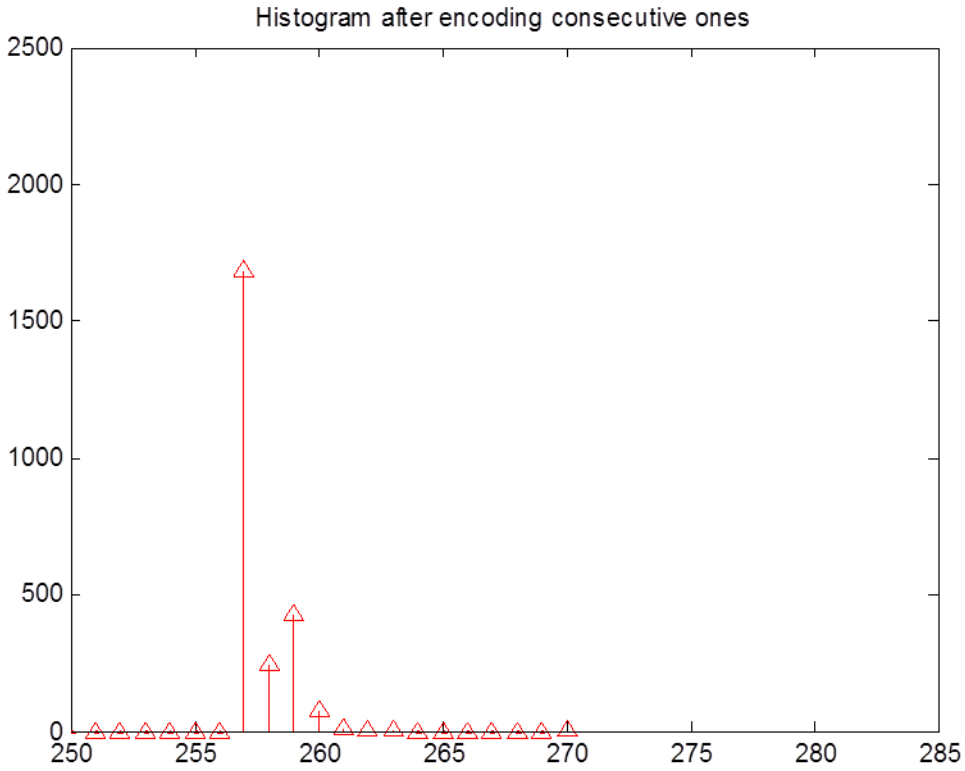
This might seem counterproductive but when this run length encoded data will be encoded by entropy coder such as Huffman coding. Huffman will generate code for each symbol according to its probability so if symbol 256 (2 consecutive 1's) has high probability then Huffman will generate small code for it.

So overall by run length encoding higher compression ratios will be achieved.

After performing Proposed run length encoding on 16x16 block of lena img															
4	3	9	4	14	4	6	260	7	12	12	9	13	259	2	5
2	258	9	1	4	8	3	2	259	11	2	2	5	4	2	9
2	13	259	7	8	3	1	6	5	2	260	4	3	13	15	2
259	9	259	15	6	8	1	5	12	3	2	9	11	10	8	8
8	5	12	10	10	8	259	7	6	3	5	4	7	4	7	258
15	11	4	4	2	1	7	5	2	2	2	6	2	1	2	5
5	259	5	13	1	3	259	8	2	6	3	4	3	7	4	1
2	1	2	7	10	257	5	8	10	5	1	4	6	259	6	6
7	1	5	1	10	5	3	4	1	2	6	6	6	6	10	7
260	4	14	8	5	8	8	4	3	1	3	8	5	12	3	8
1	12	4	10	5	1	6	2	6	1	2	9	2	259	2	8
3	3	1	8	5	259	10	9	5	9	6	3	10	7	7	4
5	8	9	3	1	2	9	1	3	3	2	1	6	4		

The histogram shown in figure clearly shows the consecutive runs of 1's shifted to 256 and above values


Histogram after encoding consecutive ones
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250
300


Histogram after encoding consecutive ones
2500
2000
1500
1000
500
0
250
255
260
265
270
275
280
285

3.4 LZW Algorithm

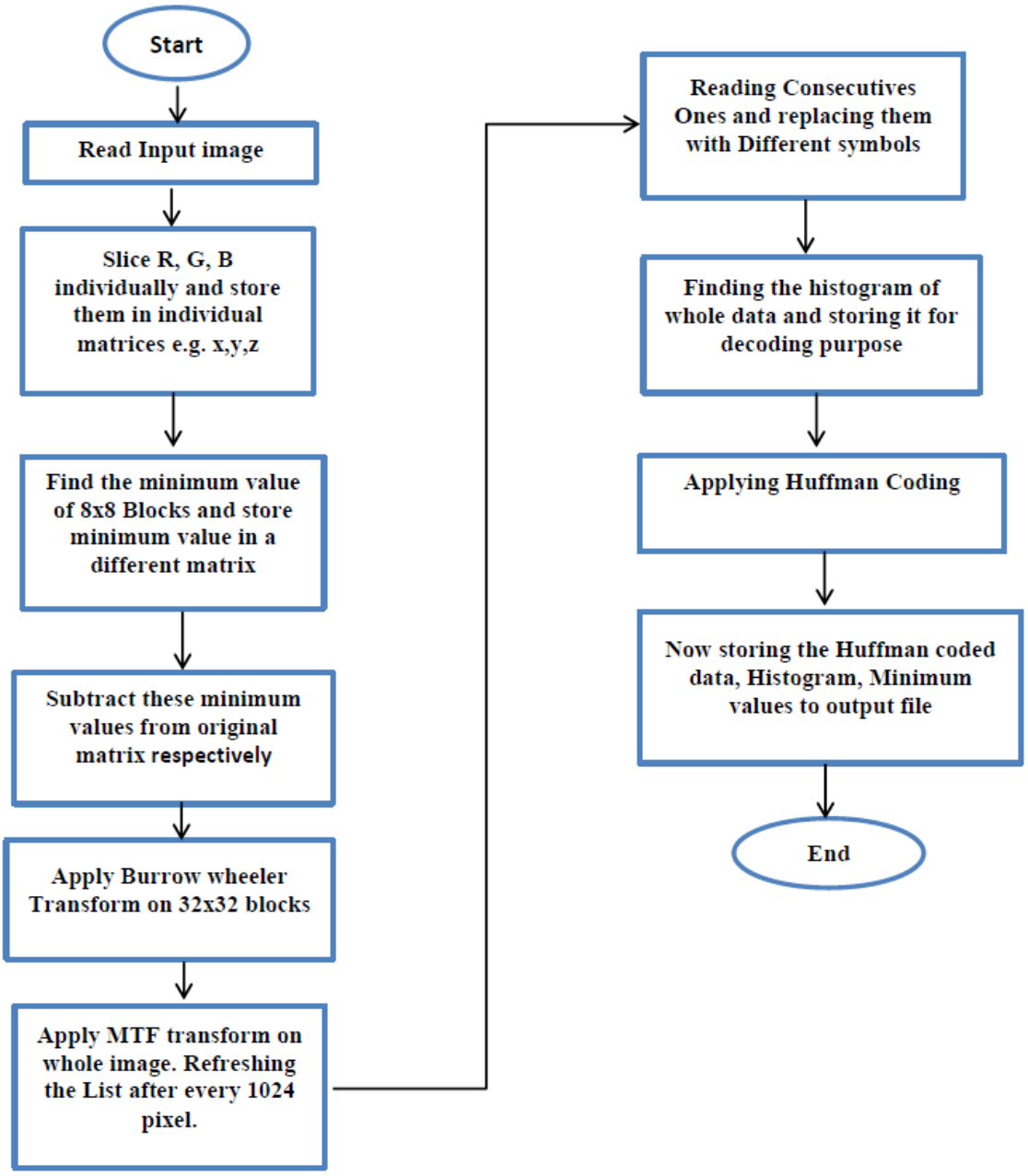
Lempel Ziv Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It is a dictionary based algorithm and strings of input are replaced by the dictionary entries. It was published as an improved implementation of the LZ78 algorithm. LZW coding assigns fixed length code word to variable length sequences of source symbols. The coding is done by creating the dictionary first which contains all the source symbols. For grey scale images the first 256 locations of dictionary are assigned values from 0 to 255.

As the input is sequentially encoded the input sequences that are not in the dictionary are placed in the dictionary at locations determined by algorithm. For example if first two pixels that are to be encoded are “125 130”, this sequence is assigned to location 257(the address following the locations reserved for intensity levels 0 to 255). Next time whenever sequence “125 130” is encounteredthe address of the location of sequence is used to represent it. If a nine bit dictionary is implemented then the two pixels found in dictionary will be replaced by a single 9 bit code word.

In LZW the size of dictionary and refreshing of dictionary if sufficient compression ratio is not achieved (if sequences are not detected in dictionary) is very important to achieve good compression.

If dictionary is too small then the matching of sequences will be less likely, but if the dictionary is too large then the size of code word will reduce compression ratio. Similarly good implementations also include mechanism that monitors the rate of detection of sequences, if it becomes low the dictionary is populated again with relevant sequences, thereby increasing the probability of finding a sequence in dictionary. The results of proposed scheme are compared with LZW algorithm

Compression Phase of Proposed Scheme



Chapter 4

4.1 Implementation and results

RESULTS

Lenna Image	Original Size	Original BWCA	Proposed histogram processed BWCA
R	262144	204340	184221
G	262144	197750	179373
B	262144	196350	178939
Total	786432	598440	542533
CR	--	1.31	1.45

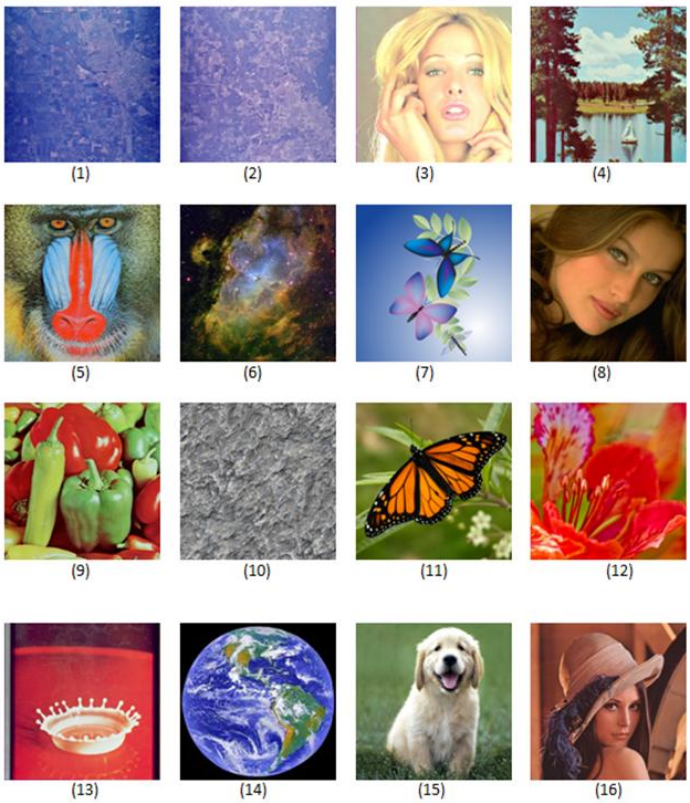
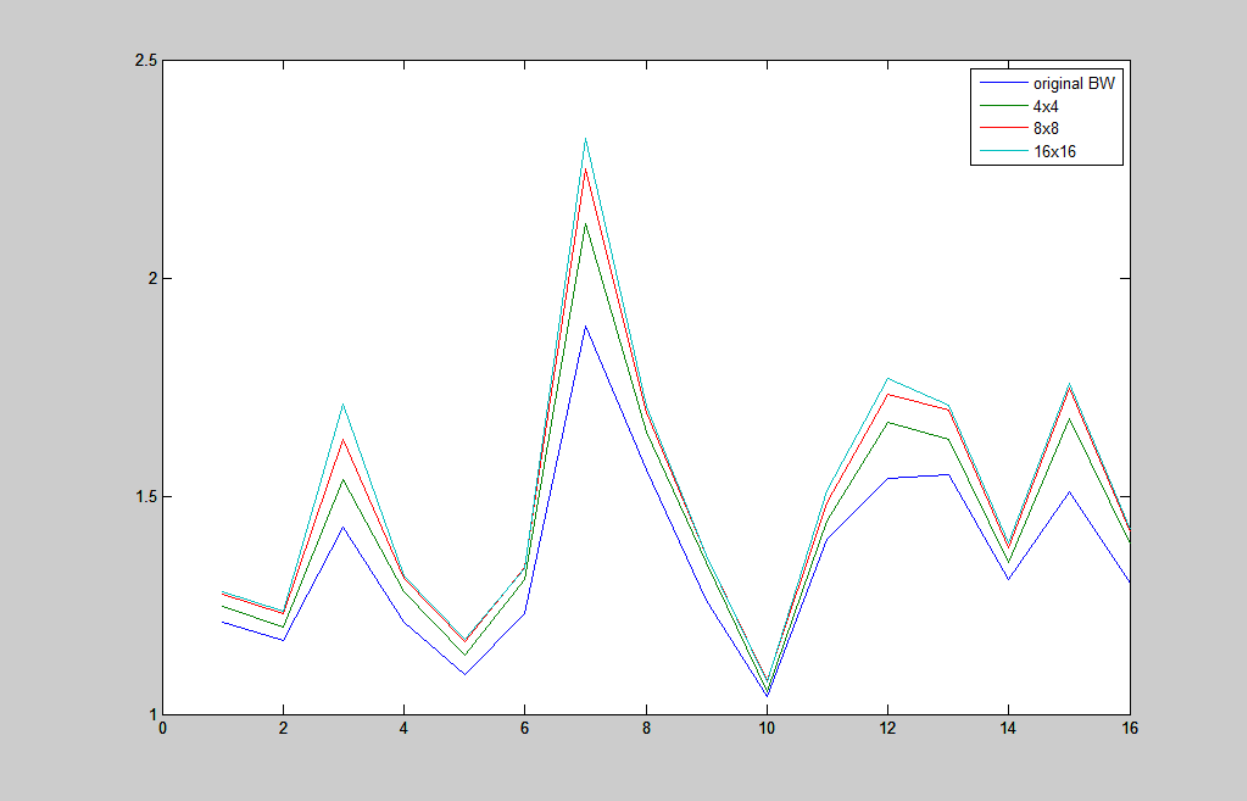


Figure 4.1: Preview of images used for compression with proposed and original scheme [9].

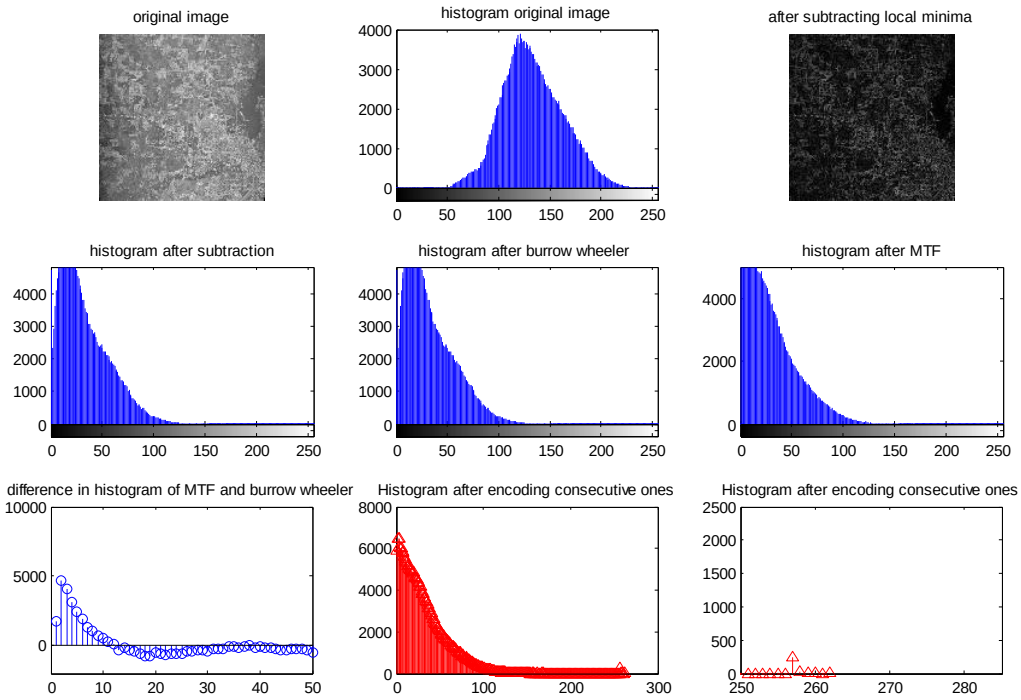
Image	effect of size of local minima on compression ratio			original
	4x4	8x8	16x16	BWCA
1	1.246063	1.275113	1.28195	1.21
2	1.198649	1.231052	1.236668	1.17
3	1.537983	1.62888	1.709544	1.43
4	1.281342	1.311647	1.317523	1.21
5	1.13488	1.167271	1.17067	1.09
6	1.308741	1.33574	1.333408	1.23
7	2.124636	2.250917	2.320456	1.89
8	1.647196	1.692367	1.705823	1.56
9	1.347384	1.363303	1.365428	1.26
10	1.051926	1.075452	1.073801	1.04
11	1.443158	1.484374	1.511931	1.4
12	1.66876	1.733534	1.770745	1.54
13	1.629997	1.695659	1.708377	1.55
14	1.349059	1.382136	1.392523	1.31
15	1.676166	1.747353	1.759576	1.51
16	1.389357	1.417134	1.42298	1.3
	1.4397	1.4869	1.50508	1.35

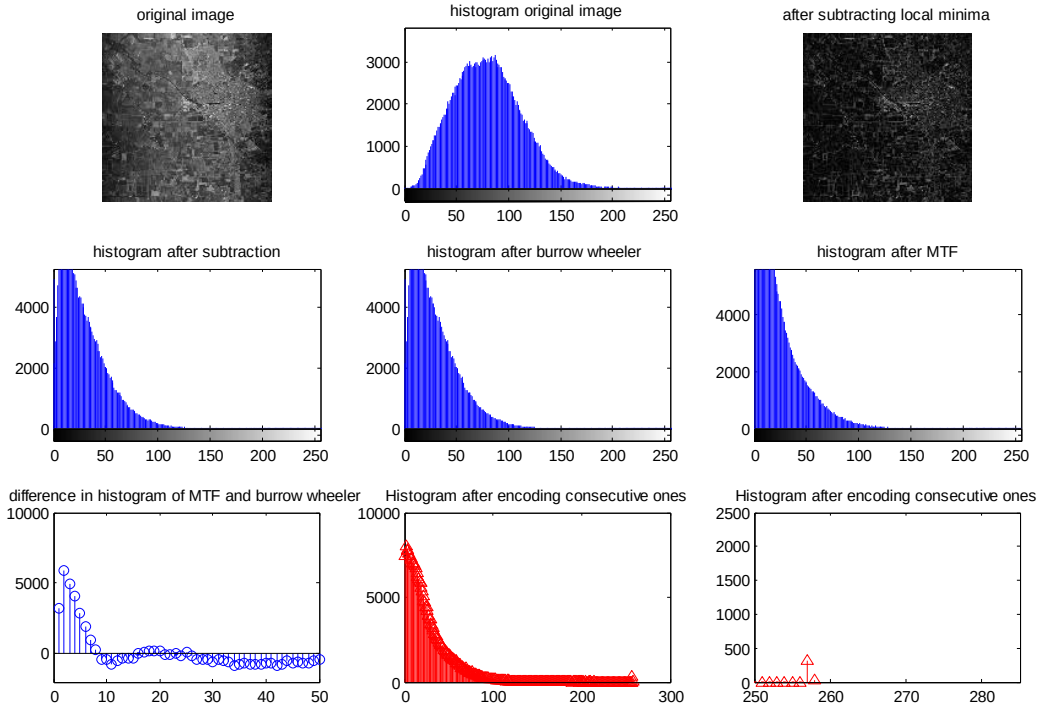
Comparison between Compression Ratio of original BWCA and proposed scheme using different size blocks for finding local minima

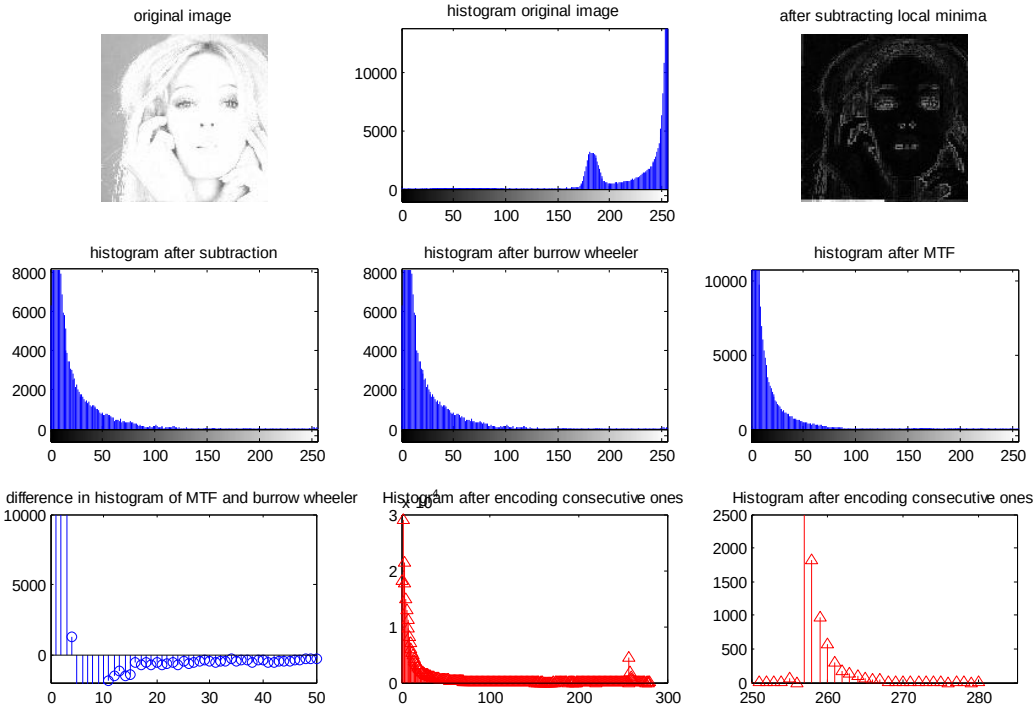


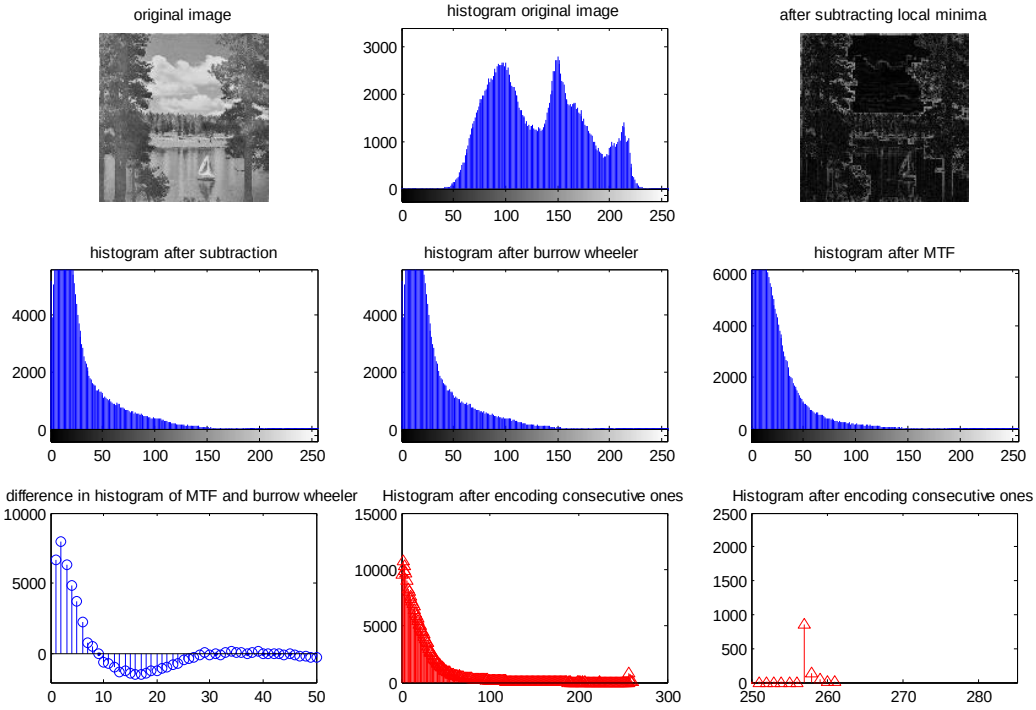
Comparison between Compression Ratio of original Burrow Wheeler Compression Algorithm and by proposed scheme using different size blocks for finding local minima

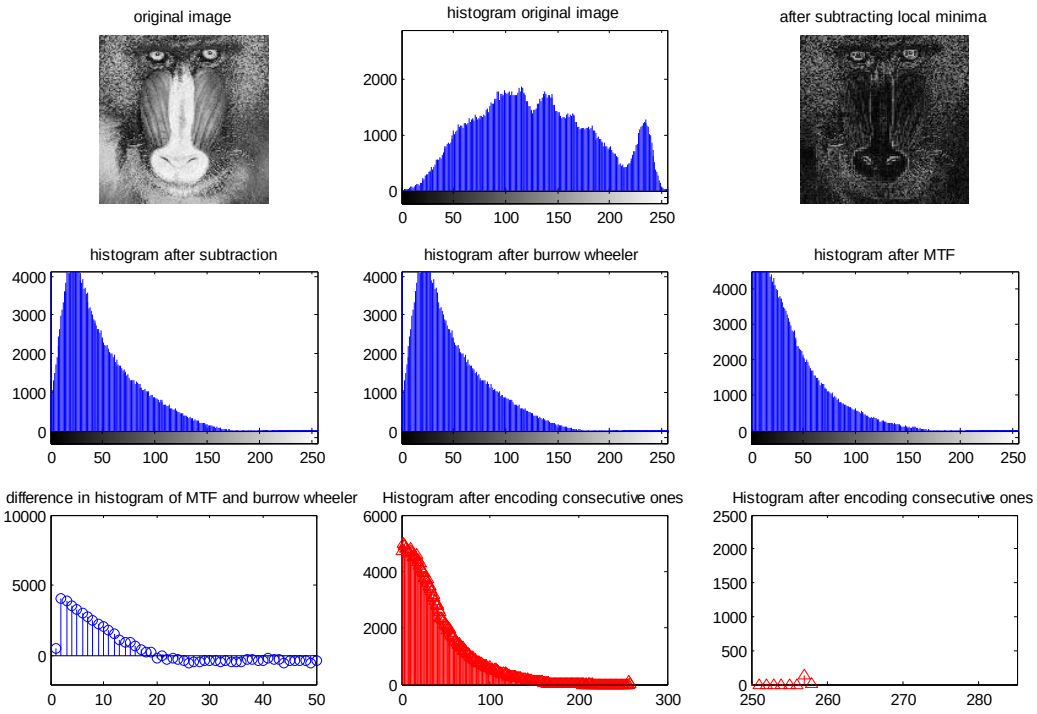
Histograms of each image after implementing the proposed scheme

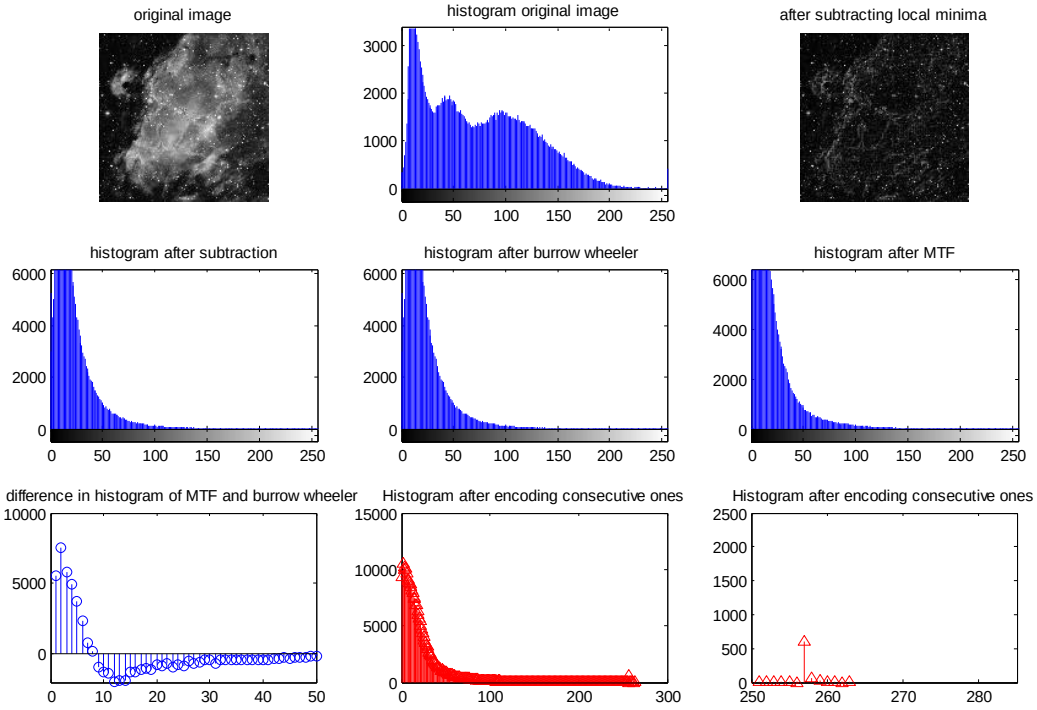


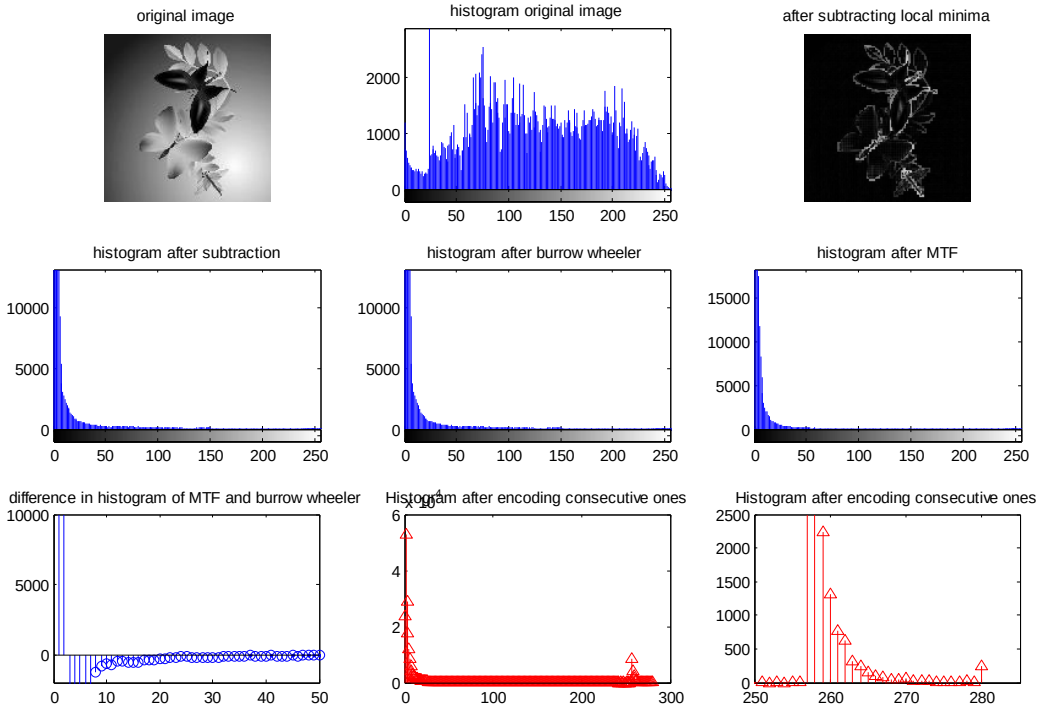


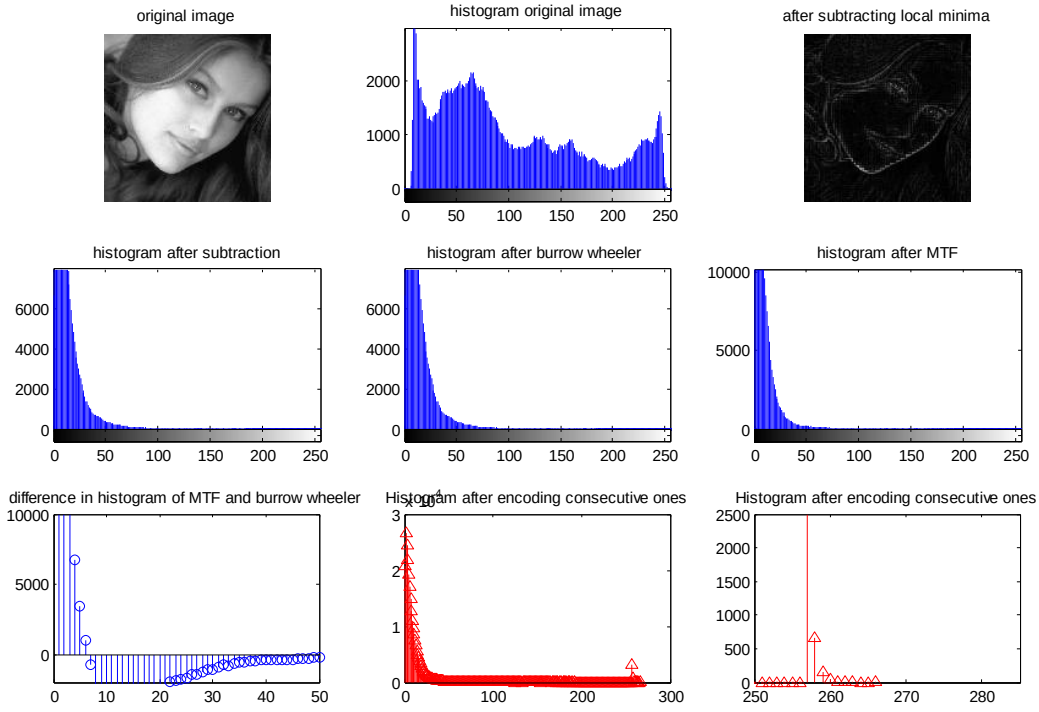

original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones

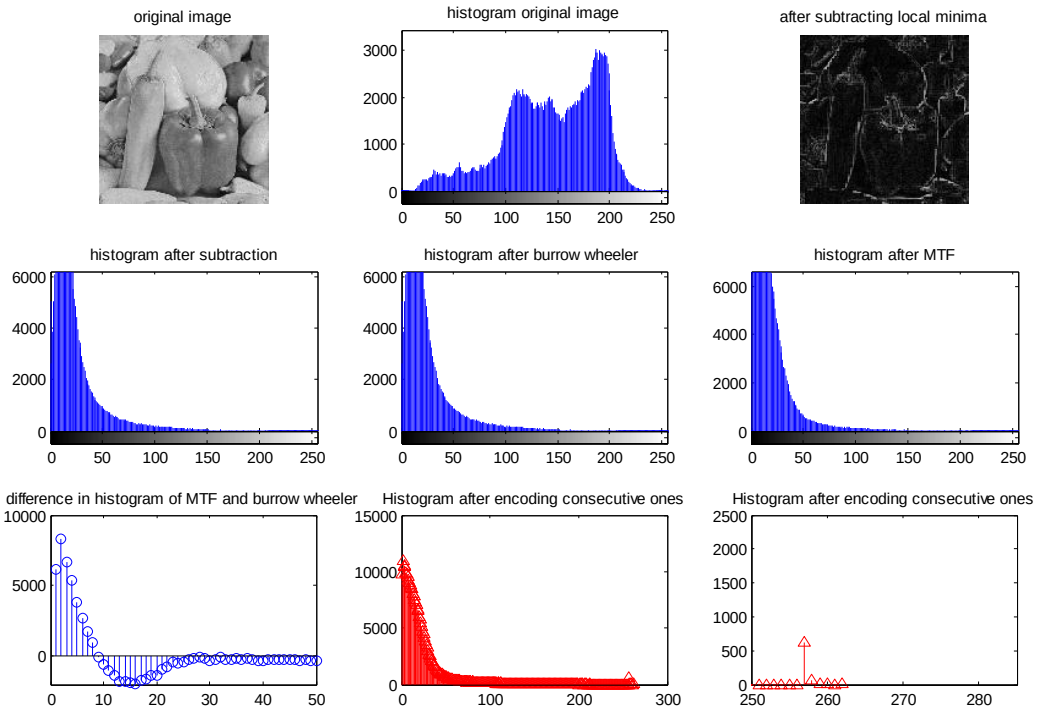

original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones

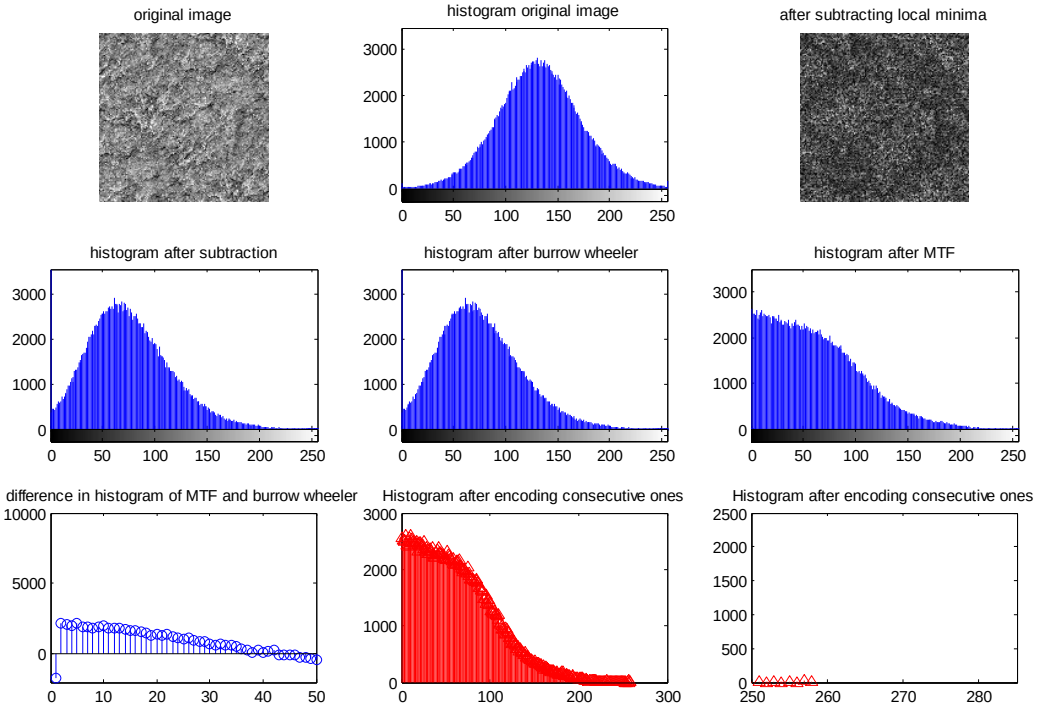

original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones

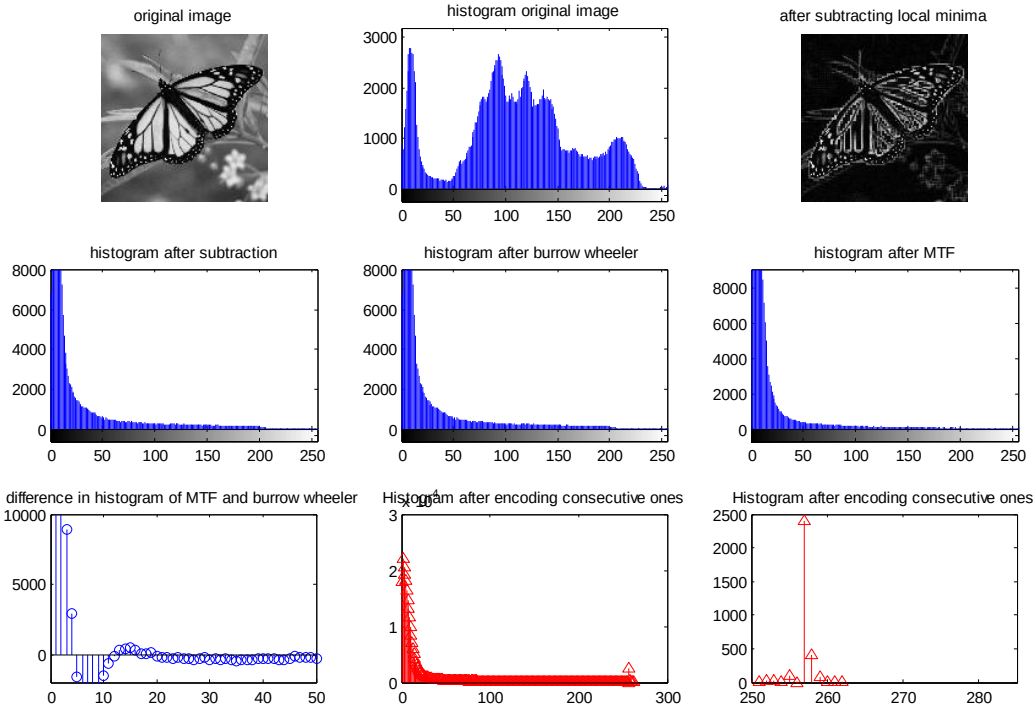

original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones

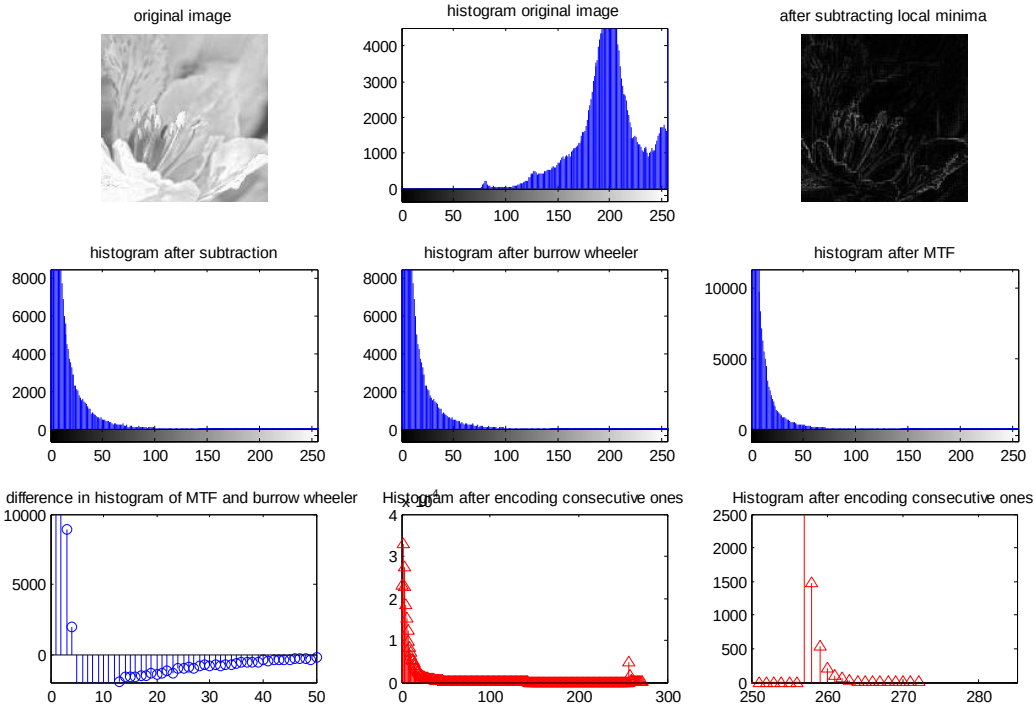

original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones

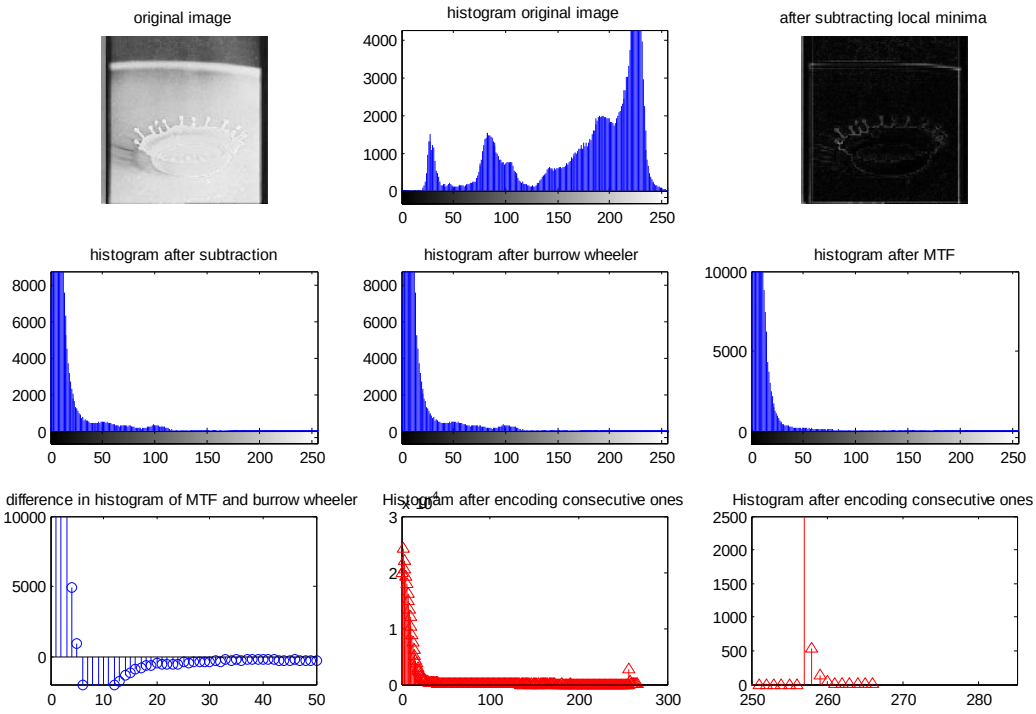

original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones

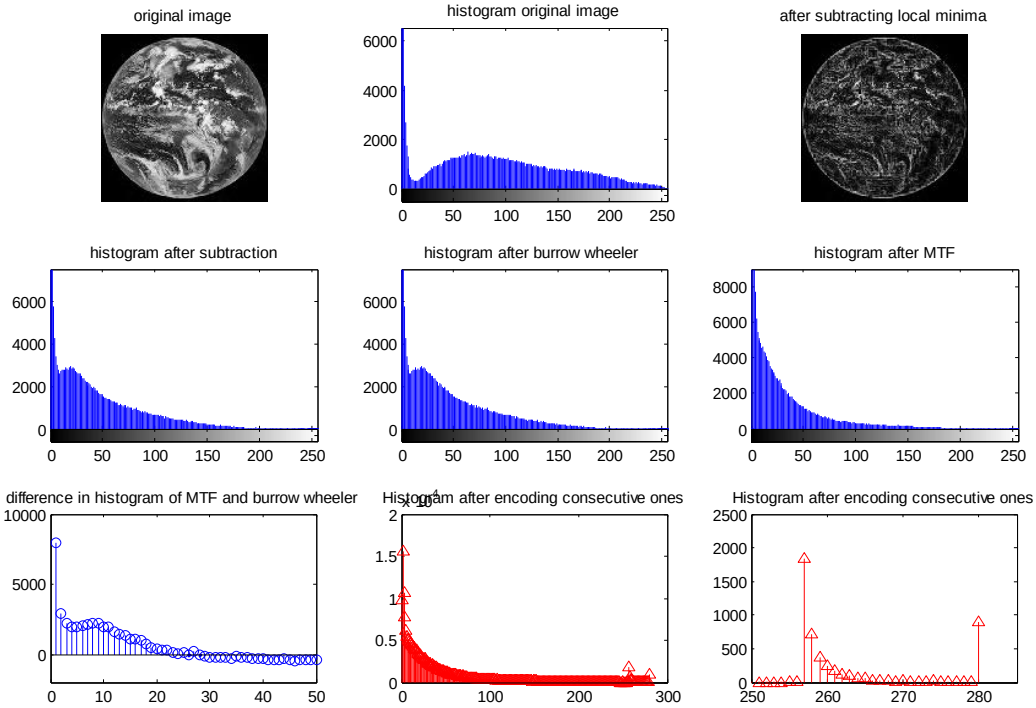

original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones

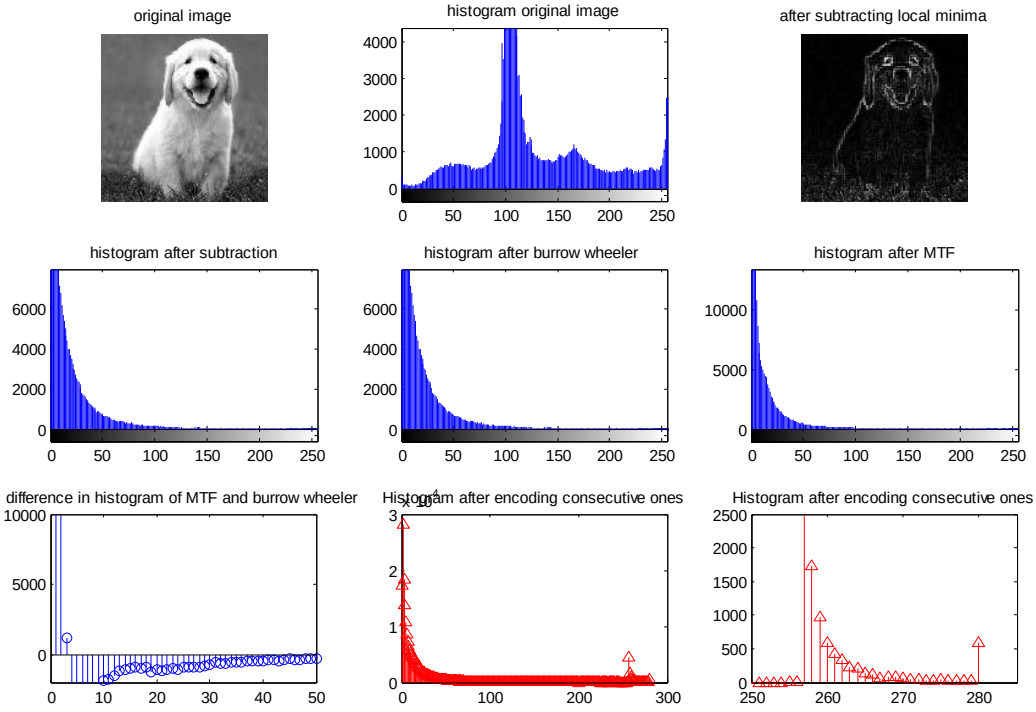

original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones

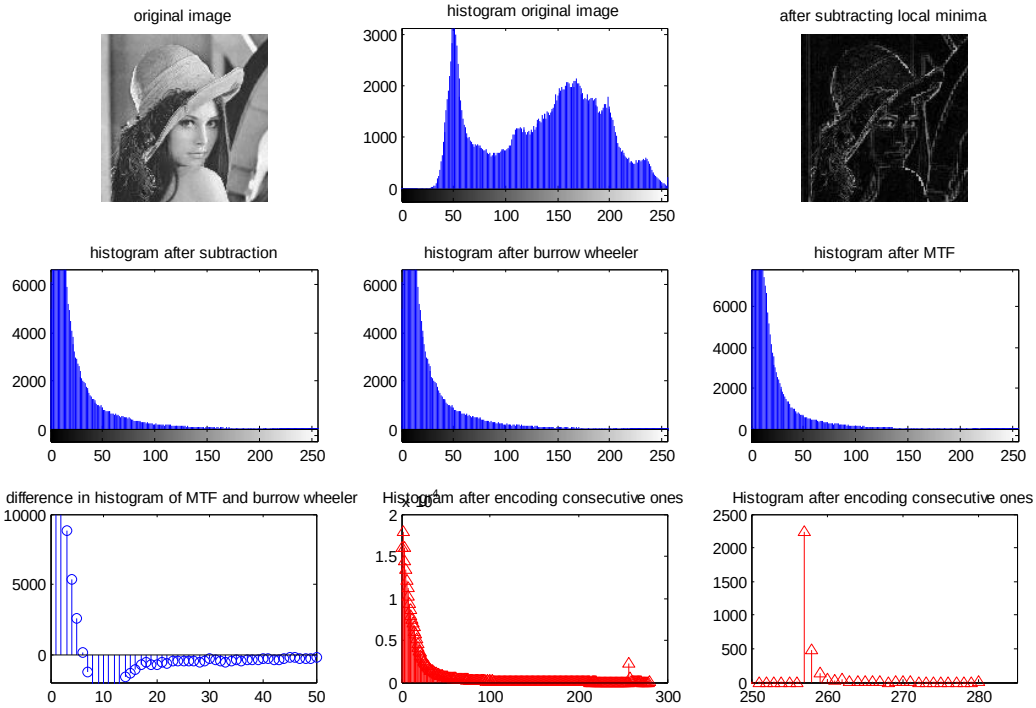

original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones


original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones


original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones


original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones


original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones


original image
histogram original image
after subtracting local minima
histogram after subtraction
histogram after burrow wheeler
histogram after MTF
difference in histogram of MTF and burrow wheeler
Histogram after encoding consecutive ones
Histogram after encoding consecutive ones

4.2 Effect of Burrow Wheeler Transform block size on compression

Image	Compression ratio for 32x32 block for BWT	Bytes required to represent 1 slice of image	Compression ratio for 64x64 block for BWT	Bytes required to represent 1 slice of image
1	1.281949841	204488.5	1.280352441	204743.625
2	1.236668302	211976	1.235860092	212114.625
3	1.709543731	153341.5	1.661107864	157812.75
4	1.317523361	198967.25	1.323477496	198072.125
5	1.170670494	223926.375	1.173891237	223312
6	1.333407936	196597	1.329304508	197203.875
7	2.320456489	112970.875	2.306004524	113678.875
8	1.70582264	153676	1.722409161	152196.125
9	1.365428451	191986.625	1.360583692	192670.25
10	1.073801201	244127.125	1.083115469	242027.75
11	1.511930553	173383.625	1.512452852	173323.75
12	1.770745221	148041.625	1.7639105	148615.25
13	1.708376712	153446.25	1.709826673	153316.125
14	1.392522887	188251.125	1.417509879	184932.75
15	1.759575652	148981.375	1.770694388	148045.875
16	1.422979763	184221.875	1.421775796	184377.875
AVG	1.505	180523	1.504	180402

4.3 Comparison of Proposed scheme with LZW algorithm.

The results of proposed scheme of Burrow Wheeler Compression algorithm are also compared with LZW algorithm. Lempel Ziv Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It is a dictionary based algorithm and strings of input are replaced by the dictionary entries. It was published as an improved implementation of the LZ78 algorithm. Two implementations of LZW algorithm are applied on the test images.

- **Implementation #1**

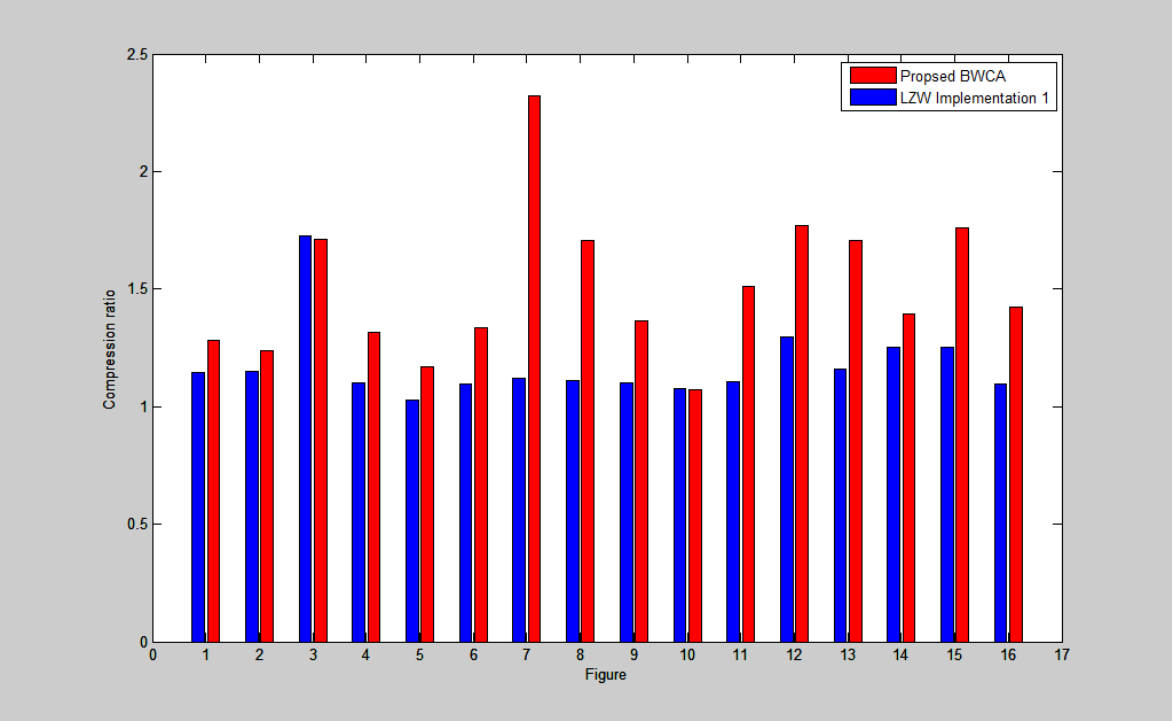
  In Implementation 1 the test images are encoded by LZW algorithm with size of dictionary taken as 1024 locations and then Huffman encoding is performed on the encoded data.

- **Implementation # 2**

  In implementation 2 the test images are encoded by LZW algorithm with size of dictionary taken as 2048 and the dictionary entries are refreshed several times during encoding of image so that the dictionary contains the most recent sequences in the dictionary and to enhance the probability of finding sequence in dictionary and then Huffman encoding is performed on the encoded data.

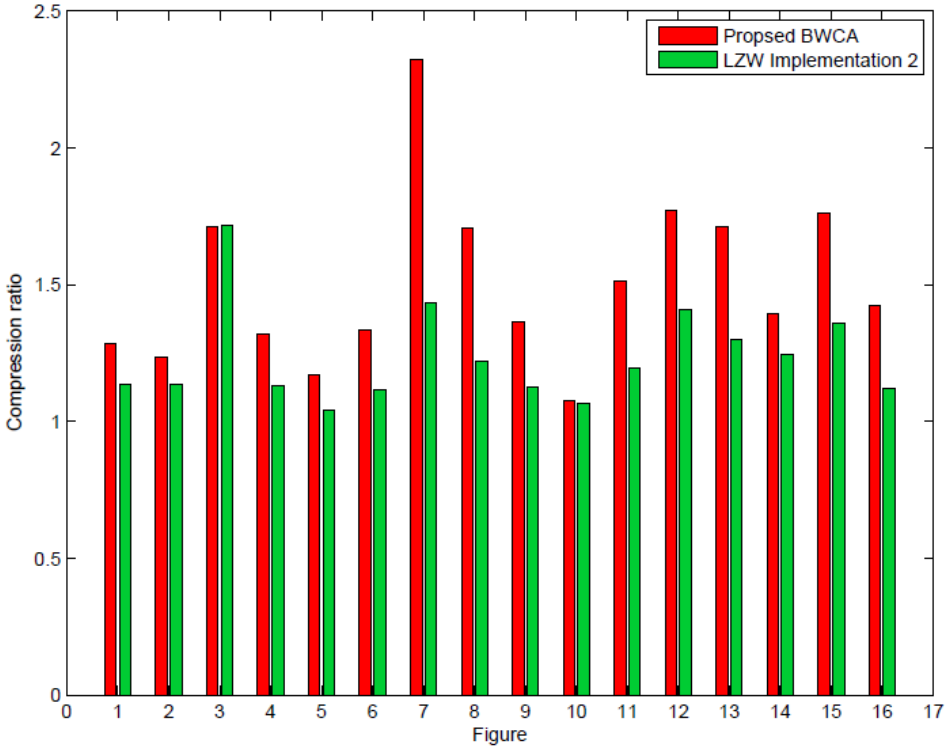
Comparison of proposed BWCA with LZW implementation #1

Image	Compression ratio	Bytes Required To Represent one slice of image
1	1.145651647	228816.5
2	1.148229079	228302.875
3	1.724820231	151983.375
4	1.103314755	237596.75
5	1.027281482	255182.25
6	1.098130225	238718.5
7	1.120753057	233899.875
8	1.112433581	235649.125
9	1.102565316	237758.25
10	1.075540334	243732.375
11	1.103559181	237544.125
12	1.296380412	202212.25
13	1.159241225	226134.125
14	1.250654205	209605.5
15	1.254346531	208988.5
16	1.098276297	238686.75
AVG	1.1763235	225925.69

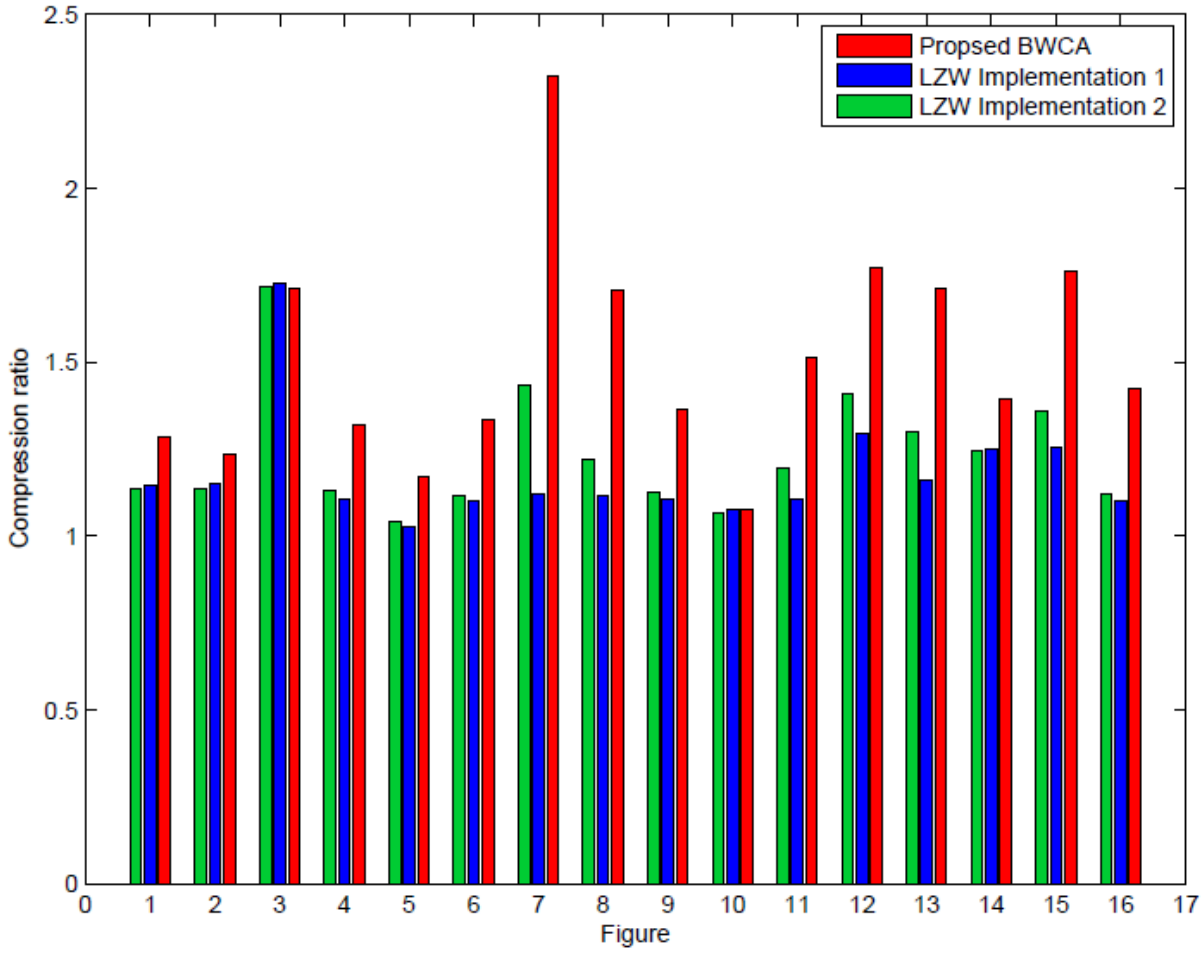


Comparison of proposed BWCA with LZW implementation #2

Image	Compression ratio	Bytes Required To Represent one slice of image
1	1.134814528	231001.625
2	1.135980621	230764.5
3	1.71478987	152872.375
4	1.13214225	231546.875
5	1.039416145	252203.125
6	1.112576401	235618.875
7	1.433822635	182828.75
8	1.220634148	214760.5
9	1.123026479	233426.375
10	1.064276743	246311.875
11	1.193448064	219652.625
12	1.409712667	185955.625
13	1.297361234	202059.375
14	1.242722129	210943.375
15	1.359808227	192780.125
16	1.1224098	233554.625
AVG	1.23355	216017.5



Comparison of proposed BWCA with LZW.



5.1 Conclusion and future work

In this thesis lossless image compression by incorporating histogram processing stage in Burrows Wheeler Compression Algorithm (BWCA) is presented. Original BWCA scheme was designed at first for the text based compression, but later on the efficiency of the algorithm proved its potential in various fields, the proposed scheme is the modified and improved version of BWCA, in which the improvements has been made to achieve even higher compression ratios for lossless image compression.

In the proposed scheme maximum compression ratio is achieved by taking the block size of 16x16 pixels for finding local minima in histogram processing stage of algorithm and by taking the block size of 32x32 pixels for BWTransform. Standard images as well as random images are chosen to ensure the improvements of proposed method.

The results of proposed scheme are also compared with LZW algorithm which is a universal dictionary based algorithm.Two variants of LZW (Lempel Ziv Welch) algorithm are implemented and are applied on test images.The results show that proposed scheme has also achieved higher compression ratios compared to LZW algorithm.

In order to increase the encoding and decoding speed the proposed scheme can be implemented on hardware technology. Field Programmable Gate Array (FPGA) based implementation of proposed method can result in even more faster and robust scheme.

References

[1] M. Burrows and D.J. Wheeler, “A Block-sorting lossless data compression”, SRC Research Report 124, Digital systems research center, Palo Alto, 1994.

[2] J. Abel, “Improvements to the Burrows-Wheeler compression algorithm: after BWT stages”, ACM Trans. Computer Systems, submitted for publication, 2003.

[3] P. Fenwick, "Block sorting text compression–final report", Technical reports 130, University of Auckland, New Zealand, Department of Computer Science. 1996.

[4] M. Schindler, “A Fast Block-sorting Algorithm for lossless Data Compression”. In Proceedings of the IEEE Data Compression Conference 1997, Snowbird, Utah, STORER, J.A. AND COHN, M. Eds. 469.

[5] K. Sadakane, “A fast algorithm for making suffix arrays and for Burrows-Wheeler transformation,” in *Proc. Data Compression Conf.* 1998, pp. 129–138.

[6] Rafeal C. Gonzalez and Richard E. Woods “Digital Image Processing” 3rd Edition, SBN:013168728X

[7] Jacob Ziv and Abraham Lempel; *Compression of Individual Sequences Via Variable-Rate Coding*, IEEE Transactions on Information Theory, September 1978

[8] D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952, pp 1098–1102.

[9] M. Asif Ali , “*lossless image compression using kernel based global structure transform (GST)*” 6th international conference on emerging technologies (ICET)