12-2015

Exclusive-or preprocessing and dictionary coding of continuous-tone images.

Takiyah K. Cooper
University of Louisville

EXCLUSIVE-OR PREPROCESSING AND DICTIONARY CODING OF CONTINUOUS-TONE IMAGES

By

Takiyah K. Cooper
B.S., University of Louisville, 2004
M.Eng., University of Louisville, 2005

A Dissertation
Submitted to the Faculty of the
J.B. Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy
in Computer Science and Engineering

Department of Computer Engineering and Computer Science
University of Louisville
Louisville, Kentucky

December 2015

EXCLUSIVE-OR PREPROCESSING AND DICTIONARY CODING
OF CONTINUOUS-TONE IMAGES

By

Takiyah K. Cooper
B.S., University of Louisville, 2004
M.Eng., University of Louisville, 2005

A Dissertation Approved on

November 30, 2015

by the following Dissertation Committee:

_____
Dissertation Director
Dr. Ahmed Desoky

_____
Dr. Mehmed Kantardzic

_____
Dr. John Naber

_____
Dr. Prasanna Sahoo

_____
Dr. Adel Elmaghraby

DEDICATION

Brenda G. Hart

for guiding me through the beginning of my collegiate journey

Cynthia M. Sanders

for guiding me to the completion of my collegiate journey

Dorothy R. McCormick

for guiding me through every journey

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Ahmed Desoky, for all of the time, instruction, counsel, and patience that he has given throughout this eight year journey. My gratitude also goes to my committee, Dr. Adel Elmaghraby, Dr. Mehmed Kantardzic, Dr. John Naber, and Dr. Prasanna Sahoo, for the coaching and direction that they have provided. This team has challenged me in ways that have made me a stronger academic with a deep passion for the possibilities of Computer Science and Engineering.

ABSTRACT

EXCLUSIVE-OR PREPROCESSING AND DICTIONARY CODING
OF CONTINUOUS-TONE IMAGES

Takiyah K. Cooper

November 30, 2015

The field of lossless image compression studies the various ways to represent image data in the most compact and efficient manner possible that also allows the image to be reproduced without any loss. One of the most efficient strategies used in lossless compression is to introduce entropy reduction through decorrelation. This study focuses on using the exclusive-or logic operator in a decorrelation filter as the preprocessing phase of lossless image compression of continuous-tone images. The exclusive-or logic operator is simply and reversibly applied to continuous-tone images for the purpose of extracting differences between neighboring pixels. Implementation of the exclusive-or operator also does not introduce data expansion. Traditional as well as innovative prediction methods are included for the creation of inputs for the exclusive-or logic based decorrelation filter. The results of the filter are then encoded by a variation of the Lempel-Ziv-Welch dictionary coder. Dictionary coding is selected for the coding phase of the algorithm because it does not require the storage of code tables or probabilities and because it is lower in complexity than other popular options such as Huffman or Arithmetic coding. The first modification of the Lempel-Ziv-Welch dictionary coder is that image data can be read in a sequence that is linear, 2-dimensional, or an adaptive combination of both. The second modification of the dictionary coder is that the coder can instead include multiple, dynamically chosen dictionaries. Experiments indicate that the exclusive-or operator based decorrelation filter when combined with a modified Lempel-Ziv-Welch dictionary coder provides compression comparable to algorithms that represent the current standard in lossless compression. The proposed algorithm provides compression performance that is below the Context-Based,

Adaptive, Lossless Image Compression (CALIC) algorithm by 23%, below the Low Complexity Lossless Compression for Images (LOCO-I) algorithm by 19%, and below the Portable Network Graphics implementation of the Deflate algorithm by 7%, but above the Zip implementation of the Deflate algorithm by 24%. The proposed algorithm uses the exclusive-or operator in the modeling phase and uses modified Lempel-Ziv-Welch dictionary coding in the coding phase to form a low complexity, reversible, and dynamic method of lossless image compression.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Lossless compression of continuous-tone images is critical in the fields of image archiving, medical imaging, and initial image capture. These fields use images as the starting point and source for varying processes. No error can be introduced into the image during processing, display, transfer, or storage. Lossless image compression addresses these needs because it is able to compress images without any data loss in the compression and decompression process. The exact image can be reproduced without error. Lossless image compression also aids in reducing the transmission and processing capacity requirements of images by reducing the file size of the image. For these reasons, there is a continued need for lossless image compression.

Lossless image compression is a subset of data compression. Data compression is a process that allows a data set to be represented in a lower bit count than the bit count of the original representation. When the original data set can be completely recovered from the compression process it is considered lossless compression. Images are a type of data available for compression. Image compression exploits the strong relationship between neighboring pixels in images or omits data not noticeable to the human eye. Lossless image compression is the intersection of image compression and lossless compression. It consists of algorithms that exploit the characteristics of images which are also able to recover the exact image after decompression.

There are many lossless compression algorithms that can be applied to continuous-tone images; some examples include Huffman coding, Arithmetic coding, Run Length coding, and the Lempel-Ziv series of coders. These algorithms are used in the creation of continuous-tone image file formats that include compression using lossless methods. PNG, Portable Network Graphics, is an image file format that is created through a process that uses a combination of the Lempel-Ziv

77 (LZ77) coding algorithm and Huffman coding algorithm. Another common file type is TIF, Tagged Image File, which is developed using a process that leverages Run Length coding and Huffman coding. Lossless image file formats use lossless image compression algorithms to generate compressed image storage.

Image compression is often a two phase procedure of modeling or preprocessing, followed by coding or compression. Preprocessing is the portion of lossless image compression that allows lossless compression algorithms to be leveraged efficiently on images. The image to be compressed is preprocessed by a series of steps designed to prepare the data for compression by the lossless compression algorithm. Preprocessing may include partitioning the data into blocks, transforming the data set into a different frequency, or data redundancy removal steps. The methods used in the preprocessing phase of a lossless image compression algorithm are selected to enhance the performance of the algorithm used in the compression phase. One of the most common methods used in preprocessing to improve compression is decorrelation, the removal of redundancy.

Decorrelation improves compression by reducing the amount of similarity between pixels in continuous-tone images. To further illustrate this statement, consider an image of a blue sky. It may simply appear as blue, but there will be subtle variations in the shade of blue throughout the image. The color variance will not only be slight, but the variations may modulate in a similar pattern as its neighbors. After similarities and patterns are removed through decorrelation, the remaining data set represents the relationships between pixel values. The amount of space required to represent the relationships in the image may require less space than the representation of the actual values of the image.

An example of a relationship that can be captured through decorrelation is the difference between neighboring pixels. A sample data set is [179 179 176 176 179 179 181 180]. Using the subtraction operator between pixels is low complexity and reversible. Although subtraction would result in a data set of small values, the values could include negative numbers. The values of the sample data set when extracted with subtraction operator, with a boundary value that is equal to

180, are [-1 0 -3 0 3 0 2 -1]. The Boolean exclusive-or operator can also be used to identify differences between pixels and can also result with a data set of small values. This operator is not only reversible, it only returns positive integers. The values of the sample data set when extracted with the exclusive-or operator are [7 0 3 0 3 0 6 1]. When two data sets have the same range of absolute values, the data set of positive integers will require less space for compression than the data set of both positive and negative integers. The exclusive-or operator is the preferred choice for decorrelation because it is low complexity, reversible, and produces positive integer results.

Table 1-1 : Sample XOR Difference

Values		Relationship	
Decimal	Binary	Binary	Decimal
179	10110011	00000111	7
179	10110011	00000000	0
176	10110000	00000011	3
176	10110000	00000000	0
179	10110011	00000011	3
179	10110011	00000000	0
181	10110101	00000110	6
180	10110100	00000001	1

Extracting the differences between neighboring pixels is a valuable decorrelation method. The size of a continuous-tone image data set can be reduced by using the exclusive-or operator to extract neighboring pixel differences as the representation of the original image. The compression performance of lossless compression algorithms on the data set reduced by exclusive-or decorrelation will be greater than the performance of lossless compression algorithms on the original data set. This study focuses on the application of the exclusive-or operator as a decorrelation methodology for the improvement of lossless compression algorithms on images.

The motivation behind this study is the current state of lossless image compression. Lossless image compression performance lags behind its lossy counterpart. Lossy image compression is also a process that reduces the bit count required to represent an image. However, lossy image

compression uses methods that introduce data loss during the compression process. This data loss prevents the image from being recreated in its exact form during decompression process. Lossy compression algorithms have an average compression ratio of 20:1. Lossless compression algorithms have an average compression ratio of 2.5:1 [1]. Fields that do not allow the use of lossy image compression are at a disadvantage and will benefit from improved lossless image compression ratios.

Research shows that the performance of lossless image compression algorithms can be improved by including reversible decorrelation methods in preprocessing. The authors of *Differential Block Coding of Bilevel Images* [2] improve the performance of an existing bilevel image compression algorithm by a factor of 4. Compression performance is improved through the use of a differencing operation prior to encoding. In *Context-Based, Adaptive, Lossless Image Coding* [3], the authors focus on an adaptive and predictive preprocessing method that is compatible with various lossless compression algorithms. The combination of the preprocessing method with lossless compression results in an algorithm that outperforms the lossless Joint Photographic Experts Group standard by 12%. Decorrelation through preprocessing is an impactful way to improve lossless compression algorithms for images.

The two papers mentioned above, [2] and [3], are two very different examples of effective preprocessing. The first paper, [2], includes a decorrelation method in preprocessing that is targeted to a specific lossless compression algorithm. The targeted design limits the ability to integrate the decorrelation methods with other lossless compression algorithms. The preprocessing methods of second paper, [3], are more complex and have several calculation intensive steps. The complexity of the design is significant enough to prevent the algorithm from being adopted as the lossless JPEG standard although the algorithm outperforms its peers. The field of lossless image compression needs a decorrelation methodology that integrates well with varying lossless compression algorithms and has low complexity.

The exclusive-or operator is as a low complexity and reversible operator that can be leveraged in decorrelation. Significant decorrelation results using the exclusive-or operator have been proven

on wireless sensor network data [4] and bilevel images [2]. The compression improvements that result from applying the exclusive-or operator to other data types have proved the exclusive-or operator to be significant. However, the exclusive-or operator has not yet been examined extensively the field of lossless image compression for continuous-tone images. One of the goals of this study is to identify images that will respond well to decorrelation through the exclusive-or operator. A better understanding of the exclusive-or operator as a differential coder could lead to enhanced decorrelation and improved lossless image compression performance.

The exclusive-or operator has not only been applied to varying data types, it has also been applied to these data types using varying methods. In [5], the purpose of use is for data reduction. The exclusive-or logic operator is used in the preprocessing phase of the CoXoH compression algorithm with the purpose of reducing the input data by half prior to Huffman Coding. In [6] and [4], the exclusive-or operator is used as a data transform. The authors of [6] aim to produce long runs of zeros and ones by applying neighboring bit-wise exclusive-or logic operations to the original data set as a transform. This same exclusive-or logic based differential transform is used in [4] prior to the application of Run-Length Encoding. Although all of these authors utilize the exclusive-or logic operator, direct analysis of the operator as a decorrelation method is either missing or incomplete. A goal of this study will be to identify an effective method of applying the exclusive-or operator in the decorrelation of continuous-tone images. This may allow images to gain the same compression improvements witnessed by other data types that leverage the exclusive-or operator in the preprocessing phase of their compression algorithms.

Initial research examined the exclusive-or operator as a differential filter on continuous-tone images prior to Huffman coding. One row at a time, the current row of the image is applied to the next row of the image using the exclusive-or operator. This repeats in a cyclic manner to produce the filtered image as an output. The filtered image is the collection of exclusive-or differences between neighboring pixels. To identify the impact of applying the filter, the original image and the filtered image were individually compressed using Huffman coding. The impact of using the exclusive-or filter was measured by calculating the amount of compression improvement and

entropy reduction between the filtered and unfiltered data set. Preliminary research has shown that entropy is reduced and compression is increased by using the exclusive-or operator as a decorrelation filter.

This dissertation includes efforts in both the preprocessing and the coding phases of lossless image compression. The preprocessing phase is centered on the use of the exclusive-or operator as a decorrelation filter. The filter window size used in preliminary work will be expanded to differing window sizes and dimensions. Applications of the various filter window sizes will be compared and contrasted to identify the more optimal application of the exclusive-or filter.

The data produced by the exclusive-or filter is then coded. Initial coding was conducted using Huffman coding. This dissertation includes coding the filtered data using Lempel-Ziv-Welch (LZW) dictionary coding. Compression performance between multiple variations of the LZW dictionary coder is examined. Variations include the formation of dictionary words as well as dictionary count.

The use of the exclusive-or operator as a differential filter combined with two dimensional LZW coding may introduce substantial improvement to the field of lossless image compression. The performance of algorithm created through this study is compared to algorithms that represent the current standard in lossless compression. Through this study, a lossless image compression algorithm is introduced that is low complexity, reversible, and dynamic.

CHAPTER 2

LOSSLESS IMAGE COMPRESSION OVERVIEW

2.1 BACKGROUND IN IMAGE COMPRESSION

2.1.1 Huffman Coding

Huffman is an example of an entropy coder. The goal of this type of coder is to create a variable length code word set that most closely represents the lower bound identified by entropy. Entropy is based on the probability of the data set. Shannon's theorem defines the optimal code length of a symbol as $-\log_b P$. The variable P is the probability of a symbol and b is the size of the output alphabet. This equation indicates that symbols with higher probability should have shorter code words and symbols with lower probability should have longer code words. Optimal code word length is listed in the last column of Table 2-1.

Table 2-1 : Optimal Code Word Length for ABRACADABRA

Symbol	Probability	$-\log_2 P$
D	0.09	3.47
C	0.09	3.47
R	0.18	2.47
B	0.18	2.47
A	0.45	1.15

The first step of Huffman coding is generating the Huffman Tree. The data set that will be used or this example is the word "ABRACADABRA". The symbols, A, B, C, D, and R, are ordered by the frequency of their occurrence. The two nodes with the lowest frequencies, D and C, are combined to make a subtree with a frequency that represents the sum of the 2 nodes.

Table 2-2 : Huffman Symbol Step 1

Symbol	Count	Probability
D	1	9.09%
C	1	9.09%
R	2	18.18%
B	2	18.18%
A	5	45.45%

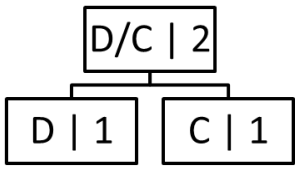

Figure 2-1:Huffman Tree Build D/C

The new subtree and its value are added back in to the ordered list as node D/C. The process repeats and the next two entries with the lowest frequencies are combined. This time, the subtree D/C is combined with node R. The new subtree with nodes D, C, and R and a frequency of 4 is then added back in to the ordered list.

Table 2-3 : Huffman Symbol Step 2

Symbol	Count	Probability
D/C	2	18.18%
R	2	18.18%
B	2	18.18%
A	5	45.45%

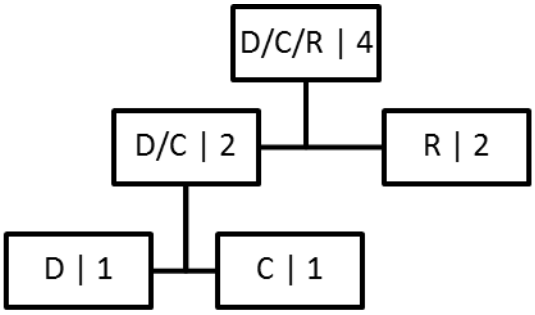

Figure 2-2: Huffman Tree Build D/C/R

The process repeats for node B and subtree D/C/R to create a new subtree with nodes B, D, C, and R and a frequency of 6.

Table 2-4 : Huffman Symbol Step 2

Symbol	Count	Probability
B	2	18.18%
D/C/R	4	36.36%
A	5	45.45%

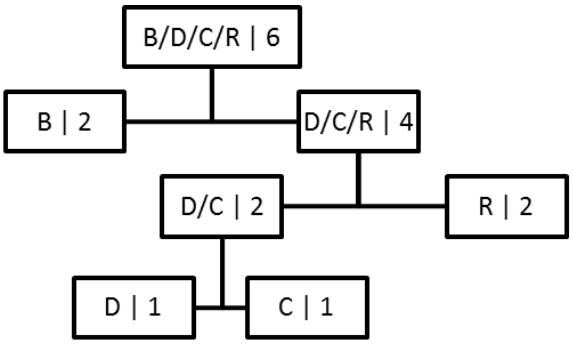

Figure 2-3: Huffman Tree Build B/D/C/R

The new subtree is then added back to the ordered list. The last 2 entrees in the ordered list are then combined to complete the Huffman tree.

Table 2-5 : Huffman Symbol Step 4

Symbol	Count	Probability
A	5	45.45%
B/D/C/R	6	54.54%

Table 2-6 : Huffman Symbol Step 5

Symbol	Count	Probability
A/B/D/C/R	11	100.00%

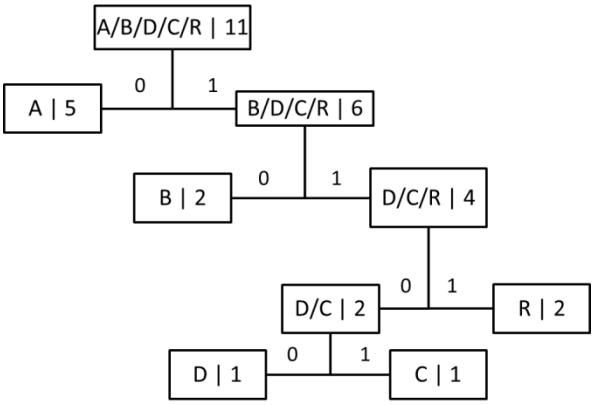

Figure 2-4: Huffman Tree Build A/B/D/C/R

Table 2-7 : Huffman Symbol Step 6

Symbol	Count	Probability	Code Word
D	1	9.09%	1100
C	1	9.09%	1101
R	2	18.18%	111
B	2	18.18%	10
A	5	45.45%	0

To assign code words to the symbols, the process begins at the root node. Traverse the tree from root node to each symbol node. Assign a 0 when moving to left children and assign a 1

when moving to right children. The code word for each symbol is created through the path of each node. Variable length code words are possible because each code word is a prefix code. Prefix codes are those that have no code word that is the prefix for another valid code word. When encoding, each symbol in the input ABRACADABRA is replaced by the Huffman code word for the symbol. This process generates the encoded value of 0 10 111 0 1101 0 1100 0 10 111 0.

```
A B  R   A C    A D    A B  R   A
0 10 111 0 1101 0 1100 0 10 111 0
```

Figure 2-5: Huffman Coded ABRACADABRA

2.1.2 Image Types

An image can be defined “as a set of two-dimensional arrays of integer data (the samples), represented with a given precision (number of bits per component)” [7]. Data samples in the context of image compression are pixels. Each individual array of the set is a component. One example of a multi-component image is seen in the color image diagramed in Figure 2-6 below. The example image is comprised of 3 components. There is one component for each of the intensity measures of red, green, and blue. A single component of pixel intensities with more than 2 distinct values is considered a grayscale image in the context of this study. This study will include both grayscale and RGB color images.

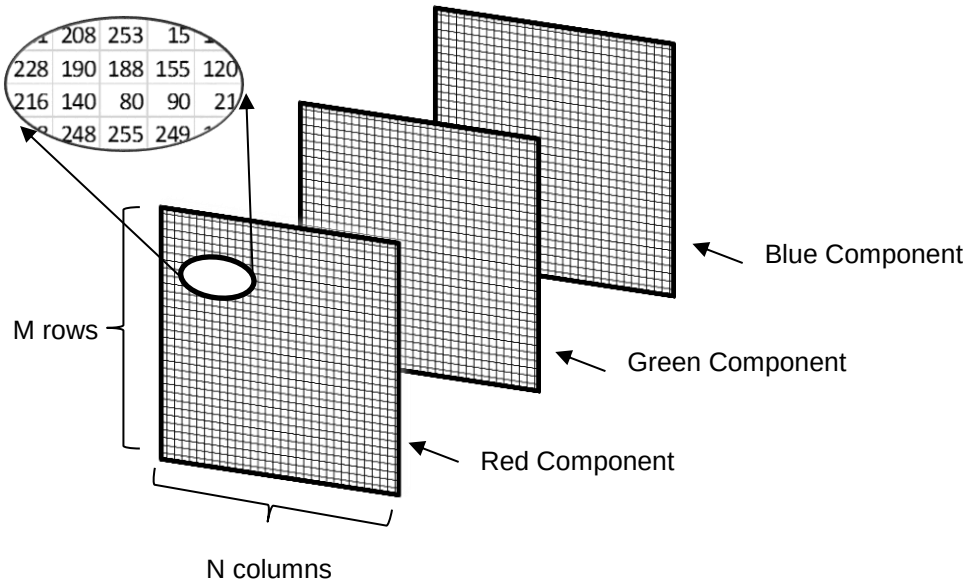

Figure 2-6: Multi-Spectral Image

The example in Figure 2-6 is also considered a continuous-tone image. In continuous-tone images each sample, or pixel, of a components can be represented by an individual value. Images that require multiple values to represent individual pixels are considered halftone images. Halftone image pixels are represented by a combination of dots of varying size, shape, or spacing [8]. Vintage comic books and newsprint often use halftone images for their graphics. An example of halftone coloring is given in Figure 2-7.

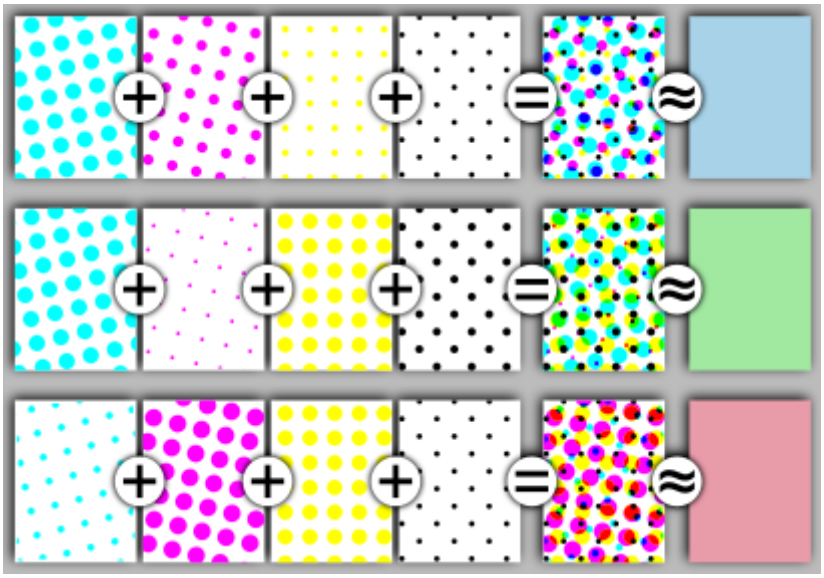

Figure 2-7: Halftone Examples

The example in Figure 2-6 is a multi-spectral image based on color space. A color space is a combination of components that represent a color. Each component has a value that is used to describe the intensity of the component. Figure 2-6 is in the Red-Green-Blue (RGB) color space and has a component with values representing the intensity of each color. The values of each pixel in the color components combine to make the color of each pixel in the image.

There are many variations of the multi-spectral image. A slight variation of multi-spectral images based on color space is the High Dynamic Range image (HDR) discussed in [8] and [9]. This image type also contains multiple components, but the pixel values are not limited to the range of 0 to 255. HDR images generally use 16-bits for half-precision and 32-bits for full precision [8]. Hyper-spectral images are another variation of multi-spectral images. This image type is uniquely identified because it often contains 10s to 100s of components instead of just 3 like RGB color images. In each of the examples, color space based, HDR, and hyper-spectral, the components contain information that combines to produce a single continuous-tone color image view.

A secondary type of continuous-tone color image is the color-mapped image. Color-mapped images, also known as pseudo-color images or palletized images, are two dimensional arrays with one component. The values of each pixel in the component represent an index to a list of

available colors. Palletized images are beneficial when the list of colors is small. A color image with only 255 colors can be represented with 8 bits per pixel. The same image in the RGB color space would require 8 bits per each of the 3 components. The palletized image has a lower bit count than the RGB image.

An additional single component image type is a binary image. Binary images are images that have pixels that can only be one of two values. Text documents are a common kind of binary image. Although it is common for the pixel values to be 1 or 0, the value options are not limited to this range. In binary images, each pixel in the image is either the background of the foreground color.

2.1.3 Color Spaces and Color Space Transforms

Colors can be represented using many different variables. Colorfulness and chrominance variables, such as red, describe visual color. Saturation and luminance variables describe brightness. Combinations of these and other variables form descriptors for color image pixels. RGB is a color space combing measures of colorfulness of red, green, and blue. Cyan, Magenta, Yellow, and Black (CMYB) are another common colorfulness color space. The YIQ color space combines Y, luminance, with I and Q, chrominance, to represent a color space. Similar to YIQ, the LAB and LUV color spaces combine L, lightness, with A and B, chrominance for LAB or U and V, chrominance for LUV to represent a color space. Multiple color spaces exist to meet the needs varying applications.

Color space transforms are the formulas and methods used to represent a color in one color space in a different color space. The authors of [9] conducted a comparative study of color transforms. The goal of their study was to analyze how the application of color transforms can improve the performance of compression algorithms and to generate an integer reversible color transform. Because standard color transforms introduce errors from rounding, the authors propose a reversible integer to integer color transform for use in lossless image compression procedures. The goal of creating an integer reversible color transform is also in the research conducted in [10]. The authors of [10] present an integer reversible version of the YIQ, luminance

and chrominance, color transform for use with lossless image compression. As discussed in [9], it is possible for a color transform to be mathematically based on the image being compressed. This is accomplished by using the statistics of the image to be converted in the formation of the formula for color space transformation. Images can be transformed from one color space to another in an effort to format image data in ways that make it perform better in compression algorithms.

2.1.4 Decorrelation

Continuous-tone images usually have a high amount of correlation between pixels. Correlation is the relationship between pixels. The values of neighboring pixels usually vary together, which indicates a strong correlation. This correlation can be removed in a way that does not cause data loss to reduce the entropy of data that needs to be stored or transmitted. Entropy can be described as a measure of uncertainty. In a completely random data set of 256 possible values, the entropy will be the number of bits, 8, required to represent the possible data values. In a data block of only a single value, the entropy will be 0 because it is known that only one value is possible. Based on this, Shannon's theory, reduced entropy can lead to increased data compression ratios.

Correlation can be reduced through the process of decorrelation. An effective method of decorrelation is transformation. An example of a transform is the Discrete Cosine Transform (DCT). This transform converts a block of correlated data into an equal sized block with a few key values and several other values biased to zero. In order to accommodate lossless coding, some transforms have been rewritten to ensure the results of the transform are integers. To accomplish this limitation the transforms use a combination of integer reversible, integer-to-integer, shifting, lifting, multi-lifting, and integer-based arithmetic processes. Progressive transforms, such as DCT and Karhunen-Loeve Transform (KLT) as described in [19], can enable progressive coding, also known as lossy-to-lossless coding.

Another effective method of decorrelation is prediction. An example of prediction is Differential Pulse Code Modulation (DPCM). DPCM creates a decorrelated error signal by capturing the

difference between a predicted and actual value at each data point. It is also possible to have a decorrelation method that combines both prediction and transform as described in [11]. The authors use a transform in preprocessing and prediction prior to encoding data. Prediction systems have an advantage over transform systems because they are typically lower in complexity and typically execute at a faster rate [12]. It is noted in [13] that predictive coding does not allow the ability to transmit data gradually by loading from low quality to full quality, known as progressive coding. This statement is proven incorrect by the research of [14]. The authors of [14] create a predictive based algorithm that allows for progressive coding.

2.1.5 Correlation

Decorrelation is generally discussed without mention of any correlation measures. The reader is left to measure the success of decorrelation solely on the compression that is produced in its use. This does not enable the reader to gain insight into the actual measure of decorrelation outside of the impact of the coding method that was applied. To properly measure decorrelation, the amount of correlation in the original data set would need to be compared to the amount of correlation that exists in the data set after decorrelation has been applied.

One reason for the lack of analysis on correlation could be correlation within image processing varies from standard statistical applications of correlation. In statistical applications, correlation is a measure of how much two sets of data change together. In the realm of image compression the measure of change is within a single data set, the image. Because there is only one data set, there are multiple ways to apply correlation. No single application can be considered to be a standard and each application's effectiveness is subjective to the decorrelation method to be used.

For this dissertation, correlation is calculated using the Matlab *corrcoef* function. The function is built upon the formula (1. The matrix representation of an image is converted to a column array by reshaping values in a column-wise manner, top to bottom and left to right. The column array representation of the image is vector, *x*. The second vector, *y*, is created by shifting the array up by one element. The first value is removed and added to the end of the array. The last variable,

n, is the number of elements in the array. This configuration of the image measures the correlation between each pixel and the one above it in the image.

$$\frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \quad (1)$$

2.2 BACKGROUND IN DATA COMPRESSION

2.2.1 Data Compression Subsets

Image compression has the same goal as general data compression. The goal is to represent data in its most minimal form [7]. Data compression is usually accomplished through the removal of redundancy and is an effort to reduce storage and transmission capacity requirements [15]. Data compression techniques can be divided into two categories: lossy and lossless [16]. When a compression method can be reversed to reproduce an exact replication of the original data set, that method is considered lossless. Methods that are unable to reproduce or exactly replicate the original data set are categorized as lossy [15]. Both categories, lossy and lossless, utilize redundancy reduction measures. Redundancy is defined as the additional bits that represent data that could be represented in fewer bits. Only lossy methods are able to leverage irrelevancy reduction [17]. Irrelevance is defined as information that is purposely removed to reduce the bit count of the data. Examples of irrelevance include the omission of audio data from an audio file that cannot be perceived by a listener and omission of color data from an image or video that is unnoticeable to a viewer. The ability to omit data generates greater compression ratios and allows lossy methods to generally outperform lossless methods [16].

2.2.2 Data Compression Research Approaches

The methodologies, codes, and algorithms under review in this study can also be categorized by their research approach. The most common approaches are either efficiency conscious or compression conscious. Efficiency conscious methods focus more on improving computational complexity, run time, memory usage, and other performance factors. Compression ratio is not the solitary focus of efficiency conscious methods. Compression conscious methods focus on

finding the greatest compression ratio possible with less attention given to the secondary performance variables. Articles [18] and [19] are two examples of efficiency conscious research. The aim of their research is to improve the computational complexity of select elements of existing methodologies. In contrast, “the UCM algorithm was intended, according to its authors, not for practical use but for providing a theoretical framework of lossless image coding, and for computing a lower bound for achievable code length by other more practical methods such as CALIC” [18].

Compression conscious algorithms, like Universal Context Modeling (UCM), can be considered a theoretical framework because they introduce ideas and methods that produce strong compression results. The ideas and methods that are introduced can then be further explored and refined by others. Other researchers use the original theories of compression conscious algorithms to generate variations that are more efficiently implemented while maintaining most of the compression performance of the original design. Compression conscious research helps to push the bounds of the lossless data compression field. Efficiency conscious research is valuable because it brings the theoretical improvements found in compression conscious research into applicable reality.

2.3 LOSSLESS COMPRESSION ALGORITHMS

2.3.1 Processing Algorithms by Component

A practice seen in multiple papers is to apply compression algorithms to each separate component of a multi-spectral image. The output from applying the algorithm to individual components is then combined to represent the entire compressed image. Figure 2-6 shows a multi-spectral image that is made of 3 components. This is a RGB color image with a red, green and blue component. Each of the 3 components can be treated as grayscale images. Each of the grayscale images, components, can be compressed using a lossless image compression algorithm. The combined results represent the entire image and can be decompressed to recover an exact copy of the original color image.

Compressing each component separately does not leverage the spectral correlation that often exists in images. Spectral correlation is the relationship between components of an image. Simple Inter-Color Lossless Image Coder (SICLIC) [20] [7] and Inter-Band CALIC (IB-CALIC) [21] [7] are two algorithms that process images using the information of other components while processing a single component. Other methods incorporate the benefits of spectral correlation by applying a color space transform to the original color space prior to processing. The authors of [7] describe a compression improvement of 25% to 30% simply by applying the [R-G G B-G] color space transform to an RGB image prior to compression by JPEG-LS. The performance of this transform is evidence that there is a level of redundancy between components that can be leveraged to improve lossless image compression.

2.3.2 Lempel-Ziv-Welch Coding

An alternative to entropy based compression algorithms are dictionary based compression algorithms. The Lempel-Ziv-Welch (LZW) algorithm is an example of a dictionary based lossless data compression algorithm. This algorithm begins with a static dictionary of symbols and creates new entries as the data is processed. In this example, the alphabet from A to Z is used. Each symbol is assigned an ordered code. Because there are 26 letters in the alphabet, a minimum of 5 binary digits will be required. A representation of 5 bits allows for 32 dictionary entries. To allow for dictionary growth 6 binary digits will be used to represent symbols. The additional bit allows for a dictionary with 64 entries.

Table 2-8 : LZW Initial Symbol Set

String	Code word
A	000000
B	000001
C	000010
…	…
M	001101
…	
Z	011010

Once the initial dictionary is set, the input string is then processed. The longest string in the dictionary that matches the prefix of the input data is identified, A. The code word for the dictionary entry is output, 000000. A new entry is added to the dictionary. The entry consists of the concatenation of the identified string and the next character in the input, AB. The code word for the new dictionary entry is the sequence, 011011. The process then repeats for the duration of the input to generate the encoded value of 000000 000001 001010 000000 000010 000000 000011 011011 011101.

Table 2-9 : LZW Processing

Match String	Output	New Dictionary Entry	Code word
A	000000	AB	011011
B	000001	BR	011100
R	001010	RA	011101
A	000000	AC	011110
C	000010	CA	011111
A	000000	AD	100000
D	000011	DA	100001
AB	011011	ABR	100010
RA	011101		

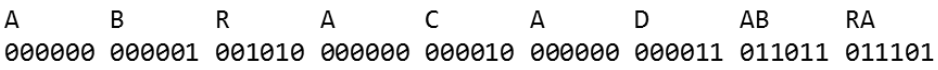

Figure 2-8: Lempel-Ziv-Welch Coding

2.3.3 Comparison of Huffman and Lempel-Ziv-Welch Coding

The Huffman and LWZ algorithms work using two different strategies. Huffman coding is very productive when the probabilities of symbols are already known and each symbol is pulled from a known distribution. Huffman focuses on representing the most frequent symbols with short code words, thereby reducing the size of the greatest portion of the data set. LZW does not require prior knowledge of symbol probabilities and is an adaptive algorithm that adjusts as the data set is processed. LZW is capable of representing multiple symbols in a single code word. This

capability reduces the amount of code words used to represent the original set of images. The performance of the Huffman algorithm is greatly dependent on the probability distribution of the data set. The performance of the LZW algorithm is greatly dependent on the amount of repetition of patterns in a data set. In the examples provided above it can be assumed that the string ABRACADABRA will require 88 bits of data to represent in the American Standard Code for Information Interchange (ASCII) binary representation of 8-bit character representations. Huffman encoding reduces the representation to 23 bits and a 3.83 compression ratio while LZW reduces the representation to 54 bits and a 1.63 compression ratio.

2.3.4 Foundational Data Compression Algorithms

Modern lossless compression methodologies often are modifications or combinations of foundational algorithms. Although a lot of change and growth has happened in the modeling phase of compression, the coding phase continues to rely on established foundational algorithms such as Huffman coding or Arithmetic coding. These algorithms are described as foundational because they represent some of the original theories and initial methods of the field of lossless data compression. Novel application of foundational algorithms is seen in [22]. The compression method described by the authors is the same as Run Length Encoding (RLE). RLE processes runs of repetitious input, such as AAAAABBBCCCCCCCCBBBB, and converts it into a list count, such as 5A3B8C4B. The RLE compression method is applied to medical images. The application of RLE is new, but there is no difference in the encoding methods of RLE.

Some of the earliest and most popular algorithms in the field of lossless data compression are described in [23]. The authors discuss Shannon-Fano, Huffman, Arithmetic, and dictionary coding. The Shannon-Fano algorithm described in the article is founded on information theory. This algorithm generates variable length code based on the probability of the symbols in the data set. Although the Huffman coding and the Shannon-Fano algorithms are both entropy encoders, the compression results of Huffman coding are able to exceed those of the Shannon-Fano algorithm. The stronger compression performance of Huffman coding is due its more optimal building of the binary tree from which the variable length code is derived.

Arithmetic coding is similar to the Shannon-Fano and Huffman algorithms because they are all probability based entropy coders. However, Arithmetic coding differs in coding procedure by representing the entire encoded data set as a single number. The coding procedure uses a cyclic method that divides the range 0 to 1 based on symbol probability. In Figure 2-9, the probabilities of [a b c] = [0.2 0.5 0.3]. To encode the string "babc", the input string is processed one character at a time. The range 0-1 is partitioned based on the probabilities of each possible character, [{0-0.2} {0.2-0.7} {0.7-1}]. Because the first character is b the range of {0.2-0.7} is selected. This range is then partitioned based on the probabilities of each possible character. The range of {0.2-0.7} is partitioned to [{0.2-0.3} {0.3-0.55} {0.55-0.7}]. This process of selection and partitioning continues until the end of the string is reached. The final encoded value will be a number that falls within the range of all of the combined encoded symbol probabilities.

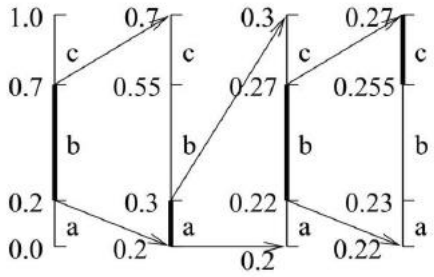

Figure 2-9: Arithmetic Coding Example

A last foundational algorithm type is dictionary coding. Many dictionary coding methods for compression are founded on research by A. Lempel and J. Ziv. The Lempel-Ziv series of coders capitalize on redundancy reduction by replacing frequently occurring patterns with an index to the dictionary that stores them.

All of the discussed foundational algorithms represent some of the most successful early algorithms in lossless compression. Huffman coding, Arithmetic coding, and Lempel-Ziv related coding continue to have a strong presence in modern lossless compression algorithms. The value of these foundational algorithms continues to grow with the exploration of improved preprocessing methods.

2.4 IMAGE COMPRESSION ALGORITHMS

2.4.1 Binary Lossless Image Compression Algorithms

Each image type has its own characteristics that may aid in the process of lossless image compression. Many image compression algorithms are tailored for particular image types to leverage the characteristics that are unique to that image type. Logic compression as described in [24] is designed to capitalize on the characteristics of binary images. A context-based sequential logic method using Ordered Binary Decision Diagrams (OBDDs) is presented in [25] for the purpose of compressing binary images. Although binary image compression algorithms are designed for the binary image type, the algorithms do not have to be limited to use on only the image type for which it is designed. It is possible to apply algorithms designed for one image type to other image types by modifying the application of the algorithm.

Binary image algorithms are designed for data sets that have at most 2 distinct values. These algorithms normally cannot be directly applied to grayscale images. A grayscale image consists of a single component with multiple pixel values. If a component's pixel values range from 0 to 255, each pixel is represented by a binary number of 8 bits. The component can then be partitioned into 8 components, also known as bit-planes. Each component represents a bit of the complete 8 bit pixel value. Each bit-plane therefore represents a binary image of each bit of the possible 8 bit range. This partitioning creates multiple binary images that allow binary image algorithms to be applied at each bit-plane of a grayscale image.

Some multi-spectral images also contain areas within them that may have binary regions. If a color image includes an area of only 1 or 2 values, this area may have a significantly lower amount of entropy than the rest of the image. Algorithms tailored only to multi-spectral images may not be able to take advantage of the change of entropy in the binary area of the image. As a result, some designers are creating adaptive image compression algorithms that can automatically detect binary regions and apply binary image based coding methods. CALIC is able to capitalize on areas of low entropy by including a binary compression component that

greatly improves its compression performance results [3]. Binary image algorithms tend to be most valuable when applied to data with low entropy.

2.4.2 Color-Mapped Image Compression Algorithms

Color-mapped images are images containing a single component with pixel values representing the index of a color in a set color-map. The index of a color in a color-map may have no relation to its color value. The lack of relation between index and color causes the pixels of images in color-map format to lose the structure of the colors actually represented. Colors that are visually adjacent to each other in a color space may have indexes that greatly vary. Color-mapped images lack the correlation of color space based continuous tone images, but retain the large symbol set. As a result, the values of the color-mapped image pixel indexes will have a high amount of entropy and perform poorly in most compression algorithms.

Compression theories for color-mapped images are under review in [12]. The focus of many of the methods reviewed is on the pre-processing element of color-map sorting rather than the actual compression method being used. The goal of sorting color-maps is to restore the lost structure from pixel color values, which reduces first-order entropy [12]. This method is needed to address the smoothness, predictability, and entropy needs of most lossless compression algorithms. One of the measures of success in color-map sorting is entropy reduction. The authors confirm in [12] that “restoration of the structure increases the efficiency of lossless compression and permits the use of lossy compression algorithms”.

Sorting methodologies Greedy Sorting and Simulated Annealing are used in [12] to sort color-maps. The first-order entropy that results from the sorted color-mapped images in the RGB, LAB, and LUV color spaces is improved. Sorting by the LUV color space gives the best results in the study. Compression of the sorted image using Lempel–Ziv–Welch (LZW) is also assessed in [12]. Compression results indicate that there is a 1 to 2 bit savings per pixel depending on the image being compressed. Although the authors of [12] address the matter in the context of lossy compression, both transform and prediction decorrelation methods are proven to be applicable to

sorted color-mapped images. The authors verify lossless compression methodologies on sorted color-mapped images through the application of DCT coding and Edge Preserving DPCM coding.

Sorting of color-maps can also be completed using optimization-based methods. In [26] the problem of color-map sorting is treated as a variation of the shortest paths problem and the approximate solution of the Traveling Salesman is applied. The Traveling Salesman problem is the problem of identifying the shortest path between points that starts and ends at the same point. The path must visit each point only once and is defined using knowledge of the list of points and the distances between them. This problem has many approximate solutions, but like the previously discussed color-mapped sorting methods, none of the solutions are exact.

Another method of reintroducing correlation to color-mapped images is presented in [27]. The authors convert the color-map of an image from a linear list structure to a binary tree structure. This method takes advantage of the tree structure because the tree can be built in a progressive manner. The encoding of the image occurs with the tree building process and thus creates a progressive coding method. Additional compression is gained by using causal context in the encoding procedure. The context-tree based method of [27] is expanded in [28] to "operate on color values instead of binary layers". The expanded methodology of [28] includes creating a context tree of predetermined depth and removing nodes that do not improve compression. The gain in compression performance from context-based methods is countered by the need to store the context tree in the compressed image. The complexity and calculation time for these methods also increase proportionally with the size of the color space. The authors of [28] state that the method is "not expected to work efficiently for images with a large color palette (more than 128 colors), or for small images (with the size less than 100x100 pixels)."

The last method specific to color-mapped images that is under review does not focus on actual color-map index sorting. Instead, the authors of [29] describe a way of using the Euclidean distances of the color values in a map to determine a "pseudo-distance" between indexed pixels. This method allows the colors that are actually the closest to each other to be identified and

reduces the entropy of the differences between pixels. The encoding portion of the method described in [29] is improved by using Arithmetic coding instead of Huffman coding as in [30].

2.4.3 Grayscale Lossless Image Compression Algorithms

2.4.3.1 Algorithms with Decorrelation by Transform

Algorithms for the compression of grayscale images rely heavily on decorrelation. One of the most common decorrelation methods is transformation. Wavelet transforms can be applied to grayscale images for the purpose of decorrelating data prior to encoding [31]. The Discrete Cosine Transform (DCT) is a transform frequently used in compression methodologies and is the foundation of JPEG. The transform itself is completely lossless and "studies have shown that the energy compaction performance of DCT approaches optimality as image correlation approaches one" [31]. Proof of this optimal behavior is that a matrix of a single repeated value that is processed using DCT will result in a matrix with a value in the first position and all other elements will be zeros. [32]

The Karhunen-Loeve Transform (KLT) and Discrete Fourier Transform (DFT) are also reviewed in [31]. Success of decorrelation using DCT, KLT, and DFT is founded on the assumption that the elements of matrices being transformed are highly correlated, have low frequency, and have low spatial energy. The KLT differs from other transforms listed because it uses basis functions that are derived from the image being encoded. A machine learning method for the determination of image bases is presented in [33]. This produces high energy compaction at the expense of high computational costs. This is because the bases must be calculated for every sub-block of the image. It is noted in [31] that "the overall complexity of KLT is significantly higher than the respective DCT and DFT algorithms." The DFT is less complex because it has a fixed basis like DCT. However, the transform procedure of DFT is more complex and requires the image magnitude and phase data to be encoded in the compressed image. The DCT has better energy compaction, handles image boundaries the best, and has a basis transform that can be computed offline so that fewer computations are needed during implementation. In order to enable lossless

compression of images, integer reversible color transformation must be followed by integer reversible wavelet transformation [10].

The primary focus of [11] is the potential for improvement in the context-based lossless image compression coder underlying the multi-resolution method. Three different Reversible Integer Wavelet Transforms, S+P, (2,2+2), and (4,2), are used in the analysis of context-based encoding schemes. The magnitude set that results from these transforms are then context-base coded using three different methods: Ad-Hoc, Adaptive Partial Prediction Matching, and Adaptive Context Tree Weighting. In Ad-Hoc Coding the inter-subband context is captured through the relationship between pixels (i,j) and (floor(i/2),floor(j/2)) and intra-subband context is captured through the four neighboring pixels previously coded at one pixel's difference. Adaptive Partial Prediction Matching uses a statistical-based compression algorithm [11] and "encodes each symbol *s* by an arithmetic coder using its statistic generated from context corresponds to the past *K* encoded symbols" [11]. The final coder examined in [11] is based on a binary tree source model. Despite the power of context-based coding, these methods all remain susceptible to context dilution.

2.4.3.2 Algorithms with Decorrelation by Prediction

Prediction is another common decorrelation method applied in lossless compression. Differential Pulse Code Modulation (DPCM) is a simple predictor that encodes the difference between the predicted value of a pixel and its actual value. DPCM prediction can be based on a single previously encoded pixel as in [13]. In contrast, [3] describes prediction based on more variables. Context-Based, Adaptive, Lossless Image Coding (CALIC) employs more elaborate prediction measures. CALIC's prediction measures are based on an adaptively selected causal context. Predictions are refined by the energy of the errors within an adaptively selected context.

A comparison of methods with similar design using components of adaptive probability assignment is offered in [7]. Algorithms FELICS, SUNSET, and LOCO-I all include the use of context models to generate adaptive probabilities for use in entropy coding. Although entropy can be lowered by using larger contexts there is a negative impact on the cost of the model.

Fuzzy logic can also be applied in the methods of prediction in order to achieve enhanced compression results as discussed in [34]. Fuzzy logic allows a measurement of truth, values between 0 and 1, instead of allowing for only true, 1, or false, 0. Fuzzy logic represents the concept of partial truth and allows the context of an image have more impact than models using only exact matching.

Prediction is used in [14] to determine a successively refinable probability density function. The refinable probability density function is used to create a progression coding scheme. This is an accomplishment that was discounted in [35] which states that "predictive methods don't allow for progressive coding."

Prediction results can also be improved by preprocessing the image prior to prediction as discussed in [36]. The authors employ edge modeling to enhance decorrelation using variance removal. The goal of their method is to obtain lower residual errors during the prediction step.

Predictive decorrelation algorithms often require two passes of the data to achieve decorrelation. The first pass may include initial prediction steps. The second pass may include context modeling of the prediction errors from the first pass. The modeling steps of the second pass produce prediction adjustments and refine prediction values. The error signal generated in the second pass should have lower entropy than the error signal of the first pass. The final error signal generated from two-pass modeling would result in higher compression ratios. Although there is a benefit in two-pass modeling, one must be careful of the cost resulting from the need to store the model in the compressed result.

2.4.4 Multi-Spectral Lossless Image Compression Algorithms

Methods that incorporate the correlation between components are described in [7], [20], [21], and [37]. The Simple Inter-Color Lossless Image Coder (SICLIC) described in [20] includes context from the previous component in the causal neighborhood for predictions. The full component is processed twice. One round of processing consists of intracoding which uses context only from within the component. The other round of processing consists of intercoding which uses context

from the current and the previous component. The result of predictions from a single component, intracoding, is compared to the prediction result from cross component calculations, intercoding, and the best result set is kept. The correlation between components is also used in the binary mode, run length mode, of SICLIC. In run length mode, the other components are checked for similar runs.

The authors of [37] also use intracoding. In [37], the context model includes the previous component during prediction of current components. CALIC-IB, is defined in [21] simply as an inter-band version of CALIC. The authors of [21] highlight that correlation diminishes as saturation increases. They maintain coding efficiency by switching between intercoding and intracoding contexts. The switch is based on an inter-component correlation measure. The ability to switch between intercoding and intracoding accomplishes a reduction of compression bit rates by 20%.

2.4.5 High Dynamic Range Lossless Image Compression Algorithms

High Dynamic Range (HDR) images are multi-spectral images with pixel values that normally range between 16 to 32 bits per pixel. Because HDR images are simply an extension of grayscale images, [38] and [39] propose HDR image compression methods that are based on existing grayscale image compression methods. Neither method introduces any new algorithms, but simply modifies the HDR data to fit current LDR compression methods.

In [39], the HDR image RGB components are transformed to RGBE values. This is a type of custom color space that represents the base image in RGB and includes an exponent value component that is used to restore the base image to the original image. The base image, RGB, is then compressed like a normal grayscale, or Low Dynamic Range (LDR), image. The exponent component, E, “which is found to have spatial coherence”, is processed using lossless coding method Context-based adaptive binary Arithmetic coding (CABAC) [40]. The resulting compression performance is comparable to that of grayscale, LDR, images.

The pixel values of the HDR images evaluated in [38] are floating point numbers rather than integers. These values are converted from the RGB color space to the YCbCo color space. Then the image is partitioned in to non-overlapping 4x4 blocks that are converted to the frequency domain. The floating point values that result from the frequency transform are then cast as integer values for encoding.

2.4.6 Segmentation-Based Lossless Image Compression Algorithms

Segmentation, also known as blocking or partitioning, is a feature of many data compression algorithms. In [41], grayscale images are scanned in roster order and partitioned based on their values. A block will begin with a single pixel. The base value of the block is equivalent to the first pixel. All pixels that follow with a value that is within a set range will be included in the current block. When a pixel is found that has a value that is outside of the range of the current block, the current block is closed and a new block is formed. This produces a series of blocks that are partitioned so that each block has one base value and the other elements of the block are within an established difference.

RGB color space multi-spectral images are processed in [42]. All 3 of the color components are used to determine regions. Beginning with a single pixel as a seed pixel, the neighboring pixel is subtracted from the seed pixel. This produces a three dimensional error value with the differences between each component. If the error value is within a certain range, the neighboring pixel is included in region and the process is repeated on the next nearest neighboring pixel of the seed pixel that has not yet been processed. The steps repeat until no neighboring pixels are within the desired error range. After no valid neighbors are found, the next unprocessed pixel of the image is selected as a seed and the process repeats for that seed. Cartoon images are a special subset of color images that have large regions of homogeneous colors. Cartoon images can be highly compressed by removing the color of a section and representing several pixels using one code entry as described in [43].

2.5 SUMMARY OF OVERVIEW

2.5.1 Lossless v. Lossy Compression Research

Much like their respective compression performances, the amount of research in the area of lossy data compression far exceeds the research in the area of lossless data compression. Approximately 10% of the research in the area of data compression is in the field of lossless or reversible data compression based on a search of libraries such as Google Scholar or the IEEE Digital Xplore. The most widely reviewed lossless image compression methods include CALIC [3] and LOCO-I [44]. CALIC is considered the performance standard and LOCO-I is defined as the lossless image compression industry standard. This leads to a belief that lossless image compression may be a greater challenge than lossy image compression. It is also believed that the field of lossless image compression has room for further contribution.

2.5.2 New Theories and Modifications of Existing Theories

Documentation that is specific to lossless data compression of continuous-tone images can be divided into three major categories: original theories, modifications of existing theories, and evaluative reviews of existing theories. Articles describing new and original theories represent the smallest portion of content. The author of [45] describes what is believed to be a new method that uses serial sorting of unique values combined with a binary tree description of their placement. There seems to be a greater challenge in generating original contributions as evident through the authors of [17]. Although they title their works with “new”, the methods described do not offer any discernible new theories or alterations of the existing Huffman coding methods. Modifications of existing theories dominate the research space. A modification of the CALIC algorithm to use the author’s original design of context selection in the context-based error modeling step for dynamically adjusting the predictor is successfully presented in [18]. A modification in the implementation process of least-squared adaptive prediction is successfully implemented in [19]. Modifications in existing theories are at the forefront of the progress in this field.

2.5.3 Methods of Comparing Algorithms

Most documentation, including documentation of new and modified theories, contains a summary of lossless compression fundamentals. There is also a subset of documentation in the field that makes analysis of lossless compression methods their main purpose. Driven by the response to ISO/JPEG's 1994 solicitation for proposals for an international standard for lossless image compression as discussed in [7], there are many papers that compare today's leading methodologies and previous standards. Algorithms JPEG-LS, CALIC, LOCO-I, and JPEG-2000, UCM are all described and compared in [15], [46], and [7]. The most dominant variable used in the comparison of algorithms is compression ratio and each paper produces similar results. Ease of implementation, compression speed, decompression speed, and computational complexity are acknowledged as key measures of algorithm success, but a common trend in the documentation is that the papers do not include much in depth analysis.

2.5.4 Computational Complexity in Lossless Image Compression Algorithms

Despite the lack of analysis, the importance of implementation ease, compression speed, decompression speed, and computational complexity is best seen in the adoption of a method. Huffman coding has a lower average compression ratio than Arithmetic coding, but Arithmetic coding is not as widely used. A significant reason for less use of Arithmetic coding is the increased computational complexity requirements of the algorithm [15]. Another example of computational complexity's influence on adoption is in ISO/JPEG's 1994 solicitation for proposals for an international standard for lossless image compression. The CALIC algorithm outperforms the LOCO-I algorithm, but the latter has been selected as the foundation of the JPEG-LS standard. LOCO-I was selected due to its ability to produce competitive compression rates at a lower complexity. This point is highlighted in [7], [15], and [46]; yet these papers neglect to define what is considered in the complexity determination. The authors of [7], [15], and [46] also do not describe the actual computational complexity of the discussed algorithms.

In more recent literature, computational complexity is becoming a greater topic of discussion. The analysis seems to be limited to single elements or just the preprocessing elements of more

expansive procedures. In [18], a detailed description of the computational aspect of context modeling is given. The authors of [47] focus on the details of the computational complexity of selected transforms. The proposed algorithm in [48] is compared to JPEG-LS and S+P Transform, but only the computational complexity of S+P Transform and the data folding process are analyzed while the computational complexity of JPEG-LS is not mentioned. In order to properly understand the cost of one method's computation complexity against others, it is necessary to be able to not just generalize complexity, but provide definition and determination.

2.5.5 Observations

One of the first points observed in the literature is that lossless image compression does not perform as well as lossy image compression. The average compression results, based on comparisons conducted in [16], indicate that the rate of an image compressed in a way that is lossy is approximately 10:1 to 50:1. Lossless methods have an average compression rate is between 2:1 and 3:1. In [18], the author points out that "the ratio of compression gains versus computational complexity is diminishing". This is a major challenge to the progression of the field of lossless image compression because more of the new and modified algorithms include processing that increases complexity.

The authors of [15] suggest that lossless compression improvements will require "complex and computationally demanding source models." This can be seen in [19], which describes how least-square based adaptive prediction with optimization conducted at each pixel was previously too computationally complex to adopt into practice. The authors of [19] highlight that the availability of more powerful computers has brought renewed attention to computationally complex methods. Researchers are now able to use computationally complexity procedures in their algorithms because computers now have the power to process them.

CHAPTER 3

REVIEW OF RELATED LITERATURE

3.1 THE EXCLUSIVE-OR OPERATOR IN LOSSLESS COMPRESSION

The use of the exclusive-or operator has been explored in varying ways. However, the purpose of its application remains the same. The exclusive-or operator is used to capture the differences of select data. For many kinds of data, including but not limited to continuous-tone images, there is redundancy between data samples. Applying the exclusive-or operator can reduce the amount of entropy in the data set to be compressed or transform the data set into a more compressible format.

The authors of *CoXoH: Low Cost Energy Efficient Data Compression for Wireless Sensor Nodes using Data Encoding* attempt to use the exclusive-or operator as a data reduction operator in the lossless compression of wireless sensor network data, [5]. The wireless sensor data in consideration in this paper includes only “temperature, humidity, and pressure readings”. The readings are represented as a “string representation of the numerical quantity”. Although the authors do not provide a sample data string, an example based on the authors’ description is as follows: *80,75,14.7*. Each character of the string is converted to the 8-bit ASCII code. The CoXoH algorithm consists of two portions: data reduction and entropy coding.

Table 3-1 : CoXoH Data Sample

STRING	8	0	,	7	5	,	1	4	.	7
ASCII	00111000	00110000	00101100	00110111	00110101	00101100	00110001	00110100	00101110	00110111

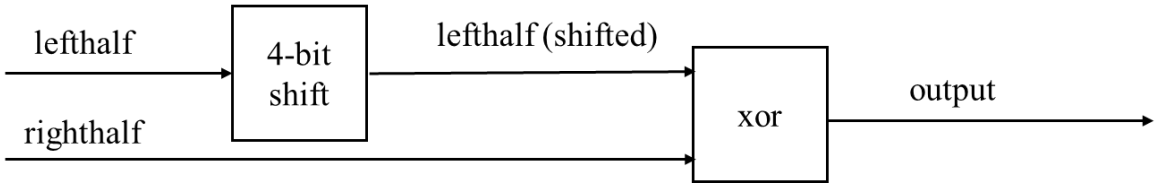

Figure 3-1: CoXoH Data Reduction

The first portion of the CoXoH algorithm is a data reduction process. This process is executed on input data consisting of the 8-bit ASCII codes of original data string. The data reduction process is applied in an iterative manner which reads in two consecutive input values at a time. The first input value is saved as the variable *lefthalf* and the second input value is saved as the variable *righthalf*. The *lefthalf* variable is then bit-shifted by 4 bits to the left. The shifted *lefthalf* variable and the *righthalf* variable are then inputs into the exclusive-or operator. The results of the exclusive-or operator are saved as an entry in the *output* variable and then the data reduction process repeats for the next two variables in the input string. This iterative process will repeat for the length of the input string. Once all of the entries in the input string have been processed the *output* data set will be half the size of the original data set. The data savings are because the *righthalf* and *lefthalf* variables are 16 bits in total and the exclusive-or operator produces an 8 bit output. This introduces a saving of 50%.

The second portion of the CoXoH algorithm is the entropy reduction process. The values of the output string, along with their probabilities, become the input for Huffman coding. Huffman coding is able to introduce additional savings on top of the initial data reduction process.

In the CoXoH algorithm, the exclusive-or operator is used for data reduction. The output of the exclusive-or operator is half the size of the input. Although the data size is reduced, the information that is represented by the output is not equivalent to the information that is input. An example of this explanation is in the simple exclusive-or equation: $1 \oplus 1 = 0$. The output value of 0 represents the status of the difference in the original values. However, the output value does not represent the actual input values. An output value of 0 would be accurate for input set [1, 1] as well as input set [0, 0]. The only way that the output value is able to recover the inputs is if at

least one of the input values is saved also. Because the authors save only the output and do not save any portion of the input, the original values cannot be recovered. Being able to recover the original input values is a key requirement in an algorithm being able to be considered lossless. For this reason the exclusive-or operator is not sufficient as a data reduction tool and CoXoH is not a lossless compression algorithm.

Wireless sensor network data is the focus of *Non-Uniform Entropy Compression for Uniform Energy Distribution in Wireless Sensor Networks*, [4]. The authors of this paper use the exclusive-or operator to decorrelate input data. The input data considered in this paper is temperature data in XML (Extensible Markup Language) format. The data is processed by the XorRLE algorithm consisting of only two portions: decorrelation and entropy encoding.

The XorRLE algorithm begins with the decorrelation portion of the process. The first value in the input string is saved as is without manipulation. The algorithm then applies the exclusive-or operator to the next value of the input along with the prior value in the series. The output of the operation is saved and then the process is repeated for each following input value. The exclusive-or operator will produce zero values where the input series is unchanged.

In XorRLE, entropy coding is completed using a run-length encoder (RLE). The output of the decorrelation process is then input into the RLE. RLE represents data by providing a count of repetitive symbols followed by the symbol counted. In the example data set, AAAAABBBAAAAAAACCCCC, each symbol series is replaced by a count-symbol pair, such as 5A3B7A5C. This reduces a 20 character data set to an 8 character data set. The output of the decorrelation portion of the process will produce long runs of zeros which makes it a good candidate for RLE.

One of the goals of the XorRLE algorithm is to produce compression in an efficiency conscious manner. Wireless Sensor Nodes have very limited resources and require algorithms and processes that are computationally inexpensive. The exclusive-or operator is a low complexity method for identifying differences between data sets. RLE also has low computational

complexity. Both methods, XOR and RLE, meet the energy concerns of wireless sensor networks while providing effective compression results.

The exclusive-or operator is used as a transformation tool in *A Compression Improvement Technique for Low-Power Scan Test Data*, [6]. The authors focus on the creation of a pre-compression technique for test data in system on a chip environments. The test data consists of strings generated by an automatic test pattern generator. The values consist of ones, zeros, and don't-cares as in the example vector "0XX1XX111". Test data is processed through a two part process of bit-filling and transformation.

The first element of the pre-compression technique is bit-filling. The don't-care values of the test data set are assigned to produce minimum transition counts. The original data set is converted from "0XX1XX111" to "000111111", producing only one transition from zero values to one values. The newly filled data is then transformed using the exclusive-or operator. A neighboring bit-wise exclusive-or operator is applied to the test data. This results in data set that has mostly zero values and only ones at the points of bit change, "000100000". The transform reduces the amount of entropy in the data set and results in improved compression performance.

Much like the authors of the previous paper, [6], the authors of *Differential Block Coding of Bilevel Images* [2] focus on the preprocessing step of transformation. Transformation is accomplished through the application of the exclusive-or operator as a differencing operation. The research is one of the few instances where the exclusive-or operator is used to decorrelate images. The data to be processed are bilevel images, black and white facsimile images. The transform is combined with Zeng and Ahmed's (ZA) block coding to create the image differing (ID) block coding algorithm. Only the transform is described by the authors of [2].

The ID block coding algorithm begins with partitioning. The image is divided into blocks. For an *N* row by *M* column image, division will produce *N* blocks of size 1x*M*. After the image is partitioned, transformation is applied to each block. Like the algorithm of *A Compression Improvement Technique for Low-Power Scan Test Data* [6], the neighboring bit-wise exclusive-or operator is applied to each block. This process is able to reduce the number of one values in a

string and create a sparse binary image. Blocks without any one values are not encoded and blocks with one values move forward in processing. The authors have shown that this preprocessing method provides 4 times the compression improvement and the preprocessing method has the ability to be applied to varying image types.

The content of *Lossless Image Compression using Binary Wavelet Transform* [49] describes progressive partitioning binary wavelet-tree coder (PPBWC). This algorithm is designed to provide lossless image compression of continuous-tone gray-level images. The test data compressed includes JPEG standard, JPEG 2000, and a medical image set. These images are processed through the PPBWC algorithm with steps consisting of transformation and entropy encoding. The exclusive-or operator is used to transform data prior to encoding.

The first element of PPBWC is transformation using the Binary Wavelet Transform (BWT). The BWT is a unique wavelet transform that is reversible and does not introduce quantization errors. Limitations of the BWT ensure that the output values of the transform are of the same range as the input. Because the images to be transformed will be processed one bit-plane at a time the selected range includes only one and zero. The root of the BWT is a series of exclusive-or operations that manipulate the original values. Neighboring values are combined through exclusive-or and the results are then re-ordered. The reordering is designed to promote long runs of zeros.

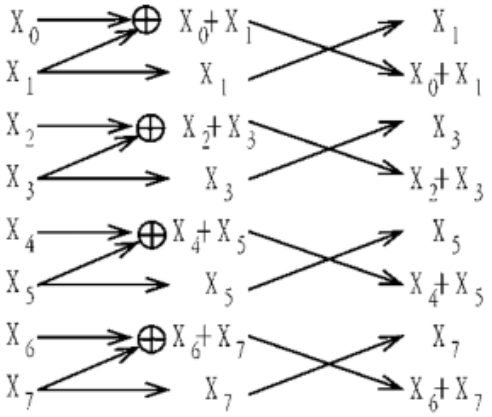

Figure 3-2: XOR element of BWT

After each bit-plane is transformed the process continues with 3 more steps. The next step includes joint bit scanning to get the significant bit stream and the insignificant bit stream. Context modeling and quantization are included in the following step. This ensures that the data is prepared for the final step of Arithmetic coding.

The BWT is a good example of the power of the exclusive-or operator for decorrelation. The performance of the BWT shows that it is possible to strategically apply the simple operator for significant results. The PPBWC uses the exclusive-or operator as an effective decorrelation method that competes with the compression results of top lossless compressors.

The articles discussed above, [5], [4], [6], [2], and [49], help to define the picture of possibility for the exclusive-or operator. The operator excels at decorrelation and transform while retaining its simplicity and low complexity. It has the best potential when partnered with entropy algorithms that are modified to match the decorrelated input data. Further study includes the exploration of the exclusive-or operator in decorrelation applications and modifications of entropy algorithms to better match the decorrelated image data.

3.2 TOP PERFORMING LOSSLESS COMPRESSION ALGORITHMS

The current standard of lossless image compression does not employ the exclusive-or operator. Instead, the field of lossless image compression has been shaped by a few key lossless compression algorithms. An algorithm that has been used as a benchmark for performance is Context-Based, Adaptive, Lossless Image Compression, CALIC. This algorithm is frequently compared with the Low Complexity Lossless Compression for Images, LOCO-I, algorithm. The LOCO-I algorithm is the foundation of the widely used JPEG-LS image file format. Another image file format with great popularity is the PNG image file format. This file format is based on the deflate data compression algorithm. The deflate algorithm is also the foundation of the ZIP file format. In this study, the CALIC, LOCO-I, and deflate algorithms represent the standard for the field of lossless image compression.

The CALIC algorithm represents the pinnacle of compression performance. It is one of the highest compression performers available. The CALIC algorithm produces an average compression ratio of 2.68:1 on grayscale continuous-tone images. This great compression performance is due to the focus the authors place on preprocessing. The preprocessing steps of the CALIC algorithm rely heavily on statistical modeling of the image. Large counts of states are used to create an adaptive, self-correcting, nonlinear predictor. States are also known as modeling contexts and are a representation of the values that surround a pixel. The efficiency of the algorithm is due to well-designed modeling contexts. The modeling contexts are used to manipulate and refine predicted pixel values. These predicted pixel values are subtracted from actual pixel values to capture the difference as an error signal. The better the predictor the more the error signal is biased to zero. This improves entropy and thus the compression performance of the entropy coding of the error signal.

The CALIC algorithm was designed in an efficiency conscious manner so that it could have practical application in both hardware and software implementations. Other algorithms such as Universal Context Modeling, UCM, use context modeling to refine prediction values, but are unable to find the balance between performance and complexity that CALIC has done. The designers of CALIC describe their algorithm as “conceptually more elaborate”, but “algorithmically quite simple”.

The simplicity of the CALIC algorithm is seen in the general design. CALIC uses a sequential coding scheme that is executed in raster scan order, reading pixels from row by row from left to right and top to bottom, with one pass through the image. Prediction and context modeling are based only on previous 2 rows of pixels. The algorithm also offers two modes of operation: binary and continuous-tone. The process of switching between these modes is automated within the coding process using the state of the pixel being processed. Prior to coding the pixel being processed, the values of 6 neighboring pixels, I_{ww}, I_w, I_{nw}, I_n, I_{ne}, and I_{nn} as mapped in Figure 3-3 are checked. If the neighboring pixels only have 2 different values, binary mode is triggered. Otherwise, continuous-tone mode is triggered.

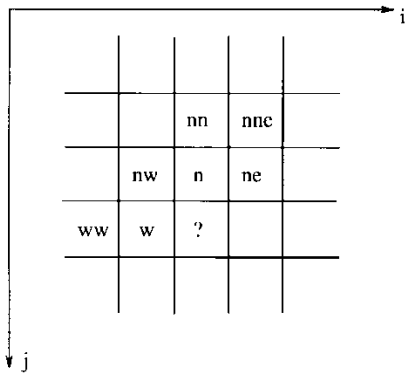

Figure 3-3: CALIC Context

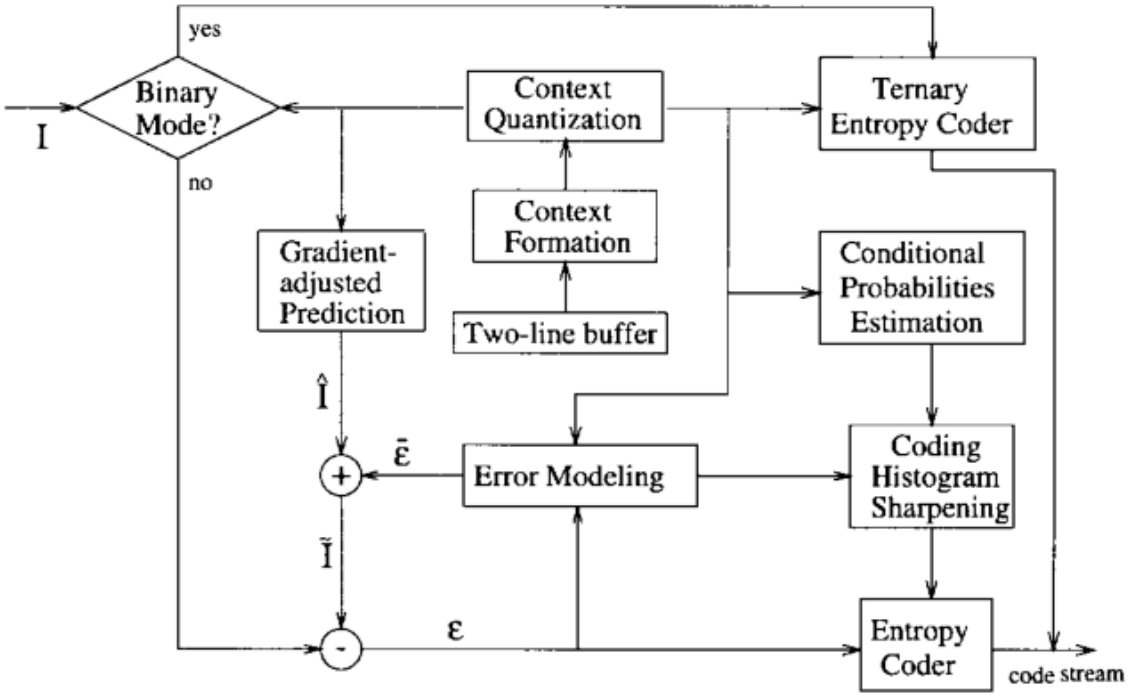

Figure 3-4: CALIC Algorithm Components

The continuous-tone mode of the CALIC algorithm consists of 4 components. The process begins with a gradient-adjusted prediction (GAP) step that predicts the expected value of a pixel based on its context model. The next step, context selection and quantization, identifies the error energy of a pixel based on its context model and the error value of its neighbor. Context modeling of prediction errors is in the following step. In this step a texture value is calculated

from neighboring pixels and the combination of texture and error energy form a context model for prediction errors. The average error for the identified context is combined with the GAP value of the current pixel to create an adjusted GAP value. The difference between the adjusted GAP value and the actual value are then used to generate an error feedback loop to further improve the GAP value. The last step of the process is the entropy coding of prediction errors.

Gradient-adjusted prediction is based on the context of the current pixel, I. The values of 7 neighboring pixels, I_n, I_w, I_{ne}, I_{nw}, I_{nn}, I_{ww}, and I_{nne} as described by Figure 3-3, are used to calculate the expected value of I. The measures of horizontal directional change and vertical directional change near pixel I are calculated and captured as d_h and d_v respectively. The change amount is a combination of neighboring pixel differences along the directional axis. The directional change amounts of d_h and d_v are good indications of directional edges and their intensities. The values of d_h and d_v are used in the prediction of the value of I, noted as $\hat{I}$. The prediction process is described in the procedure in formula (4.

$$d_h = |I_w - I_{ww}| + |I_n - I_{nw}| + |I_n - I_{ne}| \quad (2)$$

$$d_v = |I_w - I_{nw}| + |I_n - I_{nn}| + |I_{ne} - I_{nne}| \quad (3)$$

$$\begin{aligned}
&IF(d_v - d_h > 80)\{sharp\ horizontal\ edge\}\hat{I}[i,j] = I_w \\
&ELSE\ IF(d_v - d_h > -80)\{sharp\ vertical\ edge\}\hat{I}[i,j] = I_n \\
&ELSE\{ \\
&\quad \hat{I}[i,j] = (I_w + I_n)/2 + (I_{ne} + I_{nw})/4; \\
&\quad IF(d_v - d_h > 32)\{horizontal\ edge\}\hat{I}[i,j] = \left(\hat{I}[i,j] + I_w\right)/2 \\
&\quad ELSE\ IF(d_v - d_h > 8)\{weak\ horizontal\ edge\}\hat{I}[i,j] = \left(3\hat{I}[i,j] + I_w\right)/4 \\
&\quad ELSE\ IF(d_v - d_h < -32)\{vertical\ edge\}\hat{I}[i,j] = \left(\hat{I}[i,j] + I_n\right)/2 \\
&\quad ELSE\ IF(d_v - d_h < -8)\{weak\ vertical\ edge\}\hat{I}[i,j] = \left(3\hat{I}[i,j] + I_w\right)/4 \\
&\}
\end{aligned} \quad (4)$$

GAP removes a significant amount of correlation from the input image. The errors, e, generated by taking the difference between the predicted value and the actual value, $e = I - \hat{I}$, will result in a data set with lower entropy than the original values. The correlation that remains in the image is now between the variance of prediction errors and the texture surrounding pixels. To begin

decorrelating the relationship between error and texture, the authors create a way of estimating the error energy, Δ, around a pixel. The values of horizontal and vertical directional change, d_h and d_v, are combined with the amount of error found at previous pixel I_w. The formula below generates a quantifiable measure of error that can be used in the decorrelation of error and texture.

$$\Delta = d_h + d_v + 2 * |e_w| \quad (5)$$

$$e_w = I[i-1, j] - \hat{I}[i-1, j] \quad (6)$$

Now that error energy is captured, the next portion of the relationship to quantify is texture. Because errors change in relation to texture, the texture around a pixel is a major factor in error prediction. To quantify texture, the authors define a texture context, C, that consists of 6 neighboring pixels and 2 neighboring pixel combinations. The values of each element in C are then quantized into 0 if the value is greater than or equal to the predicted value $\hat{I}[i,j]$ or quantized to 1 otherwise. This quantization generates an 8 bit texture identifier, B, which is used in the decorrelation of error and texture.

$$C = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\} \quad (7)$$
$$= \{I_n, I_w, I_{nw}, I_{ne}, I_{nn}, I_{ww}, 2I_n - I_{nn}, 2I_w - I_{ww}\}$$

The error energy captured in Δ and the texture captured in B are combined to create a set of possible compound contexts, $C(\delta,\beta)$. For each compound context that occurs, a count of the occurrence of that compound context is incremented. The value of the error, e, within the compound context is added to a running total of all errors that have occurred within that compound context. The average error, $\bar{e}(\delta,\beta)$, is sum of all error within the compound context divided by the count of occurrences of the compound context.

Average error, $\bar{e}(\delta,\beta)$, is used to refine the GAP value of a pixel, $\hat{I}$, and to create an error feedback loop. A refined prediction, $\tilde{I}$, of the current pixel is created by adding the average error

$$\tilde{I} = \hat{I} + \bar{\epsilon}(\delta, \beta) \quad (8)$$

at the context to the GAP value of the current pixel. Rather than adjust the GAP, the new prediction error, $\epsilon = I - \tilde{I}$, is considered instead of the original GAP error, $e = I - \hat{I}$. The new prediction error, ϵ, is then used in another round of context modeling. Using two stages of adaptive prediction further improves the prediction of Ĩ.

The information captured in error modeling also includes the sign of the error. This allows the decoder to employ sign flipping. Prior to encoding the error value, ϵ, the encoder can check the average error, $\bar{\epsilon}$, at the compound context of the current pixel. If the average estimation error is less than 0, $\bar{\epsilon}(\delta, \beta) < 0$, -$\epsilon$ is encoded instead of ϵ. Significant decorrelation of the original image values is produced using the above described adaptive prediction schemes. The resulting error signal, ϵ, will have reduced entropy and a data set biased to 0. The error signal is then entropy coded using an Arithmetic coder.

The binary mode of the CALIC algorithm is also designed to be efficiency conscious. As previously describe, the mode is triggered automatically when the neighboring pixels have only 2 distinct pixel values. The first pixel value option, s_1, is set to the value of the left neighboring pixel. The other pixel value option, if one exists, is set to s_2. If the current pixel is equal to s_1, a 0 is encoded. If the current pixel is equal to s_2, a 1 is encoded. When neither value is found a 2 is encoded and CALIC's mode of operations is switched to continuous-tone mode.

Context modeling in binary mode is used during the encoding process. The 6 neighboring pixels of the current pixel form the context under consideration. Each value in the context is then converted to a 0 if equal to s_1 or 1 if equal to s_2. The context is used to generate conditional probabilities of the encoded value based on neighboring values, $p(T|B)$. These probabilities become an input for a ternary, adaptive Arithmetic coder.

$$C\{x_0, x_1, x_2, x_3, x_4, x_5, \} = \{I_w, I_n, I_{nw}, I_{ne}, I_{ww}, I_{nn}\} \quad (9)$$

The adaptive nature of CALIC is a primary factor in the strong performance of the algorithm. Key elements in the algorithm include using context to make value predictions and then refining the

predictions through error feedback. Although the authors designed the algorithm to have practical application, the complexity of the algorithm is still greater than its competition. The LOCO-I algorithm is an example of an algorithm that has similar compression performance, but lower complexity. These compression and complexity measures are the reason that the LOCO-I algorithm was selected as the lossless JPEG standard rather than CALIC.

The LOCO-I algorithm was also designed to be efficiency conscious. The authors' goals include combining the "simplicity of Huffman coding with the compression potential of context models." Essentially, the LOCO-I algorithm is targeted as a low complexity version of universal context modeling. The complexity of the algorithm is reduced because the authors use a fixed context model, use adaptively chosen encoding parameters, and avoid the use of the Arithmetic coder.

The structure of the LOCO-I algorithm is also predictor-modeler-coder like CALIC. LOCO-I processes images in one pass by raster scan order using a sequential coding scheme. The context of a given pixel and any predictions made are based on the previous row of data. Because of this LOCO-I requires on 1 row of the image to be stored as opposed to CALIC which requires 2 rows. LOCO-I offers 2 modes of operation: run mode and continuous-tone. The context of the current pixel is used to automatically determine which mode the algorithm should enter during the encoding process. When examining the context of a pixel, the algorithm examines the values of the 4 nearest processed neighbors, *a*, *b*, *c*, and *d*. If all of the neighboring values are equal, run mode is triggered. Otherwise, continuous-tone mode triggered.

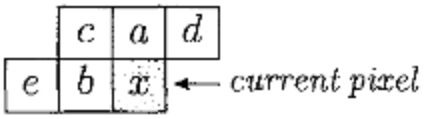

Figure 3-5: LOCO-I Context

There are 3 components in the continuous-tone mode of the LOCO-I algorithm. The initial step of the algorithm is prediction. In this step as simplified GAP is used to generate the expected value of the current pixel based on its context. The next step is context modeling of prediction errors.

This step includes context quantization, context modeling of prediction errors, and bias cancellation. The last step consists of entropy coding the prediction errors.

The prediction step of the LOCO-I algorithm can be considered as a simplified version of the GAP. Rather than create a separate variable to capture horizontal and vertical directional values, a simplified edge detector is included in the conditions of the predictor. The pixels that form the context of the value to be predicted are *a*, *b*, *c*, and *d*. These pixels are the input of the Median

Edge Detection predictor of formula (10. This predictor will choose x=b when there is a horizontal edge above x. When there is a vertical edge to the left of x, x=a. When no edge is detected, the value of x is generated from a combination of *a*, *b*, and *c*.

$$x = \begin{cases} \min(a,b) & if\ c \geq \max(a,b) \\ \max(a,b) & if\ c \leq \min(a,b) \\ a+b-c & otherwise \end{cases} \qquad (10)$$

Context modeling in the LOCO-I algorithm is also simplified. The context of pixel x is calculated using the differences of neighboring pixels, g_1=*d*-*a*, g_2=*a*-*c*, and g_3=*c*-*b*. The gradient values of the context, $c = [g_1, g_2, g_3]$, act as a descriptor of the activity around a pixel. The gradient values are then quantized to reduce the size of the context model set. The each of the gradient values, g_1, g_2, and g_3, are quantized into regions of equal probability. The authors have defined the ranges for quantization as {0}, {1,2}, {3,4,5,6}, {7,8,..,14}, and {15, 16, .., 255}. The quantization produces a set of 3 values that can range between -5 and 5.

$$Prob\{e_{i+1} = \Delta | C_i = [q_1, q_2, q_3, q_4]\} \qquad (11)$$
$$= Prob\{e_{i+1} = -\Delta | C_i = [-q_1, -q_2, -q_3, -q_4]\}$$

An additional round of simplification is done by merging contexts of opposite signs. This is possible because the probability of the negative context is equal to the probability of the positive context. Merging is possible because LOCO-I also leverages sign flipping. Although the absolute value of the error value is encoded, the decoder is still able to determine the sign needed. The decoder is able to check the sign of the first non-zero error amount of the same context. If the value of the error is negative, the decoder negates the decoded value.

After the context of the current pixel is determined, the algorithm can estimate the bias in the original prediction value. This step is similar to the error feedback conducted in CALIC. However, the LOCO-I differs in how the average error of a given context is calculated. A sum of all of the errors at the given context is calculated and the count of all occurrences of the given context is kept. Rather than use the division operator and dividing the sum by the count, the LOCO-I algorithm employs the procedure in figure (12. This procedure creates a division-free process that reduces the number of additions and subtractions used to determine the average error. The average error represents the estimation bias of the prediction. The estimation bias is added to the original prediction value to refine the original prediction value. The difference between the refined prediction and the actual pixel value create the error signal to be encoded. The error signal is encoded using Golomb-Rice coding.

$$
\begin{aligned}
&B = B + \epsilon; /* \, accumulate \; prediction \; residual \, */ \\
&N = N + 1; /* \, update \; occurrence \; counter \, */ \\
&/* \, update \; correction \; value \; and \; shift \; statistics \, */ \\
&if(B \leq -N)\{ \\
&\qquad C = C - 1; B = B + N; \\
&\qquad if(B \leq -N) \; B = -N + 1; \\
&\} \\
&else \; if \; (B > 0) \; \{ \\
&\qquad C = C + 1; B = B - N; \\
&\qquad if \; (B > 0) B = 0; \\
&\}
\end{aligned}
\qquad (12)
$$

The run mode of LOCO-I is also simplified. This mode is automatically engaged when neighboring pixels, *a*, *b*, *c*, and *d* are all equal. In this mode, a repetition of the value in pixel *b* is expected. The run length, *r*, of this repetition is considered rather than the pixel value itself. When a value not equal to the pixel value of *b* is encountered, the error, *e*=*x*-*b*, is encoded and the continuous-tone mode is triggered. The encoder for run mode is also Golomb-Rice coding. Run lengths are ranked by frequency. The rank of the run length is then encoded rather than the run length value, *r*.

Both the LOCO-I and the CALIC algorithms are examples of context modeling algorithms. They employ adaptive prediction methods that are enhanced using feedback from errors. The CALIC algorithm is recognized as top performing compressor. The LOCO-I algorithm has comparable,

although slightly lower, compression performance, but is the foundation of one of the most popular compressed image file formats. JPEG-LS is a common image file format that is built using the LOCO-I algorithm. Other popular file formats include PNG and ZIP. The PNG file format is designed for images and the ZIP file format is designed for general data formats. Both of these file formats, PNG and ZIP, are built from the deflate compression algorithm.

The deflate compression algorithm is a combination of both dictionary coding and entropy coding [50]. The dictionary coding methods are based on the LZ77 and entropy coding is accomplished through Huffman coding. Input data is divided into blocks through the encoding process. Each compressed block consists of data with two possible element types. The first element type is the literal value of a series of data bytes. The other element type is a pointer generated using dictionary coding. The pointer maps the data bytes to be encoded to a series of data bytes that have already been encoded. Both element types, literals and pointers, are then encoded using Huffman coding. This algorithm has the potential to introduce two rounds of compression into the encoding process.

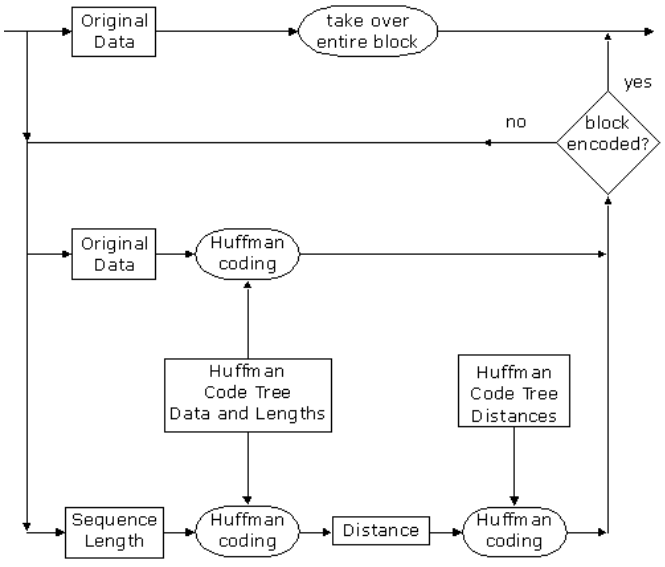

Figure 3-6: Deflate Compression Options

The elements of a compressed data block are determined by the encoder during the encoding process. Within this encoding process there are 3 compression options. The first option is no

compression. The second option is compression using a static Huffman coding tree. The last option is compression using a Huffman coding tree targeted to the data within the given block. The selection of available options is adjustable and controlled by the encoder.

The first option, no compression, is selected when compression is not possible on the selected data set. This happens when the original data set has already been compressed prior to input or if the original data has minimal repetition. The deflate encoder can compress a data set and then check if the bit count of the result is greater than the bit count of original data set. If the result is greater than the input, the encoder aborts the compression and encodes the original data set.

In options two and three, the process of dictionary coding is used to encode input data. Data that has been previously encoded is examined to find determine if the current input has already been captured. If a match is found, a pointer is created. Pointers contain length and backward distance pairs. Length represents the bit count of the data that should be selected. Backward distance indicates the starting index of matching data as a count of bits from the current point. When no match is found, the literal value is encoded rather than the pointer.

The representation of elements in the deflate algorithm is accomplished with predefined alphabets of numeric values. Literal values can be any value between 0 and 255. The length portion of the pointer element can be any value between 3 and 258. Representation of literal and length values are merged into a single alphabet of 0 to 285. Within the merged range, alphabet values 0 to 255 indicate their literal values and the alphabet value 256 indicates the end of the data block. Alphabet values 257 to 285 can be combined with extra bits as needed to indicate length codes 3 to 258 as listed in Table 3-2. The backward distance portion of the pointer element can be any value between 1 and 32,768. Representation of the distance values is accomplished with a separate alphabet of values that range between 0 and 29 with extra bits as need to indicate distance values 1-32,768 as listed in Table 3-3.

Table 3-2: Deflate Length Codes

Code	Extra Bits	Length(s)	Code	Extra Bits	Length(s)	Code	Extra Bits	Length(s)
257	0	3	267	1	15,16	277	4	67-82

258	0	4	268	1	17,18	278	4	83-98
259	0	5	269	2	19-22	279	4	99-114
260	0	6	270	2	23-26	280	4	115-130
261	0	7	271	2	27-30	281	5	131-162
262	0	8	272	2	31-34	282	5	163-194
263	0	9	273	3	35-42	283	5	195-226
264	0	10	274	3	43-50	284	5	227-257
265	1	11,12	275	3	51-58	285	0	258
266	1	13,14	276	3	59-66			

Table 3-3: Deflate Distance Codes

Code	Extra Bits	Length(s)	Code	Extra Bits	Length(s)	Code	Extra Bits	Length(s)
0	0	1	10	4	33-48	20	9	1025-1536
1	0	2	11	4	49-64	21	9	1537-2048
2	0	3	12	5	65-96	22	10	2049-3072
3	0	4	13	5	97-128	23	10	3073-4096
4	1	5,6	14	6	129-192	24	11	4097-6144
5	1	7,8	15	6	193-256	25	11	6145-8192
6	2	9-12	16	7	257-384	26	12	8193-12288
7	2	13-16	17	7	385-512	27	12	12289-16384
8	3	17-24	18	8	513-768	28	13	16385 – 24576
9	3	25-32	19	8	769-1024	29	13	24577-32768

The difference between options two and three of the deflate algorithm is in the Huffman coder. In option two of the deflate algorithm, a fixed set of Huffman codes is used when encoding the merged alphabet of literal and length elements. Because the Huffman code is fixed, the information for the Huffman code does not have to be included in the encoded data block. The distance alphabet is not Huffman coded. It is represented by the 5 bit representation of values 0-31. Option two is preferred when the use of a custom Huffman does not introduce compression with a bit count less than compression using the fixed Huffman code.

Option three of the deflate algorithm involves the use of a custom Huffman code. A separate Huffman code table is included for the literal and length merged alphabet and the distance alphabet. Code tables are represented using code length sequences. The deflate algorithm is able to regenerate Huffman trees using code length sequences. In order to increase compactness, the code length sequences are also Huffman coded. The code length alphabet

includes 0 to 15 as the actual code lengths. The alphabet is expanded to include 16 as an instruction to copy the previous value 3 to 6 times. Alphabet entries 17 and 18 are instructions to copy the value 0 for 3 to 10 times and 10 to 138 times respectively. The encoded Huffman tree is included in the compressed data block preceding the compressed data.

The dictionary based coding of the deflate algorithm introduces compression by leveraging repetition within the data set. The more repetitious the data source, the greater the compression performance. The algorithm replaces values that have already been processed by the encoder with pointers to the previously encoded data. Multiple data values can be represented using a single pointer.

The deflate algorithm is very effective in a broad range of applications. It was designed as the foundation of the ZIP file format. This format allows compression of a diverse set of input. Performance on text data is strong because text has a limited alphabet and repetitive patterns. Binary images perform well in the deflate algorithm for the same reason. Although the deflate algorithm is effective on image data, the PNG file format is able to produce greater compression using the deflate algorithm.

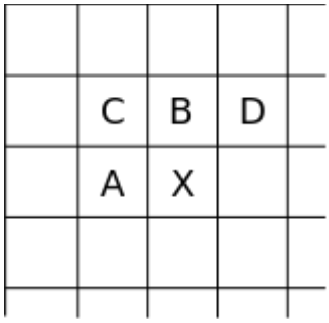

Figure 3-7: PNG Context

The improved performance of PNG is due to the preprocessing steps that are employed prior to deflate encoding. Preprocessing consists of applying a filter to the image in raster-scan order. For each pixel, *X*, in the image, the value of *X* is predicted using one of the filter options listed in Table 3-4 [51]. Along with the option to not include filtering, PNG allows the application of 4 types of filters: Sub, Up, Average, and Paeth. The values used in the filters are extracted from

neighboring, previously encoded pixels in the image. The value that results from applying the filter is then subtracted from the actual value. This process generates an error signal that is more biased to zero than the original signal. The preprocessing steps of the PNG file format are very similar to the preprocessing steps of CALIC and LOCO-I.

Table 3-4 : PNG Predictor

Type byte	Filter name	Predicted value
0	None	Zero (so that the raw byte value passes through unaltered)
1	Sub	Byte A (to the left)
2	Up	Byte B (above)
3	Average	Mean of bytes A and B, rounded down
4	Paeth	A, B, or C, whichever is closest to $p = A + B - C$

PNG, CALIC, and LOCO-I all improve the compression performance of their respective encoders by decorrelating image data through prediction. Predicted values incorporate information from the context of each pixel. The information provided by these neighboring pixels improves prediction values and reduces the amount of uncertainty in the pixel being processed. The error signal that is produced by subtracting predictions from actual values will be biased to zero. The reduction in uncertainty is translated as a reduction in entropy. The reduced entropy generates improvements in the encoding process. The prediction methods used in CALIC, LOCO-I, and PNG will be explored in conjunction with the use of the exclusive-or operator to improve decorrelation. Information provided by the context of pixels will also be explored to improve the encoding process. The methodologies of top performing algorithms will be incorporated into research to potentially generate further compression improvements.

The “Algorithm Components” chapter provides detailed information about the dissertation algorithm. Each possible component of the algorithm is described. Background information, processes, and performance of each component are reviewed. In the “Algorithm Design” chapter the best combination of components are identified and used to form the final dissertation algorithm. The compression performance of the final algorithm when applied to natural, medical, and synthetic images is reviewed in the chapter. The “Algorithm Comparison” chapter conducts a

comparison between the dissertation algorithm and three of the most popular lossless image compression algorithms currently available: CALIC, LOCO-I, and DEFLATE. The chapters that follow provide an understanding of the design, performance, and application of the dissertation algorithm.

CHAPTER 4

ALGORITHM COMPONENTS

The goal of this dissertation is to leverage the exclusive-or Boolean logic-operator (XOR) in the context of lossless image compression. Initial research consisted of applying the exclusive-or operator as a difference coder to images prior to entropy coding using Huffman compression. Through this dissertation, the initial research is extended to explore additional preprocessing steps. The extended research also includes exploration of dictionary coding. The preprocessing components of the algorithm are designed to restructure data so that the performance of the XOR difference coder and the performance of the dictionary coder will be enhanced.

In this chapter, each of the algorithm components is presented. A discussion of the purpose of the component is provided. The methods used in each component are detailed and explained. Relationships between components will also be explained so that the influence of a component on the components that follow is highlighted. The information described in this chapter provides a foundation for understanding the algorithm design and experiments in the next chapters.

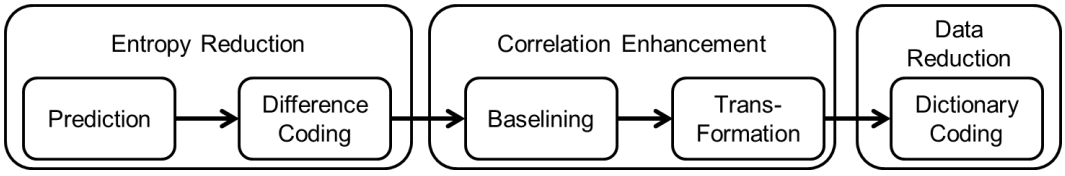

Figure 4-1: Dissertation Algorithm Components

Five algorithm components are considered in the dissertation: difference coding, dictionary coding, prediction, baselining, and transformation. Multiple combinations of the algorithm components are examined. The primary components of the algorithm are difference coding and dictionary coding. These two components are included in all combinations. Difference coding is conducted using the exclusive-or operator. The difference coding component captures the

difference between neighboring pixels. Dictionary coding is conducted using several variations of the Lempel-Ziv-Welch dictionary coder. The 3 remaining components can be classified as preprocessing components. Prediction, baselining, and transformation can be applied prior to difference coding and dictionary coding in an effort to improve performance.

Difference coding is the first of the primary components in the algorithm. The performance of this component can be enhanced with the prediction component. The prediction component generates a data set that is closer in value to the original image data set. The use of the prediction component in conjunction with the difference coding component generates a difference data set with lower entropy than the difference data set generated by the difference coding component alone.

The output of the first primary component, difference coding, is the input for the second primary component, dictionary coding. Prior to dictionary coding, it is possible to improve the data set so that it performs better in dictionary coding. The baselining and transformation components are used to improve the correlation of the difference coded data set. The baselining component is used to transform the difference data set from bit-plane distance to adjusted-linear distance. This adjustment helps to restore correlation in the data set. The transformation component uses the Burrows-Wheeler Transform (BWT) to improve compression by generating long runs of values and patterns in data. The BWT introduces additional correlation between the values of the data set. The baselining and transformation components can be used in the dissertation algorithm because they are both reversible and do not require the storage of additional data for lossless data recovery. The correlation restored by the baselining and transformation components improves the compressibility of data by the dictionary coding component.

Applying each of the algorithm components to an image in the sequence described in Figure 4-1 has the potential to introduce greater compression than only using difference coding and dictionary coding. The primary focus of the prediction component is accuracy. An accurate predictor can reduce the variance and entropy of the difference data set generated by the difference coding component. The baselining component modifies the representation of

difference data set values. The modification of data set values is done in a way that reintroduces some of the correlation that is lost by the use of the XOR difference coder. The correlated representation of the difference data set may have less entropy and increased restoration of data patterns than the original data set. When image data consists of many patterns, the BWT may be able to transform data into long runs of symbols which can introduce further variance reduction. The repetition from symbol runs and symbol patterns in the transformed data set is more easily compressed by dictionary coders. The individual components of the algorithm work together to refine the original image data so that is in a format that is more easily compressible using a dictionary coder.

4.1 PREDICTION

4.1.1 Background on Prediction

Prediction is the first component of the dissertation algorithm. The prediction component generates the values that are compared to the actual image values when difference coding. The difference between the predicted values and the actual values forms the difference data set. The accuracy of the predictor has a direct impact on the values of the difference data set. Predictors with accurate performance will generate difference data sets that are highly biased to zero. Signals biased to zero will have lower entropy than the original image and allow for increased compression.

The prediction component is not balanced between encoding and decoding. The full data set of the original image is available to the encoder. The decoder does not have access to the same amount of image data. The decoder is only able to access image data that has already been decoded. To ensure the reversibility of the dissertation algorithm, the encoder must make its predictions using only the data that will be available to the decoder. This limitation requires that the predictor determine the most likely value of a pixel based on the information that has already been processed. In this study, images are processed in raster order which is left to right and top to bottom. For a pixel, X, located in row i and column j, the pixels in rows 1:i-1 and columns 1:j-1

of row *i* are available. Pixels closest to the pixel being predicted, the target pixel, are typically used as the context for prediction calculations.

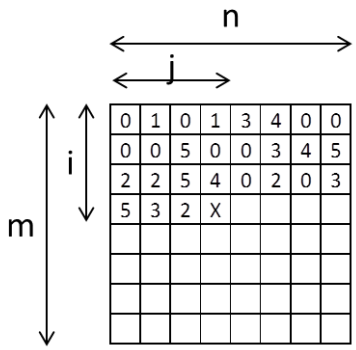

Figure 4-2: Context of Target Pixel

A major challenge in the prediction component is determining the context pixels and related processes that will be the best predictors of a target pixel in all conditions. In all of the reviewed literature, the context pixels vary within a 3 pixel radial distance of the target pixel. The processes applied to the context pixels also vary. Predictors may analyze the amount of change in pixel values, change energy, around the target pixel in multiple ways. There are also predictors that use the same context pixels and change energy estimators, but use differing formulas to predict the target pixel values. The variance in predictor design results in variance in predictor performance. In this study, the performance of a predictor is measured through the mean absolute error of the predicted data set and the original data set. The best performing predictor will generate the best possible difference data set.

There are 10 different predictors considered in this study. The first 4 predictors are simple and only use 1 pixel for context. The fifth predictor is the Median Edge Detection (MED) predictor and is based on a 3 pixel context. The MED is a common predictor that is currently used in the LOCO-I algorithm that is the foundation for JPEG-LS. The sixth and seventh predictors are Gradient Edge Detection (GED) predictors which are based on a 5 pixel context. The eighth predictor is the Gradient Adjusted Predictor (GAP) which is based on a 7 pixel context. The last 2 predictors are built on gradient based selection of prediction values. Accurate Gradient Selection

Predictor (AGSP) uses a 9 pixel context and Gradient Based Selection and Weighting (GBSW) uses a 10 pixel context. The combination of context, change energy capture methods, and prediction formulas is unique for each predictor. Each predictor is analyzed and compared in the following sections.

4.1.2 Static Linear Predictors

Continuous-tone images typically have a high amount of correlation between pixels. Pixel values will vary with their neighboring pixel values. This relationship allows for the first and simplest of the predictors, static linear predictors. The target pixel will be predicted to be equal to its neighbor. When processing the image in raster order there are 4 immediate neighboring pixels available: pixel *N*, pixel *W*, pixel *NW*, and pixel *NE*. Four predictors are made that use the four neighboring pixels respectively as the predicted values. Change energy around the pixel is not considered in this predictor.

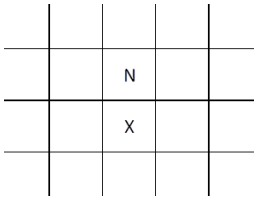

Figure 4-3: Linear North Target Pixel Context

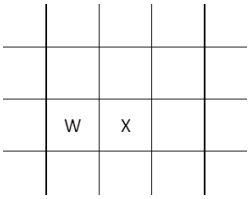

Figure 4-4: Linear West Target Pixel Context

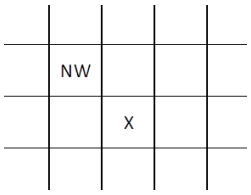

Figure 4-5: Linear Northwest Target Pixel Context

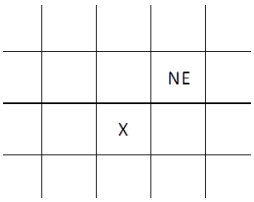

Figure 4-6: Linear Northeast Target Pixel Context

4.1.3 Median Edge Detection

The next predictor considered, the MED predictor, uses 3 of the 4 available neighboring pixel values in prediction calculations. These 3 pixels, pixel *N*, pixel *W*, and pixel *NW*, form the context of the target pixel. The method of measuring the change energy that surrounds the target pixel is

built into the prediction process. When the value of pixel *NW* is greater than or equal to the values of both the *N* and *W* pixels, the target pixel is assumed to be a member of a low edge as in Figure 4-8. The target pixel is set to the minimum value of the *N* and *W* pixels. When the value of the pixel *NW* is lower than or equal to the values of both the *N* and *W* pixels, the target pixel is assumed to be a member of a high edge as in Figure 4-9. The predicted value is set to the maximum value of the *N* and *W* pixels. If the value of pixel *NW* is in between the values of the *N* and *W* pixels, no edge is detected at the target pixel. The value of the target pixel is set to be a combination of the values of the *N, W,* and *NW* pixels

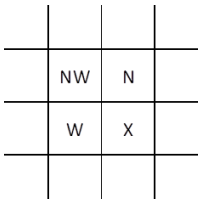

Figure 4-7: Median Edge Detection Target Pixel Context

$$\hat{X} = \begin{cases} \min(W,N), & if\ NW \geq \max(W,N) \\ \max(W,N), & if\ NW \leq \min(W,N) \\ W + N - NW, & otherwise \end{cases} \quad (13)$$

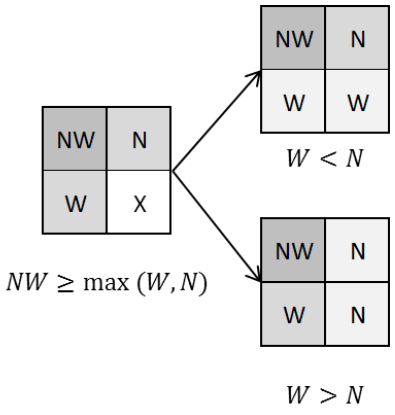

Figure 4-8: Median Edge Detection Low Edge

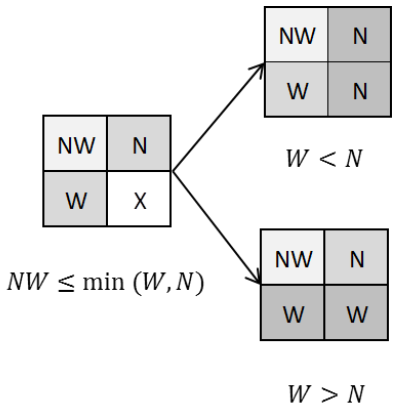

Figure 4-9: Median Edge Detection High Edge

4.1.4 Gradient Edge Detection

The author of *Lossless Compression of Medical Images based on Gradient Edge Detection,* [52], presents a predictor designed to match the performance of the GAP while matching the simplicity of the MED predictor. Pixel context is expanded in the GED predictor. The 5 pixels that form the context are pixels *N*, *W*, *NW*, *NN*, and *WW*. The additional pixels in the context allow the GED predictor to capture change energy around the target pixel. The measure of change in the vertical direction and the measure of change in the horizontal direction are calculated using formulas (14 and (15, respectively. A change threshold, *T*, is defaulted to be a value of 8. If the difference in horizontal and vertical change energy is greater than the threshold, a horizontal edge is detected at the target pixel. The value of the target pixel is predicted to be equal to the value of pixel *W*, as given in the if-condition of formula (16. When the difference in horizontal and vertical change energy is less than the negative of the threshold, a vertical edge is detected at the target pixel. The value of the target pixel is predicted to be equal to the value of pixel *N*, as given in the else-if condition of formula (16.

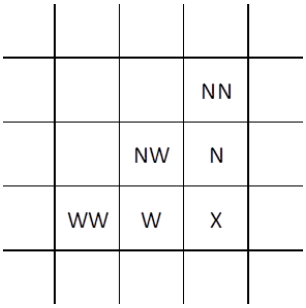

Figure 4-10: Gradient Edge Detection Target Pixel Context

$$g_v = |NW - W| + |NN - N| \quad (14)$$

$$g_h = |WW - W| + |NW - N| \quad (15)$$

$$\begin{array}{c} if\ g_v - g_h > T, P = W \\ elseif\ g_v - g_h > -T, P = N \\ else\ P \end{array} \quad (16)$$

There are 2 formulas available for predicting the value of the target pixel when an edge is not detected. The first formula is the same formula used in the MED predictor when an edge is not detected. The second formula is a more expanded formula that includes the values of each pixel in the context. This study includes 2 versions of the GED predictor which are representative of the 2 formulas available to the GED predictor. The first GED predictor includes the simplified formula for *P*, formula (17, and the second GED predictor includes the complex formula for *P*, formula (18.

$$P = W + N - NW \quad (17)$$

$$P = \frac{3(W+N)}{8} + \frac{(NW + WW + NN)}{12} \quad (18)$$

4.1.5 Gradient Adjusted Predictor

Another popular predictor that is included in this study is the GAP. This predictor is used in the CALIC algorithm. The context of the GAP is made of 7 pixels: *N*, *W*, *NW*, *NN*, *WW*, *NE*, and *NNE*. This 7 pixel context consists of the pixel context of the GED and MED predictors, but is expanded to include pixels east of the target pixel. The inclusion of the *NE* and *NNE* pixels allows the calculation of change energy to be more accurate. Accuracy is improved because information about change to the east of the target pixel is factored into the prediction unlike the static linear, MED, and GED predictors. Using formulas (19 and (20, the change energy in the horizontal and the change energy in the vertical direction are captured.

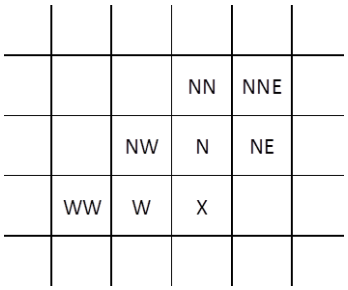

Figure 4-11: Gradient Adjusted Predictor Target Pixel Context

$$d_h = |W - WW| + |N - NW| + |N - NE| \quad (19)$$

$$d_v = |W - NW| + |N - NN| + |NE - NNE| \quad (20)$$

The GAP algorithm uses a base value, $\bar{x}$, in the calculation of target pixel predictions. The base value is a combination of the average of the *N* and *W* pixel values and a fourth of the difference between the *NE* and *NW* pixel values as given in formula (21. Depending on the change energy of the target pixel, the base value, $\bar{x}$, is combined with neighboring pixel values to generate an improved prediction value, $\hat{x}$, for the target pixel.

$$\bar{x} = \frac{W + N}{2} + \frac{NE - NW}{4} \quad (21)$$

Edge detection using the GAP algorithm is more robust than the previously discussed predictors. The MED and GED predictors focus on identifying if a horizontal or vertical edge exists. Horizontal and vertical edges are detected using the GAP, but the GAP also assesses the strength of horizontal and vertical edges. Using the difference between the horizontal change energy and the vertical change energy, the GAP is able to determine strong, average, and weak horizontal or vertical edges. Using formula (22, the predicted value of the target pixel, $\hat{x}$, is calculated based on the strength of the edge found at the target pixel. The base value, $\bar{x}$, has more influence on the value of the target pixel, $\hat{x}$, as the absolute value of change energy diminishes.

$$\hat{x} = \begin{cases} W, if\ (d_v - d_h > 80) \\ \frac{(\bar{x} + W)}{2}, if\ (d_v - d_h > 32) \\ \frac{(3\bar{x} + W)}{4}, if\ (d_v - d_h > 8) \\ N, if\ (d_v - d_h > -80) \\ \frac{(\bar{x} + N)}{2}, if\ (d_v - d_h > -32) \\ \frac{(3\bar{x} + N)}{4}, if\ (d_v - d_h > -8) \end{cases} \quad (22)$$

4.1.6 Accurate Gradient Selection Predictor

The authors of *A Gradient Based Predictive Coding for Lossless Image Compression,* [53]*,* present a predictor, Accurate Gradient Selection Predictor (AGSP), which is designed to increase the accuracy of the prediction process. The context of the target pixel used in the AGSP algorithm consists of the same context of GAP, but is expanded to include pixels *NWW* and *NNW*. Not only is the context of the target pixel expanded, the AGSP includes two additional directions of change energy. The amount of change in pixel values measured in the northeast direction, 45°, and the amount of change in pixel values measured in the northwest direction, -45°, are included in the prediction algorithm. Using formulas (23, (24, (25, and (26, four different change energies are calculated in the horizontal, D_h, vertical, D_v, 45°, D_p, and -45°, D_m directions respectively.

		NNW	NN	NNE	
	NWW	NW	N	NE	
	WW	W	X		

Figure 4-12: Accurate Gradient Predictor Selection Target Pixel Context

$$D_h = \frac{\begin{pmatrix} 2 \times |W - WW| + 2 \times |N - NW| + 2 \times |N - NE| \\ +|NN - NNW| + |NN - NNE| + |NW - NWW| \end{pmatrix}}{9 + 1} \quad (23)$$

$$D_v = \frac{(2 \times |W - NW| + 2 \times |N - NN| + |NE - NNE| + |WW - NWW| + |NW - NNW|)}{7 + 1} \quad (24)$$

$$D_p = \frac{(2 \times |N - W| + 2 \times |N - NNE| + |NW - WW| + |NW - NN|)}{6 + 1} \quad (25)$$

$$D_m = \frac{(2 \times |W - NWW| + 2 \times |N - NNW| + |NE - NN|)}{5 + 1} \quad (26)$$

Rather than identify edges, AGSP identifies change energies and their related pixel values. Formulas (27, (28, (29, and (30 are used to identify the pixel values related to a given energy. Value C_h is the horizontal change value and is equivalent to pixel *W*. Value C_v is the vertical change value and is equivalent to pixel *N*. Value C_p is the 45° change value and is equivalent to pixel *NE*. Value C_m is the -45° change value and is equivalent to pixel *NW*. Formula (31 is used to predict the target pixel value. The directional change energies act as adjustors for the amount of directional change value that should be included in the prediction of target pixel values.

$$C_h = W \quad (27)$$

$$C_v = N \quad (28)$$

$$C_p = NE \quad (29)$$

$$C_m = NW \quad (30)$$

$$\hat{x} = \frac{D_{min1} \times C_{min2} + D_{min2} C_{min1}}{D_{min1} + D_{min2}} \quad (31)$$

After directional change energies and directional change values are found, the 2 lowest directional change energies are selected, D_{min1} and D_{min2}. The directional change energy of the first minimum, $D_{min1,}$ is multiplied by the directional change value of the second minimum, C_{min2}. Then the inverse is calculated as the directional change energy of the second minimum, $D_{min2,}$ multiplied by the directional change value of the first minimum, C_{min1}. The sum of these values is then divided by sum of the 2 minimum directional change energies. The resulting value is a directionally weighted combination of neighboring pixel values. Examples can be seen in Figure 4-13 and Figure 4-14 [53].

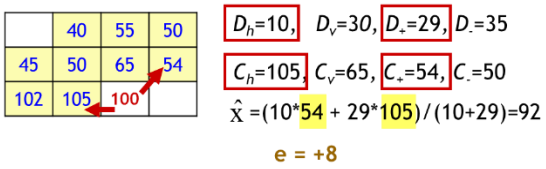

Figure 4-13: AGSP Horizontal Prediction

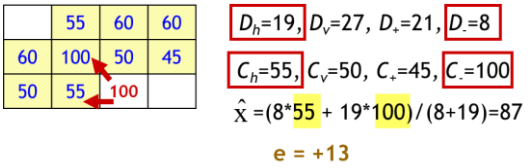

Figure 4-14: AGSP -45 Degree Prediction

4.1.7 Gradient Based Selection and Weighting

The authors of *Gradient Based Selective Weighting of Neighboring Pixels for Predictive Lossless Image Coding*, [54], present an algorithm, Gradient Based Selection and Weighting (GBSW), which "can be seen as an extension of [the] idea of [the] GAP predictor". The primary extension is that the GBSW predictor can identify complex edges and simple textures within the context of the target pixel. The size of the context of the target pixel is also the largest of those included in this study. The target pixel context of the GBSW predictor consists of the pixel context of the AGSP plus the *NNWW* pixel.

	NNWW	NNW	NN	NNE	
	NWW	NW	N	NE	
	WW	W	X		

Figure 4-15: Gradient Based Selection Weighting Target Pixel Context

Like the AGSP, the GBSW predictor calculates the change energy of four directions: horizontal, D_w; vertical, D_n; 45°, D_p; and -45°, D_m. The change energy calculations of the GBSW predictor are also similar to the change energy calculations of the AGSP, but the inclusion of the *NNWW* pixel in the GBSW predictor allows for expanded change energy capture. The GBSW predictor weighs and sums directional pairs in a first step, using formulas (32, (33, (34, and (35, and then averages the values in a second step, using formulas (36, (37, (38, and (39. The AGSP completes the same change energy calculations in only one step.

$$D_1 = 2 \times |W - WW| + 2 \times |N - NW| + 2 \times |NW - NWW| + 2 \times |N - NE| + |NN - NNW| + |NN - NNE| \quad (32)$$

$$D_2 = 2 \times |N - NN| + 2 \times |W - NW| + 2 \times |NW - NNW| + 2 \times |NE - NNE| + |WW - NWW| + |NWW - NNWW| \quad (33)$$

$$D_3 = 2 \times |W - NWW| + 2 \times |N - NNW| + |NW - NNWW| + |NE - NN| \quad (34)$$

$$D_4 = 2 \times |WW - NW| + 2 \times |N - NNE| + |W - N| + |NW - NN| \quad (35)$$

$$D_w = \left\lfloor \frac{D_1}{10} + 0.5 \right\rfloor \quad (36)$$

$$D_n = \left\lfloor \frac{D_2}{10} + 0.5 \right\rfloor \quad (37)$$

$$D_m = \left\lfloor \frac{D_3}{6} + 0.5 \right\rfloor \quad (38)$$

$$D_p = \left\lfloor \frac{D_4}{6} + 0.5 \right\rfloor \quad (39)$$

The GBSW predictor uses the same mappings, formulas (27, (28, (29, and (30, as the AGSP for directional change energy and the related pixel values. The neighboring pixel related to the horizontal change energy, D_h and D_1, is pixel *W*. Pixel *N* is the pixel related to the vertical change energy, D_v and D_2. The neighboring pixel related to the 45° change energy, D_p and D_4, is pixel *NE*. Pixel *NW* is the related pixel to the -45° change energy, D_m and D_3. The gradient values D_h, D_v, D_p, and D_m, are used in the formula that predicts the value of the target pixel.

The formula used by the GBSW predictor to calculate the value of the target pixel is the same formula used by the AGSP, formula (31. The 2 lowest directional change energies are selected. The first lowest gradient value, $D_{min1,}$ is multiplied by the neighboring pixel value in the second lowest direction, C_{min2}. Then the inverse is calculated by multiplying the second lowest gradient value, $D_{min2,}$ by the neighboring pixel value in the first lowest direction, $C_{min1,}$. The sum of these values is then divided by sum of the 2 minimum gradient values, D_{min2} and D_{min2}. This formula generates the value of the target pixel as a directionally weighted combination of neighboring pixel values.

4.1.8 Predictor Performance

The performance of the predictors is measured using mean absolute error; the equation is shown in formula (40. The absolute difference between the predicted value and the actual value of each pixel in an image is calculated. The sum of absolute differences is then divided by the count of pixels in the image. The lower the mean absolute error the more accurate the predictor performs. The accuracy of the predictors has a direct influence on the entropy and the distribution of values in the difference data set generated by the difference coder.

$$\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\left|I_{i,j} - \tilde{\mathrm{I}}_{i,j}\right|}{m \times n} \quad (40)$$

Each of the 10 predictors is applied to each image in the test image set. The mean absolute error between each predicted image data set and the original image data set is provided in Table 4-1. For each image, the mean absolute errors of the 4 predictors that generate the least amount of error are shaded in the table. The best performing predictor is the GBSW predictor. The GBSW predictor generates the lowest mean absolute error on 14 of 21 images and has an average mean absolute error of 5.07. The GAP is the second best predictor. The GAP has an average mean absolute error of 5.24, but only provides the most accurate predicted image values on 3 of 21 images. The third best predictor is the AGSP. The AGSP does not provide the most accurate predicted image values on any image, but provides an average mean absolute error of 5.25. The MED predictor is forth best predictor and it generates an average mean absolute error of 5.35.

Table 4-1 : Prediction Mean Absolute Error

	Linear - Horizontal	Linear - Vertical	Linear - Positive 45°	Linear - Negative 45°	Median Edge Detection	Detection - Complex Gradient Edge	Detection - Simple Gradient Edge	Gradient Adjusted Predictor	Selection Predictor	Accurate Gradient Selection & Weighting	Gradient Based
airplane.tiff	5.68%	5.48%	7.67%	7.64%	3.63%	4.12%	4.31%	3.73%	4.00%	3.89%
barn.png	6.90%	7.42%	9.31%	9.24%	5.45%	5.72%	6.19%	5.27%	5.62%	5.43%
bikes.png	13.06%	12.14%	16.96%	15.25%	9.64%	10.17%	10.36%	9.37%	8.10%	7.66%
birds.png	3.68%	3.81%	4.92%	4.65%	2.89%	2.93%	3.35%	2.72%	2.60%	2.51%
building_front.png	14.07%	11.47%	18.21%	17.13%	8.64%	9.11%	9.67%	8.76%	9.42%	9.17%
buildings.png	14.07%	16.50%	21.99%	23.68%	8.62%	8.69%	9.07%	8.75%	10.11%	9.52%
door.png	5.18%	4.68%	6.84%	6.12%	3.78%	3.76%	4.36%	3.78%	3.85%	3.73%
flower_window.png	6.26%	4.54%	7.52%	7.54%	3.42%	3.58%	3.76%	3.52%	3.08%	2.93%
girl.png	4.90%	5.81%	6.82%	7.04%	4.06%	4.16%	4.69%	3.80%	3.91%	3.74%
hats.png	4.83%	3.56%	6.04%	4.85%	3.20%	3.28%	3.68%	3.22%	2.67%	2.60%
house.tiff	6.76%	4.58%	8.04%	8.22%	3.40%	3.93%	4.17%	3.90%	4.04%	3.90%
JellyBeans.tiff	2.42%	2.75%	3.47%	3.42%	1.72%	1.85%	1.88%	1.59%	1.57%	1.52%
Lena.tiff	4.85%	6.45%	6.67%	7.54%	4.40%	4.48%	5.04%	3.94%	3.92%	3.79%
lenna.jpg	4.64%	6.63%	8.13%	7.12%	3.48%	4.03%	3.86%	3.51%	3.42%	3.25%
lighthouse.png	9.92%	6.98%	10.74%	11.34%	6.40%	6.55%	7.19%	6.28%	6.59%	6.43%
man.tiff	6.79%	7.39%	8.79%	8.78%	5.71%	5.77%	6.60%	5.27%	5.37%	5.26%
Mandrill.tiff	19.46%	15.05%	21.39%	21.13%	14.33%	14.82%	15.77%	13.78%	13.60%	13.31%
mural_home.png	9.53%	9.88%	12.53%	12.55%	7.65%	8.12%	8.40%	7.58%	7.68%	7.46%
Peppers.tiff	6.22%	6.54%	7.04%	7.31%	5.73%	5.43%	7.47%	5.29%	4.78%	4.78%
Pt5hand.tif	2.10%	2.17%	2.65%	2.64%	1.75%	1.69%	2.18%	1.64%	1.84%	1.77%
statue.png	5.84%	5.53%	7.29%	7.15%	4.45%	4.61%	5.06%	4.28%	4.02%	3.86%
average	7.48%	7.11%	9.67%	9.54%	5.35%	5.56%	6.05%	5.24%	5.25%	5.07%

The methods used in the GBSW predictor and the AGSP are very similar. One of the differences between the 2 predictors is that the GBSW predictor includes an additional pixel in the context used to predict target pixel values. The second major difference is that the change energy formulas used by GBSW also includes the additional pixel and some formula variations. The prediction results of Table 4-1 show that these differences allow the GBSW predictor to generate better predicted image values than the AGSP. Within the test image set, there is no instance where the AGSP provides better prediction than the GBSW predictor. Because the mean square error of the AGSP is consistently greater than the GBSW predictor, the AGSP can be excluded from further experiments that include the GBSW predictor.

The GED predictor is designed to be a compromise between the simplicity of the MED predictor and the accuracy of the GAP. The context of the GED predictor, 5 pixels, is greater than the context of the MED predictor, 3 pixels, but is smaller than the context of the GAP, 7 pixels. The GED predictor is similar to the GAP in that both calculate change energy in the horizontal direction and the vertical direction. Both the GED predictor and the GAP calculate a predicted value and then adjust that prediction based on a given threshold. The GED predictor offers less complexity, O(6n), by using only 3 threshold ranges versus the computational complexity, O(9n), of the GAP that uses 6 threshold ranges. Despite the compromising design of the GED predictor, there are only 4 instances within the test set that the GED predictor generates a predicted image data set more accurate than or between the accuracy of the MED predictor and the GAP. The impact of both versions of the GED predictor is minimal and the two GED predictors, simple GED using formula (17 and complex GED using formula (18, can be excluded from further experiments.

The performance of the static linear predictors is very low. There is no instance within the test image set of any static linear predictor having greater accuracy than the other predictors. The strongest predictors have an average mean sum error less than 5.50. Each of the static linear predictors has an average mean sum error above 7.00. Although the static linear predictors have

the lowest complexity, O(n), the MED predictor improves performance with only a slight increase in complexity, O(3n). The linear predictors can be excluded from further experiments.

4.1.9 Predictor Summary

Several of the predictors that were reviewed can be removed from further experimentation. Using the *Lena* image as an example, the error images of each predictor can be seen in Figure 4-16- Figure 4-25. The AGSP is a strong performing predictor, but consistently performs with less accuracy than the GBSW predictor. The GED predictor fails to consistently provide more accurate prediction image data sets than the MED predictor despite the GED predictor having a larger context and greater complexity. The linear predictors fail to provide the same performance of the MED predictor due the ability of the MED predictor to dynamically switch between vertical, horizontal, and average predictions. Further experimentations are based on the remaining 3 predictors: MED, GAP, and GBSW.

Figure 4-16: Horizontal Prediction Error

Figure 4-17: Vertical Prediction Error

Figure 4-18: Plus Angle Error

Figure 4-19: Minus Angle Error

Figure 4-20: Median Edge Detection Error

Figure 4-21: Gradient Edge Detection – Complex Error

Figure 4-22: Gradient Edge Detection – Simple Error

Figure 4-23: Gradient Adjusted Prediction Error

Figure 4-24: Accurate Gradient Selection Predictor Error

Figure 4-25: Gradient Based Selection and Weighting

4.2 DIFFERENCE CODER

4.2.1 Background on Differential Coding

Differential coding is the process of extracting the differences between a source data set and a target data set as seen in Figure 4-26. In this example, the differences are captured using the exclusive-or operator on each bit of the data. The collection of differences is the difference coded representation of the target data set. The target data set can be created by applying the difference coded data set to the source data set as seen in Figure 4-27. When the correlation between the source data set and target data set is high, the difference coded representation of the target data set may have lower entropy and a smaller range of values than the target data set. This would allow the difference coded data set to be more compressible than the target data set.

Figure 4-26: Example of Difference Coding

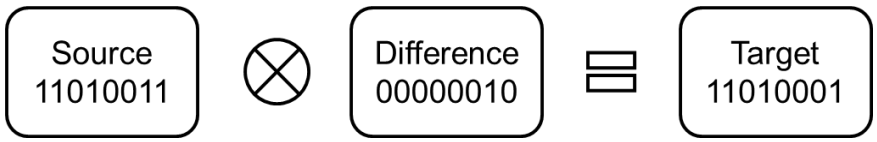

Figure 4-27: Recovering the Target Data Set

The example data set in Table 4-2 consists of temperature readings taken each hour. There is a set of temperature readings for day 1 and a set of temperature readings for day 2. Day 1 is the source data set. Day 2 is the target data set. The subtracted difference between the data sets of the 2 days is the difference coded representation of Day 2 values. The values of Day 2 require 7 bits per entry to encode. The difference encoded values of Day 2 require only 4 bits per entry to represent. This is a significant reduction in the number of bits required to represent the same amount of information. The compression ratio is the original bit count divided by the compressed bit count, (7 bits*24 entries)/(4 bits*24 entries) = 1.8.

Table 4-2 : Sample Hourly Temperature Readings

	0:00	1:00	2:00	3:00	4:00	5:00	6:00	7:00	8:00	9:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00	21:00	22:00	23:00
day 1	62°F	61°F	60°F	59°F	58°F	57°F	56°F	55°F	55°F	56°F	57°F	58°F	59°F	59°F	62°F	63°F	61°F	61°F	63°F	64°F	62°F	60°F	58°F	57°F
day 2	56°F	56°F	54°F	54°F	54°F	54°F	54°F	54°F	54°F	54°F	57°F	60°F	63°F	65°F	67°F	68°F	68°F	68°F	66°F	66°F	64°F	61°F	61°F	60°F
diff	-6°F	-5°F	-6°F	-5°F	-4°F	-3°F	-2°F	-1°F	-1°F	-2°F	0°F	2°F	4°F	6°F	5°F	5°F	7°F	**7°F**	3°F	2°F	2°F	1°F	3°F	3°F

4.2.2 Data Expansion from Differential Coding

The example data set in Table 4-3 displays a challenge of differential coding. When using subtraction to capture differences, the difference values that result can be positive or negative. If the range of input values is 0 to 255, the range of difference values can be -255 to +255. In this case, the input values require up to 8 bits for representation, but the difference values require up to 9 bits for two's compliment representation. This could cause the difference coder to expand the amount of space required to represent the target data set. The maximum value in the target data set, day 2, is 14 in Table 4-3. This value only requires 4 bits to represent. The range of difference values is -5 to 8. This range requires 5 bits using two's compliment representation. Difference coding would result data expansion and have a compression ratio of (4 bits*24 entries)/(5 bits*24 entries) = 0.8.

Table 4-3 : Sample Hourly Temperature Subtraction Difference Readings

	0:00	1:00	2:00	3:00	4:00	5:00	6:00	7:00	8:00	9:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00	21:00	22:00	23:00
day 1	7°C	6°C	5°C	4°C	3°C	2°C	1°C	0°C	0°C	1°C	2°C	3°C	4°C	4°C	7°C	8°C	6°C	6°C	8°C	9°C	7°C	5°C	3°C	2°C
day 2	2°C	2°C	0°C	0°C	0°C	0°C	0°C	0°C	0°C	0°C	3°C	6°C	9°C	11°C	13°C	14°C	14°C	14°C	12°C	12°C	10°C	7°C	7°C	6°C
sub_diff	-5°C	-4°C	-5°C	-4°C	-3°C	-2°C	-1°C	0°C	0°C	-1°C	1°C	3°C	5°C	7°C	6°C	6°C	8°C	8°C	4°C	3°C	3°C	2°C	4°C	4°C

A simple way to prevent possible data expansion is to use the exclusive-or operator instead of the subtraction operator to capture differences as in Table 4-4. The source values and target values are converted from decimal values to binary values. The bit-wise exclusive-or operator is applied to each bit of the values. The output of the bit-wise exclusive-or operation is the difference coded representation of the target data set. The XOR difference coded binary values can then be converted back to the decimal values. This process ensures that the number of bits in the difference coded value is not greater than the number of bits in the source and target values.

Table 4-4 : Sample Hourly Temperature Subtraction Exclusive-Or Readings

	0:00	1:00	2:00	3:00	4:00	5:00	6:00	7:00	8:00	9:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00	21:00	22:00	23:00
day 1	7°C	6°C	5°C	4°C	3°C	2°C	1°C	0°C	0°C	1°C	2°C	3°C	4°C	4°C	7°C	8°C	6°C	6°C	8°C	9°C	7°C	5°C	3°C	2°C
day 2	2°C	2°C	0°C	0°C	0°C	0°C	0°C	0°C	0°C	0°C	3°C	6°C	9°C	11°C	13°C	14°C	14°C	14°C	12°C	12°C	10°C	7°C	7°C	6°C
bin 1	0111	0110	0101	0100	0011	0010	0001	0000	0000	0001	0010	0011	0100	0100	0111	1000	0110	0110	1000	1001	0111	0101	0011	0010
bin 2	0010	0010	0000	0000	0000	0000	0000	0000	0000	0000	0011	0110	1001	1011	1101	1110	1110	1110	1100	1100	1010	0111	0111	0110
xor	0101	0100	0101	0100	0011	0010	0001	0000	0000	0001	0001	0101	1101	1111	1010	0110	1000	1000	0100	0101	1101	0010	0100	0100
xor_diff	5	4	5	4	3	2	1	0	0	1	1	5	13	15	10	6	8	8	4	5	13	2	4	4

4.2.3 Differential Coding in Lossless Image Compression

The foundation of differential coding is extracting the differences between two data sets, the source and the target. In lossless image compression there is only one data set, the image. The image is the target data set that must be difference coded. In order to apply differential coding to an image, a source data set is also required. The source data set is generated from the image using the prediction component previously discussed. A predictor is applied to the image which creates a data set of predicted values. The difference coder uses the predicted values as a source data set and the original image as the target data set to extract a difference coded representation of the original image. This process is illustrated in Figure 4-28.

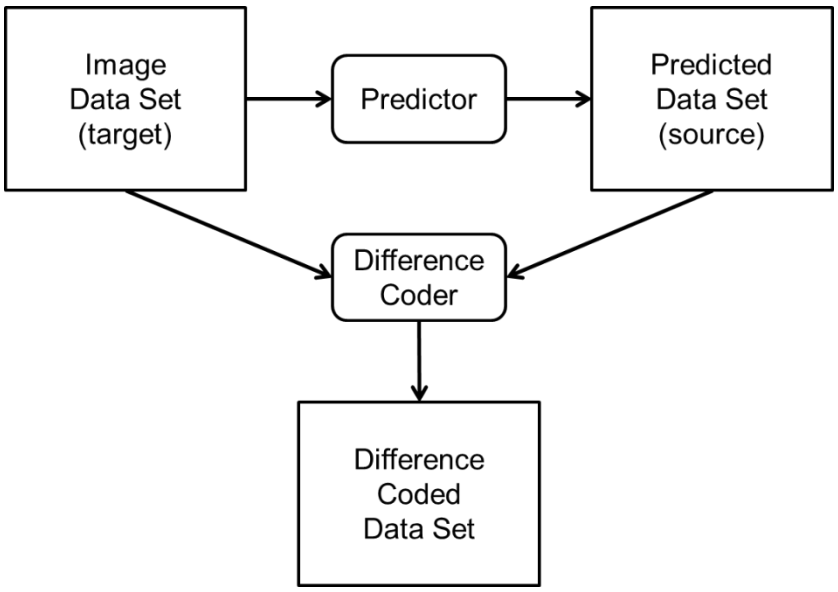

Figure 4-28: Dissertation Algorithm Difference Coder

4.2.4 The Application of Differential Coding in the Algorithm

In lossless image compression using prediction and difference coding, the two components are tightly coupled. When encoding, the entire image data set is available to the encoder as the target data set. The prediction step can be executed independently and produce the full predicted data set as the source data set. This will allow the difference coder to do matrix level operations. The difference coder can apply the exclusive-or operator in a bit-wise manner between each corresponding pixel of the source data set, the predicted image values, and the target data set, the original image values. This allows the difference coded data set to be generated in one matrix-level operation.

The decoder does not allow matrix-level operations for difference coding. Each pixel is processed sequentially. The pixels that form the context of the target pixel are used to create a predicted value for the target pixel. The predicted value is then combined with the difference coded value of the target pixel to recover the actual value of the target pixel. The newly decoded actual value of the target pixel is then available for use in the context of the next target pixel. This process repeats, pixel by pixel, until the entire image data set is recovered. This cyclic process is illustrated in Figure 4-29.

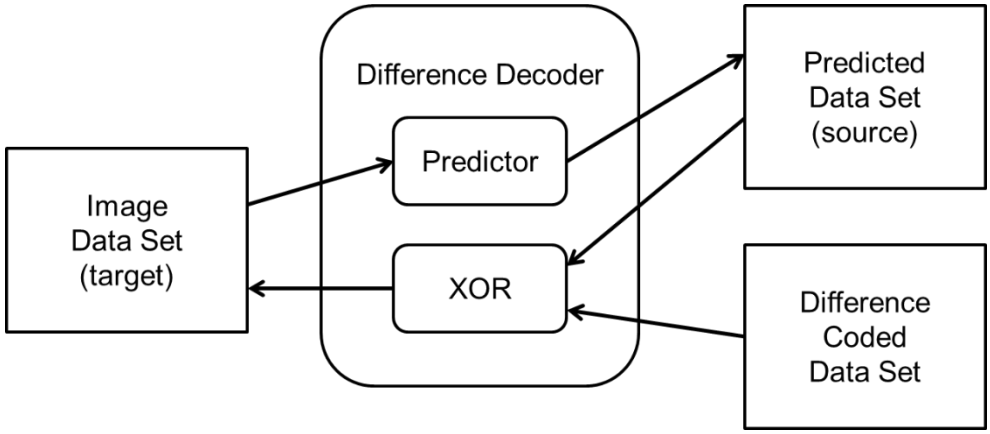

Figure 4-29: Dissertation Algorithm Difference Decoder

A major element of the dissertation is measuring the usefulness of the exclusive-or operator as a difference coder in the context of lossless image compression. This is done by analyzing the overall performance of the XOR difference coder in relation to compression performance. The difference coding component is combined with various predictors and data transforms. The performance of these combinations is measured to determine the best application of the exclusive-or operator in lossless image compression. Discussion of these combinations and their performance is detailed in the “Algorithm Design” chapter.

4.3 BASELINING

4.3.1 Impact of XOR on Correlation

Difference coding using the exclusive-or operator ensures that the range of the coded values does not exceed the range of the original image data set. However, use of the exclusive-or operator has the ability to lose the correlation of the original data set. The example in Table 4-5 is of a data set that only varies by 1 unit in either direction. The correlation of the original data set is .90. The correlation of the difference coded data set is -.48. The loss of correlation is because the exclusive-or operator captures the bit differences between 2 values and not the linear distance between two values. Bit difference and linear distance may vary greatly. This is most noticeable when the binary representation of values has change in several bits. For example, the decimal value 4, as the binary value 100, and the decimal value 3, as the binary value 011, have

no equal bits. Linearly, the 2 decimal values only differ by a value of 1. The XOR of 4 and 3 captures change at each bit, 111, and is equivalent to decimal value 7. The difference between linear distance and bit difference is the cause of correlation loss. The potential for correlation loss from the exclusive-or operator increases as the values of input data increases.

Table 4-5 : Highly Correlated Sample Data Set

data set	0	1	2	3	4	5	6	7	6	5	4	3	2	1	0
XOR		001	011	001	111	001	011	001	001	011	001	111	001	011	001
diff		1	3	1	7	1	3	1	1	3	1	7	1	3	1

4.3.2 Baseline Mapping Process

One way of restoring correlation to difference coded values is through baselining. Baselining is the process of identifying a starting point for bit differences, sorting bit differences according to the identified starting point, and then reassigning bit difference values to the sorted values. The starting point for bit differences is each set of equal source values and target values. In Table 4-6, the starting point is along the diagonal access where each value is equal to 0. The values in each column are then sorted so that the values alternate in ascending distance from the starting point as seen in Table 4-7. The top row values represent the possible source data set values and the left column values represent the possible target data set values.

The tables below, Table 4-6 and Table 4-7, are mapping tables that index the XOR difference values to baseline values. The location of the difference coded value in the column of the source data set value of the difference table is the location of the baseline value in the in the column of the source data set value of the mapping table. A simple example is for a source value of 3 and a target value of 4. The difference value is found in the fourth column and fifth row of Table 4-6. The baselined value is found in the same location, fourth column and fifth row of Table 4-7. For each difference coded value, a baseline value can be extracted from the mapping table that will restore some correlation to the difference coded values.

Table 4-6 : Baseline Difference Table

		Source							
		0	1	2	3	4	5	6	7
Target	0	0	1	2	3	4	5	6	7
	1	1	0	3	2	5	4	7	6
	2	2	3	0	1	6	7	4	5
	3	3	2	1	0	7	6	5	4
	4	4	5	6	7	0	1	2	3
	5	5	4	7	6	1	0	3	2
	6	6	7	4	5	2	3	0	1
	7	7	6	5	4	3	2	1	0

Table 4-7 : Baseline Mapping Table

		Source							
		0	1	2	3	4	5	6	7
Target	0	0	2	4	6	7	7	7	7
	1	1	0	2	4	6	6	6	6
	2	2	1	0	2	4	5	5	5
	3	3	3	1	0	2	4	4	4
	4	4	4	3	1	0	2	3	3
	5	5	5	5	3	1	0	2	2
	6	6	6	6	5	3	1	0	1
	7	7	7	7	7	5	3	1	0

Mapping of the difference value to the baseline value can be completed using the procedure shown in formula (41. The difference coded values in Table 4-5 have been listed in Table 4-8. The first difference coded value, *d*, in the “diff” row of Table 4-8 is 1. The original value from the source “data set” of Table 4-8, *t*, is 1. The row of value *d=1* in column *t=1* of the difference table, shown as Table 4-6, is 1. The baseline value, *b*, is found in the same location of the mapping table as shown in Table 4-7. The baseline value, *b*, at row 1 and column 2 is the value 2. This baselined value of 2 is logged in the last row, “baseline”, of Table 4-8. The process repeats for each difference coded value. The result of baselining is the last row of Table 4-8, the data set [2 2 2 2 2 2 1 1 1 1 1 1 1]. The baselined data set has less variance than the difference coded data set and a correlation of .71.

For each *d* in difference coded value

Mapping_table(Difference_table(:,*t*)==*d*,*t*) (41)

Table 4-8 : Baselined Highly Correlated Sample Data Set

data set	0	1	2	3	4	5	6	7	6	5	4	3	2	1	0
XOR		001	011	001	111	001	011	001	001	011	001	111	001	011	001
diff		1	3	1	7	1	3	1	1	3	1	7	1	3	1
baseline		2	2	2	2	2	2	2	1	1	1	1	1	1	1

The example in Table 4-8 shows 2 levels of data reduction. When the neighboring values are difference coded with the exclusive-or operator, the difference coded data set has fewer symbols than the original data set. The original data set has all values between 0 and 7, but the difference coded data set consists of only 1, 3, and 7. The difference coded data set also has a skewed distribution of symbol values. The smaller data set and skewed symbol distribution will allow the difference coded data set to be more compressible than the original data set. The entropy of the original data set is 2.97 and the entropy of the difference coded data set is 1.38. Baselining the difference coded values results in additional reductions to the data set. The baselined data set consist only of the values 1 and 2. Although the distribution of the baselined values is not skewed, only 1 bit is required to represent the data and entropy of the data set is 1. The example of Table 4-8 demonstrates how baselining can improve the compressibility of difference coded values generated by the exclusive-or operator.

4.3.3 Baselining Performance

The goal of baselining is to reintroduce some of the correlation that is lost by using the XOR-based difference coder. The correlation and entropy measures of the original, subtraction-based difference, XOR-based difference, and baselined XOR-based difference of the Lenna image are provided in Table 4-9. The correlation restored by baselining the XOR-based difference data set exceeds that of the subtraction-based difference data set. The baseline of the XOR difference data set also has lower entropy of than the XOR difference data set. The increased correlation and decreased entropy are beneficial to compression.

Table 4-9 : Difference Signal Correlation & Entropy

	Original	Subtraction Difference	XOR Difference	Baseline of XOR Difference
Correlation	0.9863	0.5287	0.2136	0.5314
Entropy	7.6619	3.0796	4.5196	4.0353

The impact of baselining can be seen in the histograms of the 4 different data sets. The entropy of each data set directly relates to the histograms of Figure 4-30 - Figure 4-33, respectively. The

first histogram of Figure 4-30 is of the original image. The likelihood of possible values is very distributed and the entropy is high at 7.66. The data set generated by subtracting predicted values from original values is highly skewed towards zero and has an entropy measure of 3.08. The related histogram is given in Figure 4-31. When the difference data set is generated using the exclusive-or operator, the resulting data set is also skewed towards zero as seen in Figure 4-32. Both the subtraction difference data set and XOR difference data set are more compressible than the original image data set.

The XOR-based difference data set differs from the subtraction-based difference data set in two major ways. The first difference is that the XOR difference data set only consists of positive values. The XOR difference data set has fewer possible values than the subtraction difference data set. For an 8-bit gray image, the possible range of values for the subtraction-based difference data set is -255 to 255. The possible range of values for the XOR-based difference data set is 0 to 255. Having fewer symbols may increase the compressibility of the data set.

The second significant difference between the subtraction-based difference data set and the XOR-based difference data set is that there are spikes in the distribution of XOR-based difference symbols at each power of 2. These spikes in symbol distribution are because the exclusive-or operator measures bit-plane differences instead of linear difference. Baselining remaps the XOR difference data set so that it better reflects linear differences. The baselined XOR data set can be seen in the histogram of Figure 4-33. The variations in the quality of the data sets are noticeable in the difference images shown in Figure 4-34 - Figure 4-37. The XOR difference data set of Figure 4-36 has a lot of noise that is removed by through the baselining process. The skewed values of the baselined XOR difference data set are more compressible than the original XOR difference data set.

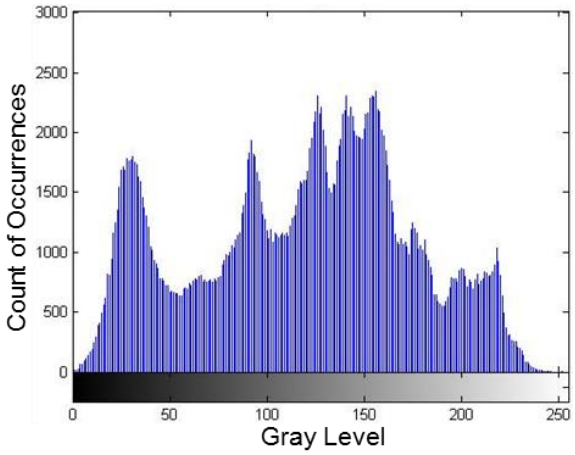

Figure 4-30: Histogram of Lenna Original Image

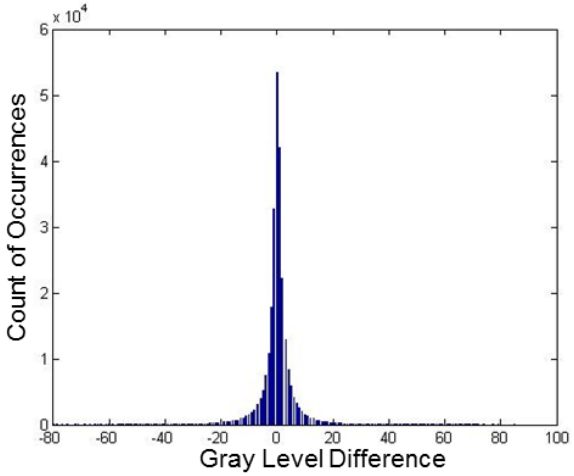

Figure 4-31: Histogram of Lenna Subtraction Difference Signal

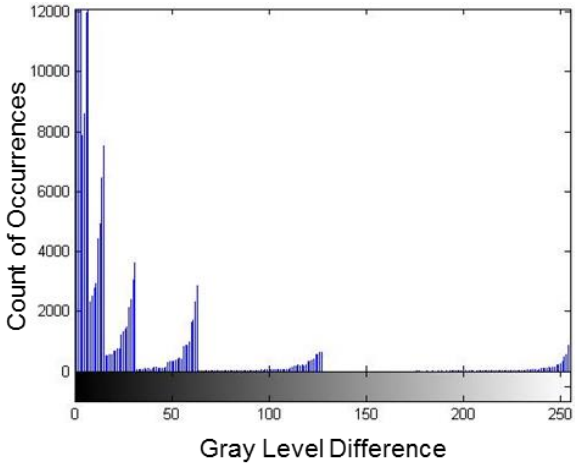

Figure 4-32: Histogram of Lenna XOR Difference Signal

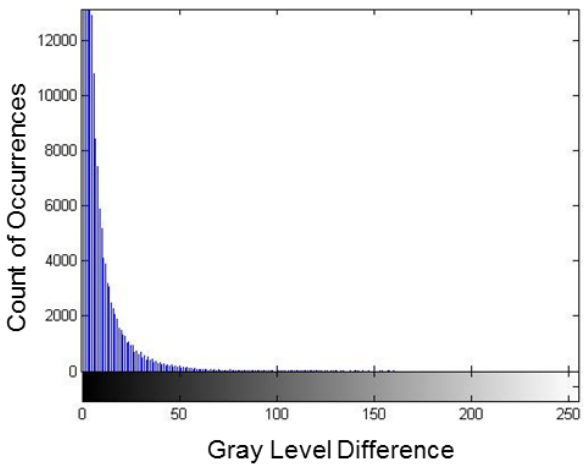

Figure 4-33: Histogram of Baseline of Lenna XOR Difference

Figure 4-34: Lenna Original Image

Figure 4-35: Lenna Subtraction Difference Signal

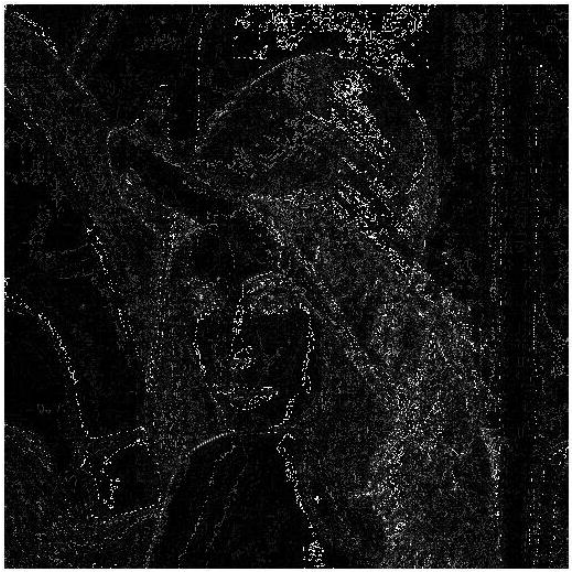

Figure 4-36: Lenna XOR Difference Signal

Figure 4-37: Baseline of Lenna XOR Difference Signal

4.3.4 Baselining Summary

The baselining process improves the correlation, entropy, and skewness of images prior to encoding. The correlation of the XOR difference data set is more than doubled due to the noise reduction provided by baselining. The noise reduction also contributes to a reduction of entropy.

The distribution of symbols in the data set without noise is in a positive exponential curve. These 3 factors, correlation, entropy, and skewness, contribute to improved compression. Baselining can have a positive impact on the performance of the proposed compression algorithm.

4.4 BURROWS-WHEELER TRANSFORM

4.4.1 Introduction to the Burrows-Wheeler Transform

The Burrows-Wheeler Transform (BWT) is primarily used as a text transform. It rearranges the symbols of an input string in an effort to create groupings of similar symbols and symbol patterns. Runs of similar symbols and patterns are generated if the input string has a lot of symbol repetition and correlation. The reordered data set, having many repeated sequences of symbols, will perform better than data with higher variance when compressed using dictionary encoding techniques such as Run-Length or Lempel-Ziv-Welch coding.

The BWT is reversible and only requires one additional character to recover the original data set. The values of the original data set are unchanged. An end-of-line character is added to the end of the data set prior to processing. The output of the BWT is a permutation of the original data set. The end-of-line character is embedded within the permuted data. Recovery of original order of the data set is completed using a series of manipulations of the permuted data. The ability to improve the order of the data in a reversible manner prior to dictionary coding, at the cost of only one additional character, makes the BWT an ideal component of the dissertation algorithm.

4.4.2 Execution of the Burrows-Wheeler Transform

4.4.2.1 Traditional Application of the Burrows-Wheeler Transform

The BWT consists of three steps. In step 1, the data set is rotated once for each character in the data set. Each rotation is saved as a row in a matrix. The rows of the matrix are then sorted in step 2. Finally in step 3, the last column of the matrix is saved as the final output of the transform.

In the following example, the difference coded values of Table 4-5 are transformed using the BWT. The "$" symbol is the end-of-line character and is added to the end of the data set. The starting values are listed in Table 4-10. This data set is then rotated one position to left and added to the data set as an additional row. The rotational process repeats until the end-of-line character is in the first position. The full transform data set is listed in Table 4-11. The rows of the matrix are then sorted in ascending order, with the end-of-line character having the lowest value. The sorted transform data set is given in Table 4-12. The last column of the transform data set, the sorted matrix, is the BWT representation of the original data set.

Table 4-10 : Burrows-Wheeler Transform Input Data Set

1	3	1	7	1	3	1	1	3	1	7	1	3	1	$

Table 4-11 : Unsorted Burrows-Wheeler Transform Matrix

1	3	1	7	1	3	1	1	3	1	7	1	3	1	$
3	1	7	1	3	1	1	3	1	7	1	3	1	$	1
1	7	1	3	1	1	3	1	7	1	3	1	$	1	3
7	1	3	1	1	3	1	7	1	3	1	$	1	3	1
1	3	1	1	3	1	7	1	3	1	$	1	3	1	7
3	1	1	3	1	7	1	3	1	$	1	3	1	7	1
1	1	3	1	7	1	3	1	$	1	3	1	7	1	3
1	3	1	7	1	3	1	$	1	3	1	7	1	3	1
3	1	7	1	3	1	$	1	3	1	7	1	3	1	1
1	7	1	3	1	$	1	3	1	7	1	3	1	1	3
7	1	3	1	$	1	3	1	7	1	3	1	1	3	1
1	3	1	$	1	3	1	7	1	3	1	1	3	1	7
3	1	$	1	3	1	7	1	3	1	1	3	1	7	1
1	$	1	3	1	7	1	3	1	1	3	1	7	1	3
$	1	3	1	7	1	3	1	1	3	1	7	1	3	1

Table 4-12 : Sorted Burrows-Wheeler Transform Matrix

$	1	3	1	7	1	3	1	1	3	1	7	1	3	1
1	$	1	3	1	7	1	3	1	1	3	1	7	1	3
1	1	3	1	7	1	3	1	$	1	3	1	7	1	3
1	3	1	$	1	3	1	7	1	3	1	1	3	1	7
1	3	1	1	3	1	7	1	3	1	$	1	3	1	7
1	3	1	7	1	3	1	$	1	3	1	7	1	3	1
1	3	1	7	1	3	1	1	3	1	7	1	3	1	$
1	7	1	3	1	$	1	3	1	7	1	3	1	1	3
1	7	1	3	1	1	3	1	7	1	3	1	$	1	3
3	1	$	1	3	1	7	1	3	1	1	3	1	7	1
3	1	1	3	1	7	1	3	1	$	1	3	1	7	1
3	1	7	1	3	1	$	1	3	1	7	1	3	1	1
3	1	7	1	3	1	1	3	1	7	1	3	1	$	1
7	1	3	1	$	1	3	1	7	1	3	1	1	3	1
7	1	3	1	1	3	1	7	1	3	1	$	1	3	1

Table 4-13 : Burrows-Wheeler Transform Output Data Set

1	3	3	7	7	1	$	3	3	1	1	1	1	1	1

4.4.2.2 Traditional Reversal of the Burrows-Wheeler Transform

The reverse of the BWT can be extracted using only the transformed data set. The reverse transform consists of a repeated cycle of 2 steps. In the first step, the transformed data set is prepended to the sorted data set. In the second step, the combination is sorted. The two steps repeat until the length of the resulting matrix is equal to the length of the transformed data set. In the resulting matrix there will be a row that has the "$" character as the last character. This row is the original data set in the original order.

Using the previously generated transform listed in Table 4-13 as an example, the original data set can be recovered. The matrix begins as a single column as shown in Table 4-14. This column contains the sorted transform. The unsorted transform is then added as a column to the front of the matrix as shown in Table 4-15. The combination of the 2 columns is then sorted by row as shown in Table 4-16. The process of adding the unsorted transform to the matrix, shown again in Table 4-17, and then sorting the full matrix by rows, shown again in Table 4-18, continues to repeat. When the full matrix is generated, as shown in Table 4-19, and then sorted, as shown in Table 4-20, the original data set in the original order is found in the row that ends with the end of

line character. In this example the original data set in the original order, displayed in Table 4-21, *is* found in row 7 of the matrix in Table 4-20.

Table 4-14 : Burrows-Wheeler Transform Recovery, Step 1

$
1
1
1
1
1
1
1
1
3
3
3
3
7
7

Table 4-15 : Burrows-Wheeler Transform Recovery, Step 2

1	$
3	1
3	1
7	1
7	1
1	1
$	1
3	1
3	1
1	3
1	3
1	3
1	3
1	7
1	7

Table 4-16 : Burrows-Wheeler Transform Recovery, Step 3

$	1
1	$
1	1
1	3
1	3
1	3
1	3
1	7
1	7
3	1
3	1
3	1
3	1
7	1
7	1

Table 4-17 : Burrows-Wheeler Transform Recovery, Step 4

1	$	1
3	1	$
3	1	1
7	1	3
7	1	3
1	1	3
$	1	3
3	1	7
3	1	7
1	3	1
1	3	1
1	3	1
1	3	1
1	7	1
1	7	1

Table 4-18 : Burrows-Wheeler Transform Recovery, Step 5

$	1	3
1	$	1
1	1	3
1	3	1
1	3	1
1	3	1
1	3	1
1	7	1
1	7	1
3	1	$
3	1	1
3	1	7
3	1	7
7	1	3
7	1	3

Table 4-19 : Burrows-Wheeler Transform Recovered Unsorted Transform Matrix

1	$	1	3	1	7	1	3	1	1	3	1	7	1	3
3	1	$	1	3	1	7	1	3	1	1	3	1	7	1
3	1	1	3	1	7	1	3	1	$	1	3	1	7	1
7	1	3	1	$	1	3	1	7	1	3	1	1	3	1
7	1	3	1	1	3	1	7	1	3	1	$	1	3	1
1	1	3	1	7	1	3	1	$	1	3	1	7	1	3
$	1	3	1	7	1	3	1	1	3	1	7	1	3	1
3	1	7	1	3	1	$	1	3	1	7	1	3	1	1
3	1	7	1	3	1	1	3	1	7	1	3	1	$	1
1	3	1	$	1	3	1	7	1	3	1	1	3	1	7
1	3	1	1	3	1	7	1	3	1	$	1	3	1	7
1	3	1	7	1	3	1	$	1	3	1	7	1	3	1
1	3	1	7	1	3	1	1	3	1	7	1	3	1	$
1	7	1	3	1	$	1	3	1	7	1	3	1	1	3
1	7	1	3	1	1	3	1	7	1	3	1	$	1	3

Table 4-20 : Burrows-Wheeler Transform Recovered Sorted Transform Matrix

$	1	3	1	7	1	3	1	1	3	1	7	1	3	1
1	$	1	3	1	7	1	3	1	1	3	1	7	1	3
1	1	3	1	7	1	3	1	$	1	3	1	7	1	3
1	3	1	$	1	3	1	7	1	3	1	1	3	1	7
1	3	1	1	3	1	7	1	3	1	$	1	3	1	7
1	3	1	7	1	3	1	$	1	3	1	7	1	3	1
1	3	1	7	1	3	1	1	3	1	7	1	3	1	$
1	7	1	3	1	$	1	3	1	7	1	3	1	1	3
1	7	1	3	1	1	3	1	7	1	3	1	$	1	3
3	1	$	1	3	1	7	1	3	1	1	3	1	7	1
3	1	1	3	1	7	1	3	1	$	1	3	1	7	1
3	1	7	1	3	1	$	1	3	1	7	1	3	1	1
3	1	7	1	3	1	1	3	1	7	1	3	1	$	1
7	1	3	1	$	1	3	1	7	1	3	1	1	3	1
7	1	3	1	1	3	1	7	1	3	1	$	1	3	1

Table 4-21 : Burrows-Wheeler Transform Recovered Data Set

1	3	1	7	1	3	1	1	3	1	7	1	3	1	$

4.4.3 Burrows-Wheeler Transform of Images

The BWT can also be applied to image data. The pixels of the image are used as the character values for input data. The matrix format of the image is converted to an array format by processing the image linearly column by column or row by row. An *m* row by *n* column by *p*

dimensional image, $m \times n \times p$, can be processed as *p* individual sets of $m \times n$ length data. Figure 4-38 is an example of the pixel ordering for an 8x8x3 dimensional image matrix.

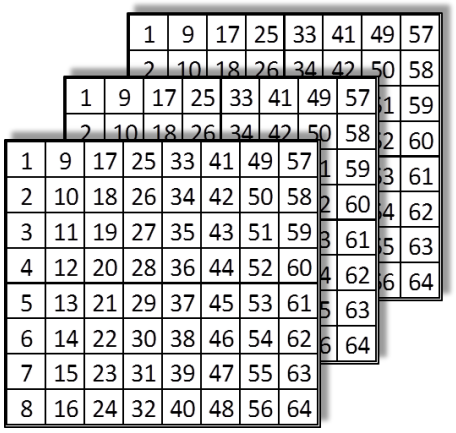

Figure 4-38: Burrows-Wheeler Transform Image Pixel Read Order

During the application of the BWT, the size of the transform matrix that is generated is exponentially greater than the input. For an image of size *m*x*n*, the matrix generated is $(mxn)^2$. This can be a memory strain during the processing of large images. To minimize memory use, a custom BWT algorithm is created that processes the image in a linear manner and avoids the creation of the transform matrix.

4.4.3.1 Non-Matrix Application of the Burrows-Wheeler Transform

The BWT can be deduced without creating the transform matrix of Table 4-12. This matrix consists of the sorted rotations of the original data set. The values in the last column of each row are the same values that precede the values of the first column of each row in the original data set order. The transform can be produced by identifying each data set element in ascending order and outputting the value that precedes the element in the original data set order.

Using the example difference coded data set from Table 4-5, the transform can be extracted as follows. The lowest value in the data set is found. The value that precedes this element is appended to the output, transform = [1].

Table 4-22 : Non-Matrix Burrows-Wheeler Transform, Step 1

1	3	1	7	1	3	1	1	3	1	7	1	3	1	0

The next lowest value is then found in the data.

Table 4-23 : Non-Matrix Burrows-Wheeler Transform, Step 2

1	3	1	7	1	3	1	1	3	1	7	1	3	1	0

When there are multiple occurrences of this value, the value that follows is considered. The lowest next value identifies which of the previously found elements should be selected. Of the possible element sets, (1,3), (1,7), (1,1), and (1,0), the element set (1,0) is selected. The value that precedes the element set, 3, is added to the output, transform = [1 3].

Table 4-24 : Non-Matrix Burrows-Wheeler Transform, Step 3

1	3	1	7	1	3	1	1	3	1	7	1	3	1	0

The next element set that should be selected is (1,1) and the 3 that precedes the element set is added to the output, transform = [1 3 3].

Table 4-25 : Non-Matrix Burrows-Wheeler Transform, Step 4

1	3	1	7	1	3	1	1	3	1	7	1	3	1	0

The next element set under consideration is (1,3). There are four instances of the element set (1,3). The values that follow each set are extracted until differences are found. The four available element sets are (1,3,1,7), (1,3,1,1), (1,3,1,7), and (1,3,1,0).

Table 4-26 : Non-Matrix Burrows-Wheeler Transform, Step 5

1	3	1	7	1	3	1	1	3	1	7	1	3	1	0

There is only one instance of the lowest ordered element set (1,3,1,0) and the element that precedes this element set is added to the output, transform = [1 3 3 7]. The next sequential element set is (1,3,1,1). There is only one occurrence of this element set and the value that precedes it is added to the output, transform = [1 3 3 7 7].

Table 4-27 : Non-Matrix Burrows-Wheeler Transform, Step 6

1	3	1	7	**1**	**3**	**1**	**1**	3	1	7	**1**	**3**	**1**	**0**

This process continues until all of the elements have been added to the transform. Table 4-28 lists the remaining element sets in sequential order. The values preceding each element set are added to the output in the same order. The final transform is [1 3 3 7 7 1 $ 3 3 1 1 1 1 1 1]. The linear method described above does not require that the transform matrix be created. The transform is created through an ordered search of the original data set.

Table 4-28 : Non-Matrix Burrows-Wheeler Transform, Steps 7-16

1	3	1	7	1	3	1	**1**	**3**	**1**	**7**	**1**	**3**	**1**	**0**
1	**3**	**1**	**7**	**1**	**3**	**1**	**1**	3	1	7	1	3	1	0
1	3	1	7	1	3	1	1	3	**1**	**7**	**1**	**3**	**1**	**0**
1	3	**1**	**7**	**1**	**3**	**1**	**1**	3	1	7	1	3	1	0
1	3	1	7	1	3	1	1	3	1	7	1	**3**	**1**	**0**
1	3	1	7	1	**3**	**1**	**1**	3	1	7	1	3	1	0
1	3	1	7	1	3	1	1	**3**	**1**	**7**	**1**	**3**	**1**	**0**
1	**3**	**1**	**7**	**1**	**3**	**1**	**1**	3	1	7	1	3	1	0
1	3	1	7	1	3	1	1	3	1	**7**	**1**	**3**	**1**	**0**
1	3	1	**7**	**1**	**3**	**1**	**1**	3	1	7	1	3	1	0

4.4.3.2 Non-Matrix Reversal of the Burrows-Wheeler Transform

The same memory constraints that exist in the application of the BWT exist for the reversal of the transform. A non-matrix reversal methodology is described in a summary, [55], derived from the textbook *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology* [56]. In this methodology, the transformed data set and the sorted values of the data set can be used to recover the original data in its original order.

Each element in the transformed data set is labeled with the place of the first occurrence of the element in the sorted data set. These values are listed in row “first” of Table 4-29. Then each symbol in the transformed data set is indexed. Row “index” of Table 4-29 indicates the ordinal occurrence of the symbol in the transform. The sum of the “first” row and the “index” row is captured in row “next” of Table 4-29. This row is a pointer to the value that follows the current element in the “transform” row. Using the values of the “next” row, the original data set in the original order can be extracted.

Table 4-29 : Non-Matrix Reversal Burrows-Wheeler Transform

transform	1	3	3	7	7	1	$	3	3	1	1	1	1	1	1
first	1	9	9	13	13	1	0	9	9	1	1	1	1	1	1
index	1	1	2	1	2	2	1	3	4	3	4	5	6	7	8
next	2	10	11	14	15	3	1	12	13	4	5	6	7	8	9

Beginning in the “$” column, the next element is listed in the “next” row as the element in column 1. This element of the transform is added to the output, original = [1]. Continuing from column 1, the next element is listed in the “next” row as the element in column 2. The transform element of column 2 is added to the output, original = [1 3]. Column 2 then points to the transform element of column 10. The output is extended to include this transform element, original = [1 3 1]. This process continues until the “$” symbol is found. The final output is original = [1 3 1 7 1 3 1 1 3 1 7 1 3 1 0]. The final end-of-line character can then be removed to reproduce the original data set and order.

4.4.4 Burrows-Wheeler Transform Performance

Initial review of the performance of the BWT is done using the Lenna image. The XOR difference data set and the baselined difference data set are transformed using BWT. The values of the transformed data sets are the same as their respective non-transformed data sets, but in a different order. Because the values do not change, the entropy of the transformed and untransformed data sets is the same. Entropy is not impacted by the transform. The change in data set order only impacts the correlation of the data set. The application of the BWT reduces correlation by 56% in the XOR difference data set and reduces correlation by 33% in the

baselined difference data set. The reduction in correlation should not impact the entropy coding of the data sets. However, the loss of correlation may impact dictionary coding which is dependent on patterns and directly related to correlation.

Table 4-30 : Burrows-Wheeler Transform of Difference Signal Correlation and Entropy

	Original	Subtraction Difference	XOR Difference	Burrows Wheeler Transform of XOR Difference	Baseline of XOR Difference	Burrows Wheeler Transform of Baseline of XOR Difference
Correlation	0.9863	0.5287	0.2136	0.0940	0.5314	0.3547
Entropy	7.6619	3.0796	4.5196	4.5196	4.0353	4.0353

A visual representation of correlation loss is provided in Figure 4-39 - Figure 4-42. In the Burrows-Wheeler Transformed images, the pixels take on a scattered gradient pattern with values generally increasing from left to right. The placement of larger pixel values in the transformed data set is not associated with the edges of the original image as in the non-transformed difference data set. The distribution of these larger values introduces the randomness that decreases correlation within the data set.

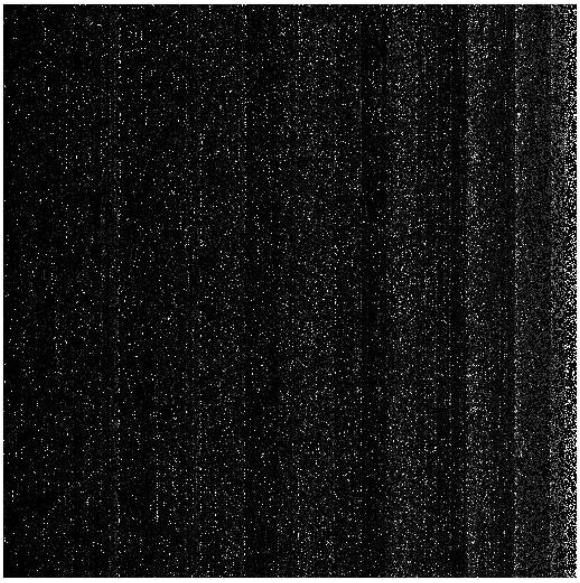

Figure 4-39: Burrows-Wheeler Transform of Lenna XOR Difference Signal

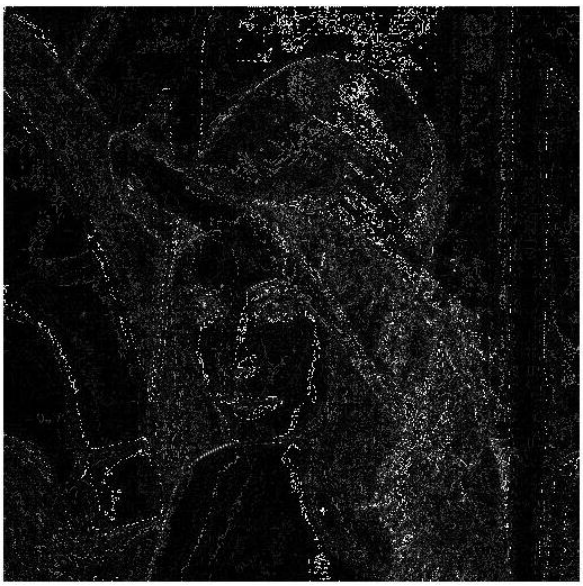

Figure 4-40: Lenna XOR Difference Signal

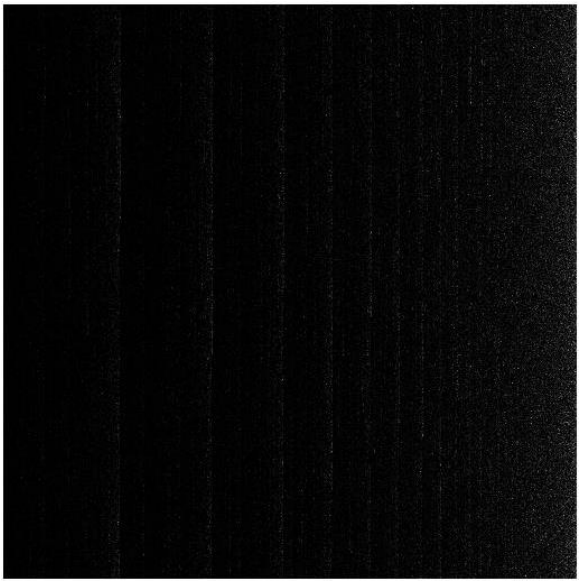

Figure 4-41: Burrows-Wheeler Transform of Baseline of Lenna XOR Difference Signal

Figure 4-42: Baseline of Lenna XOR Difference Signal

4.4.5 Burrows-Wheeler Transform Summary

The conversion of an image data set from a matrix to an array allows the BWT to be applied to images. Using non-matrix methodologies for application and reversal of the transform can help prevent memory consumption challenges. The transform has the ability to convert the many

patterns found in image data into long runs of symbols. The transformed image may be more compressible by dictionary coders than the non-transformed image.

4.5 LEMPEL-ZIV-WELCH DICTIONARY CODING

4.5.1 Review of Lempel-Ziv-Welch Dictionary Coding

LZW was first described in the “Lempel-Ziv-Welch Coding” section of the “Lossless Compression Algorithms” section in the “Lossless Image Compression Overview” chapter. The example previously provided focuses on the dictionary encoding of text string “ABRACADABRA”. The dictionary is initialized to have all possible 1-character symbols, A-Z. The string, “ABRACADABRA”, is the linear input of the LZW dictionary coder. Each character in the string is processed sequentially from left to right. As data is processed, new character sets are added to the dictionary. When previously processed characters are found in the input string, the dictionary index of the characters is used to represent the found characters. If data has repeated values, the dictionary may be able to represent multiple characters in a single index. Representing multiple characters in a single coded value can introduce data savings that allow the input to be represented in fewer bits than the bit-count of the original data.

When applying the LZW dictionary coder to images, the pixel values are used as the input. Continuous-tone, 8-bit, color images require a dictionary of 256 symbols. The dictionary should be initialized to include the values 0-255. The matrix of image data is processed the same as described in the “Burrows-Wheeler Transform of Images” section of the “Burrows-Wheeler Transform” section. An *m* row by *n* column by *p* dimensional image, $m \times n \times p$, is processed as *p* individual sets of $m \times n$ length data. The same linear pixel ordering of Figure 4-38, top-to-bottom and left-to-right, will generate a linear array of input from a matrix. The combined result of compressing each component represents the full image.

4.5.2 Dictionary Size Management

The LZW dictionary coding algorithm has the ability to create a dictionary of unlimited size. As dictionary size increases, compression performance diminishes. One reason for this loss of

performance is that the dictionary may have many unused entries. The unused entries take up space in the dictionary and cause the bit size of the index to grow. The original dictionary size recommended by the creators of the LZW dictionary coding algorithm is 4096 entries, [57]. This number limits the index size to 12 bits.

Dictionary growth must be managed to prevent loss of compression performance. One simple way to control the size of a dictionary is to clear entries 257-4096 and rebuild the dictionary when the size of the dictionary exceeds 4096. This can be done all at once with truncation or in a first-in-first-out order on individual dictionary entries. Another option for dictionary maintenance is to keep an access log of dictionary entries. The loop count can be captured in the access log for a dictionary entry whenever the dictionary entry is accessed. Any entries accessed outside of a given window can be dropped from the table. The simplest of all dictionary management options is to allow the dictionary to grow to a certain length and then stop adding entries. The method selected to manage the dictionary must also be repeatable by the decoder in order to ensure that the dictionaries are processed the same when decoding as when encoding.

4.5.3 Dictionary Search

Managing the size of a dictionary may also help the performance of the search operation. Each symbol and symbol set processed by the encoder requires a search of dictionary entries to find the index of the symbol set in the dictionary. This can be a costly operation in large dictionaries. Efficient search methods have been created that use an external indexes, hash tables, or binary trees. Those that do not wish to create external structures may simply use a full search of the dictionary. Each entry of the dictionary, from the beginning to the end, is compared to the search string until a match is found. None of the previously mentioned methods are used as a search method in the dissertation algorithm.

The search method used in the dissertation algorithm is based on a rolling column search of the dictionary. The contents of Figure 4-43 are related to the encoding of the diff values found in the example of Table 4-5. The first 12 values of the difference coded data set have been encoded and the next input begins at the value 3. The first column of the dictionary is searched to find all

occurrences of the value 3. The indexes of their locations are saved in the variable *current*, [4 10]. The next value in the input is 1. The second column is searched to find all occurrences of 1. Their indexes are saved in the variable *next*, [10 12 14 16]. The intersection of *current* and *next* are found. If the intersection is not empty, the result is saved in the variable *current*, [10]. The next value in the input is 0. The third column is searched to find all occurrences of the value 0. The indexes are saved in *next*, []. Again, the intersection of *current* and *next* is taken. If the intersection is not empty the process continues until no intersection is found or the intersection consists of only 1 value. If the intersection is empty, as in our example, the first index in *current* is output, 10, and the next value becomes the current value. The first index is selected because the first occurrence of any combination that is found is the combination without addition values. This method may require adjustment depending on the dictionary size management method that is used.

input:	1	3	1	7	1	3	1	1	3	1	7	1	3	1	0
	1	0					9	1	3						
	2	1					10	3	1						
	3	2					11	1	7						
	4	3					12	7	1						
	5	4					13	1	3	1					
	6	5					14	1	1						
	7	6					15	1	3	1	7				
	8	7					16	7	1	3					
output:	2	4	2	8	9	2	13	12							

Figure 4-43: Lempel-Ziv Dictionary Search

4.5.4 Two-Dimensional Dictionary Coding

The traditional application of LZW can only capture linear relations between pixels. In continuous-tone images there may be a lot of similarity around a target pixel. An example of a pixel with similar neighboring values is shown in Table 4-31. When the data is read top-to-bottom and left-to-right in a linear way, the first 20 entries of the resulting data set are [0 0 1 1 2 2 3 0 0 1 1 2 2 3 3 0 1 1 2 2]. Modifying the pattern used to read data may allow the data set to retain

some of the original correlation. Using the proximity read order of Table 4-32, values that are closer to the target pixel are placed closer to the front of the data set and those that are farther from the target pixel are placed towards the end of the data set. The first 20 entries of the data set when read using the proximity read order of Table 4-31 are [0 0 0 0 10 1 1 1 1 2 1 1 2 2 1 2 2 3 1]. The proximity sorted data set has more repetition and longer runs of values.

Table 4-31 : Highly Correlated Sample Block Data Set

X	0	0	1	1	2	2	3
0	0	1	1	2	2	3	3
0	1	1	2	2	3	3	3
1	1	2	2	3	3	3	2
1	2	2	3	3	3	2	2
2	2	3	3	3	2	2	2
2	3	3	3	2	2	2	1
3	3	3	2	2	2	1	1

Table 4-32 : Proximity Data Read Order

1	3	7	13	21	31	43	57
2	4	8	14	22	32	44	58
5	6	9	15	23	33	45	59
10	11	12	16	24	34	46	60
17	18	19	20	25	35	47	61
26	27	28	29	30	36	48	62
37	38	39	40	41	42	49	63
50	51	52	53	54	55	56	64

The reading of pixel values can be mapped in any sequence that is desired. This may be helpful when the directional change energy around a target pixel is known. In the previous example of Table 4-31, the pixels have a high amount of 45° change energy. A pixel mapping that orders the pixels along a diagonal direction can leverage the correlation associated with the change energy of the target pixel. Using the diagonal mapping of Table 4-33, the first 20 entries of the data set surrounding the target pixel are [0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2]. A sample pixel mapping for the -45° diagonal is given in Table 4-34. Aligning dictionary data set entries with the direction of the change energy at a target pixel leverages the correlation and repetition found in continuous-tone images.

Table 4-33 : 45° Diagonal Data Read Order

1	2	6	7	15	16	28	29
3	5	8	14	17	27	30	43
4	9	13	18	26	31	42	44
10	12	19	25	32	41	45	54
11	20	24	33	40	46	53	55
21	23	34	39	47	52	56	61
22	35	38	48	51	57	60	62
36	37	49	50	58	59	63	64

Table 4-34 : -45° Diagonal Data Read Order

1	10	24	36	46	54	60	64
9	2	12	26	38	48	56	62
23	11	3	14	28	40	50	58
35	25	13	4	16	30	42	52
45	37	27	15	5	18	32	44
53	47	39	29	17	6	20	34
59	55	49	41	31	19	7	22
63	61	57	51	43	33	21	8

4.5.5 Multi-Dictionary Coding

The original LZW dictionary coder is designed for use with a single dictionary. This dictionary begins with an entry for each possible symbol in the data set. When encoding 8-bit images, the dictionary must begin with 256 entries representing possible pixel values 0-255. As the image data set is processed, new pixel value sets are added to the dictionary. The resulting dictionary is customized to the image and built without prior knowledge. The same dictionary can be reproduced by the decoder without requiring the storage of additional data.

It is possible to use more than 1 dictionary in the encoding and decoding process. The selection from optional dictionaries usage during encoding must be based on the same context that is available to the decoder. The context of a target pixel can indicate the change energy near the target pixel. Edges near a target pixel may be identified using the context of the target pixel. The context of a target pixel may also provide an expected pixel value. Any of these 3 variables, change energy, edges, or expected pixel value, can be used to dictate dictionary selection.

A basic example of using 4 dictionaries can be made using expected pixel value. When dictionary coding, the expected value of the target pixel can be calculated using a predictor. The predicted value is used instead of the actual value because the actual value will not be available to the decoder. The range of expected values can be partitioned into 4 categories: 0-3, 4-15, 16-63, and 64-255. A dictionary for each category is available. Each dictionary must be populated with all possible pixel values 0-255. Separating dictionary usage by predicted value may cause the growth of the dictionary to be more refined. This design should cause each dictionary to be

populated with pixel value sets that are representative of the limited target values defined by the context. The use of multiple dictionaries is done without needing to save additional data.

4.5.6 Experimental Dictionary Coders

4.5.6.1 Dictionary Entry Formation

Several variations of the LZW dictionary coder are explored in this dissertation. The first set of variations is focused on dictionary entry formation. Data read in a linear way, top-to-bottom and left-to-right, represent the standard implementation of LZW, LZW Linear. The second dictionary entry formation is by block. Data values read in proximity order, as illustrated in Table 4-32, form the basis of the second variation of LZW, LZW Block. The last dictionary entry formation that is examined is adaptive word formation. The context of each target pixel is used to determine which of 4 directions a dictionary entry should be formed, LZW Adaptive. These 3 variations are the first to be compared.

Table 4-35 : Dictionary Entry Formation LZW Compression Ratios

	LZW Linear	LZW Block	LZW Adaptive
airplane.tiff	1.4942	**1.4947**	1.4805
barn.png	**1.3132**	1.3115	1.3032
bikes.png	1.1895	**1.1919**	1.1831
birds.png	**1.6907**	1.6905	1.6707
building_front.png	**1.1449**	1.1424	1.1365
buildings.png	**1.1484**	1.1437	1.1367
door.png	1.5129	**1.5156**	1.5009
flower_window.png	1.6336	**1.6397**	1.6242
girl.png	1.5432	**1.5438**	1.5247
hats.png	1.6863	**1.7012**	1.6836
house.tiff	**1.4571**	1.4558	1.4485
JellyBeans.tiff	2.5003	**2.5008**	2.4373
Lena.tiff	**1.4071**	1.4025	1.3905
lenna.jpg	**1.5485**	1.5424	1.5267
lighthouse.png	1.2835	**1.2882**	1.2823
man.tiff	1.3072	**1.3082**	1.2982
Mandrill.tiff	0.9986	**1.0003**	0.9978
mural_home.png	1.2900	**1.2911**	1.2802
Peppers.tiff	**1.3218**	1.3207	1.3127
sailboat_lake.tiff	**1.1803**	1.1802	1.1734
statue.png	1.4557	**1.4578**	1.4449
Average	1.4337	**1.4344**	1.4208

Each dictionary entry formation of the LZW dictionary coder was applied to the XOR difference data set of each image in the test set. The resulting compression ratios are provided in Table 4-35. The block entry formation of dictionary words provides the best compression. This formation generates only a marginal improvement, 0.05%, over the linear entry formation. The adaptive word formation also does not provide any improvements over the linear entry formation and decreases the average compression by 0.90%.

The Linear and Block LZW dictionary coding variations are extended to include multiple dictionaries. The dictionaries are based on values predicted by the context of each target pixel. The first variation uses the MED predictor to determine if target pixels are a member of a high edge, low edge, or neither. Image pixels that belong to a particular edge classification are then dictionary coded using the associated high edge, low edge, or non-edge dictionary. A more precise method of partitioning dictionaries is also explored that uses improved predictions. In the prediction component, the predictor option with the best performance is identified. Using this predictor, each target pixel value is estimated. The actual values of the target pixels are then dictionary coded using a dictionary assigned to the predicted value. An option for 4 dictionaries is made by creating a dictionary for each of 4 predicted value ranges: 0-3, 4-15, 16-63, and 64-255. An eight dictionary option is made by creating a dictionary for each of 8 predicted value ranges: 0-1, 2-3, 4-7, 8-15, 16-31, 32-63, 64-127, and 128-255. All multiple dictionary options, including 3-ditionary, 4-dictionary, and 8-dictionary, for the LZW dictionary coder are examined.

Table 4-36 : LZW Linear Compression Ratios

	LZW Linear 1D	LZW Linear 4D	LZW Linear 8D	LZW Linear 3D
airplane.tiff	**1.4942**	1.4818	1.4718	1.4623
barn.png	**1.3132**	1.2931	1.2810	1.2842
bikes.png	**1.1895**	1.1760	1.1629	1.1668
birds.png	**1.6907**	1.6773	1.6652	1.6565
building_front.png	**1.1449**	1.1278	1.1196	1.1252
buildings.png	**1.1484**	1.1289	1.1177	1.1256
door.png	**1.5129**	1.4929	1.4826	1.4790
flower_window.png	**1.6336**	1.6212	1.6052	1.5997
girl.png	**1.5432**	1.5306	1.5199	1.5054
hats.png	**1.6863**	1.6742	1.6611	1.6500
house.tiff	**1.4571**	1.4352	1.4241	1.4214
JellyBeans.tiff	2.5003	**2.5044**	2.4763	2.4354
Lena.tiff	**1.4071**	1.3896	1.3745	1.3791
lenna.jpg	**1.5485**	1.5369	1.5193	1.5186
lighthouse.png	**1.2835**	1.2688	1.2550	1.2529
man.tiff	**1.3072**	1.2990	1.2861	1.2871
Mandrill.tiff	**0.9986**	0.9773	0.9613	0.9669
mural_home.png	**1.2900**	1.2778	1.2638	1.2607
Peppers.tiff	**1.3218**	1.3038	1.2921	1.2964
sailboat_lake.tiff	**1.1803**	1.1620	1.1508	1.1507
statue.png	**1.4557**	1.4383	1.4268	1.4226
Average	**1.4337**	1.4189	1.4056	1.4022

4.5.6.2 LZW Linear

There are 4 variations of LZW dictionary coding using linear dictionary entry formation, LZW Linear. The variations are 1-dictionary (1D), 4-dictionary (4D), 8-dictionary (8D), and 3-dictionary (3D). The XOR-based difference data set of each image in the test set is compressed using each LZW Linear dictionary count variation. Compression ratios are listed in Table 4-36. The inclusion of additional dictionaries in LZW Linear compression only provides improved compression in one out of 21 images. The success of multiple dictionaries on the JellyBeans.tiff image relates to the high amount of correlation in the image. This image has a large amount of background which is very smooth and has little variance. When processing images with average correlation, the inclusion of additional dictionaries does not improve performance.

Figure 4-44: JellyBeans.tiff

Table 4-37 : LZW Block Compression Ratios

	LZW Block 1D	LZW Block 4D	LZW Block 8D	LZW Block 3D
airplane.tiff	**1.4947**	1.4813	1.4722	1.4619
barn.png	**1.3115**	1.2909	1.2794	1.2819
bikes.png	**1.1919**	1.1780	1.1639	1.1676
birds.png	**1.6905**	1.6774	1.6648	1.6540
building_front.png	**1.1424**	1.1256	1.1170	1.1229
buildings.png	**1.1437**	1.1239	1.1134	1.1219
door.png	**1.5156**	1.4951	1.4846	1.4803
flower_window.png	**1.6397**	1.6257	1.6083	1.6050
girl.png	**1.5438**	1.5322	1.5212	1.5061
hats.png	**1.7012**	1.6866	1.6719	1.6629
house.tiff	**1.4558**	1.4332	1.4220	1.4186
JellyBeans.tiff	2.5008	**2.5064**	2.4778	2.4308
Lena.tiff	**1.4025**	1.3844	1.3700	1.3730
lenna.jpg	**1.5424**	1.5295	1.5120	1.5110
lighthouse.png	**1.2882**	1.2732	1.2596	1.2562
man.tiff	**1.3082**	1.2992	1.2861	1.2872
Mandrill.tiff	**1.0003**	0.9773	0.9604	0.9666
mural_home.png	**1.2911**	1.2777	1.2643	1.2605
Peppers.tiff	**1.3207**	1.3019	1.2898	1.2933
sailboat_lake.tiff	**1.1802**	1.1608	1.1498	1.1496
statue.png	**1.4578**	1.4383	1.4281	1.4239
Average	**1.4344**	1.4190	1.4056	1.4017

4.5.6.3 LZW Block

LZW dictionary coding using block dictionary entry formation, LZW Block, also has 4 variations. The variations are 1-dictionary (1D), 4-dictionary (4D), 8-dictionary (8D), and 3-dictionary (3D).

Each LZW Block dictionary count variation is applied to the XOR-based difference data set of each image in the test set. The compression results are provided in Table 4-37. Again, the inclusion of additional dictionaries only improves the compression of the highly correlated JellyBeans.tiff image in Figure 4-44. Incorporating additional dictionaries does not improve the compression performance of the LZW Block coder.

4.5.6.4 LZW Adaptive

There are only 3 variations of LZW dictionary coding using adaptive dictionary entry formation, LZW Adaptive. The variations are 1-dictionary (1D), 4-dictionary (4D), and 8-dictionary (8D). Because the change energy of each target pixel is used in the formation of dictionary entries there is no value in coding with multiple dictionaries also based on change energy. The variation of LZW that uses 3 dictionaries is not considered. The XOR-based difference data set of each image in the test set is compressed using each of the 3 variations of dictionary count of the LZW Adaptive coder. The results of LZW Adaptive compression are listed in Table 4-38. The compression ratios listed are similar to the other dictionary entry formation variations. Only the JellyBeans.tiff image gains compression performance improvement from the inclusion of additional dictionaries.

Table 4-38 : LZW Adaptive Compression Ratios

	LZW Adaptive 1D	LZW Adaptive 4D	LZW Adaptive 8D
airplane.tiff	**1.4805**	1.4708	1.4591
barn.png	**1.3032**	1.2842	1.2721
bikes.png	**1.1831**	1.1712	1.1577
birds.png	**1.6707**	1.6565	1.6454
building_front.png	**1.1365**	1.1205	1.1130
buildings.png	**1.1367**	1.1176	1.1077
door.png	**1.5009**	1.4819	1.4732
flower_window.png	**1.6242**	1.6117	1.5944
girl.png	**1.5247**	1.5155	1.5062
hats.png	**1.6836**	1.6719	1.6589
house.tiff	**1.4485**	1.4260	1.4148
JellyBeans.tiff	2.4373	**2.4554**	2.4188
Lena.tiff	**1.3905**	1.3731	1.3595
lenna.jpg	**1.5267**	1.5145	1.4990
lighthouse.png	**1.2823**	1.2686	1.2554
man.tiff	**1.2982**	1.2900	1.2777
Mandrill.tiff	**0.9978**	0.9750	0.9589
mural_home.png	**1.2802**	1.2690	1.2566
Peppers.tiff	**1.3127**	1.2954	1.2827
sailboat_lake.tiff	**1.1734**	1.1563	1.1450
statue.png	**1.4449**	1.4282	1.4175
Average	**1.4208**	1.4073	1.3940

4.5.6.5 LZW Compression Performance

All prior LZW compression experiments have used the XOR difference data set of images as the input. However, there are 4 types of data sets available for compression. The first data set type is the XOR difference data set, XOR. The baselined version of the XOR difference data set, BSLN, is a second data set type available for compression. These two data set types, XOR and BSLN, can be transformed using the BWT. The BWT transform of XOR and BSLN form the third and fourth data set types, BWT XOR and BWT BSLN respectively. Each of the 4 data set types, XOR, BSLN, BWT XOR, and BWT BSLN, for each image in the image test set forms the full collection of image data sets. This collection of image data sets is used as an input for each of the 11 LZW variations. The compression ratios that result are listed in Appendix A.

The expanded test set provides a better foundation for analysis of the various LZW coders. The linear, single dictionary version of the LZW dictionary coder, LZW Linear 1D, provides the best compression ratio on 65% of the images in the full image data set. The block, single dictionary

version of the LZW dictionary coder is a distant second in performance. LZW Block 1D provides the best compression on 17% of the images in the full image data set. The 4-dictionary versions of the linear and block LZW coders, LZW Linear 4D and LZW Block 4D, provide best compression on 11% and 7% of the images in the full image data set respectively. The remaining 7 LZW dictionary coders do not provide optimal performance on any of the data sets.

Table 4-39 : Incidents of Optimal LZW Performance

LZW Coder	Count	Percent
LZW Linear 1D	55	65%
LZW Block 1D	14	17%
LZW Adaptive 1D	0	0%
LZW Linear 4D	9	11%
LZW Linear 8D	0	0%
LZW Linear 3D	0	0%
LZW Block 4D	6	7%
LZW Block 8D	0	0%
LZW Block 3D	0	0%
LZW Adaptive 4D	0	0%
LZW Adaptive 8D	0	0%

The average compression ratio for each data set type when compressed by each of the LZW dictionary coding variations is listed in Table 4-40. The linear, single dictionary version of the LZW dictionary coder, LZW Linear 1D, provides the best compression across all data set types. However, not all data set types need to be considered. The compression ratios of the BWT of both the XOR difference data set and baseline of the XOR difference data set are consistently lower than the compression ratios of their respective untransformed data sets. The BWT diminishes the performance of the LZW dictionary coders. The BWT XOR and BWT BSLN data set types will not be considered in the evaluation of the LZW dictionary coding variations. Also, the compression ratios of the baseline of the XOR difference data set type, BSLN, are consistently greater than compression ratios of the XOR difference data set type, XOR. The XOR data set type can also be excluded from evaluation of the LZW dictionary coding variations. The maximum compression ratio occurs when the baseline of the XOR difference data set type, BSLN, is the input. The best compressor for the baseline of the XOR difference data set is the linear, 4-dictionary variation of the LZW dictionary coder, LZW Linear 4D.

Table 4-40 : Average LZW Compression by Signal Type

	Entropy	Correlation	LZW Linear 1D	LZW Block 1D	LZW Adaptive 1D	LZW Linear 4D	LZW Linear 8D	LZW Linear 3D	LZW Block 4D	LZW Block 8D	LZW Block 3D	LZW Adaptive 4D	LZW Adaptive 8D
XOR	4.9853	0.2221	1.4337	**1.4344**	1.4208	1.4189	1.4056	1.4022	1.4190	1.4056	1.4017	1.4073	1.3940
BSLN	4.5222	0.4434	1.6147	1.6130	1.6031	***1.6235***	1.6095	1.5815	1.6224	1.6077	1.5795	1.6126	1.5991
BWT XOR	4.9853	0.1070	**1.3965**	1.3807	1.3735	1.3671	1.3492	1.3681	1.3506	1.3329	1.3519	1.3437	1.3263
BWT BSLN	4.5222	0.2798	**1.5815**	1.5670	1.5608	1.5707	1.5529	1.5510	1.5553	1.5378	1.5363	1.5492	1.5318
Average	4.7538	0.2631	***1.5066***	1.4988	1.4895	1.4951	1.4793	1.4757	1.4868	1.4710	1.4673	1.4782	1.4628

CHAPTER 5

ALGORITHM DESIGN

5.1 REVIEW OF ALGORITHM COMPONENTS

Initial experimentation included 5 components: Prediction, Difference Coding, Baselining, Transformation, and Dictionary coding. The Accurate Gradient Selection Predictor (AGSP), Gradient Edge Detection (GED) predictors, and linear predictors do not provide the best prediction in most cases and are not included in the final algorithm. The prediction component of the final algorithm includes the Median Edge Detection (MED) predictor, Gradient Adjusted Predictor (GAP), and Gradient Based Selection and Weighting (GBSW) predictor. The Burrows-Wheeler Transform (BWT) does not improve the compressibility of the difference coded data set and the BWT component is not included in the final algorithm. All 3-dictionary, 8-dictionary, and adaptive variations of LZW dictionary coder fail to provide optimal compression in any of the experimental cases. These 7 variations of LZW are not included in the final algorithm and only the 1-dictionary and 4-dictionary versions of the linear and block dictionary entry formations of the LZW dictionary coder are used in the algorithm. The final algorithm, Figure 5-2, is a subset of the original component set, Figure 5-1.

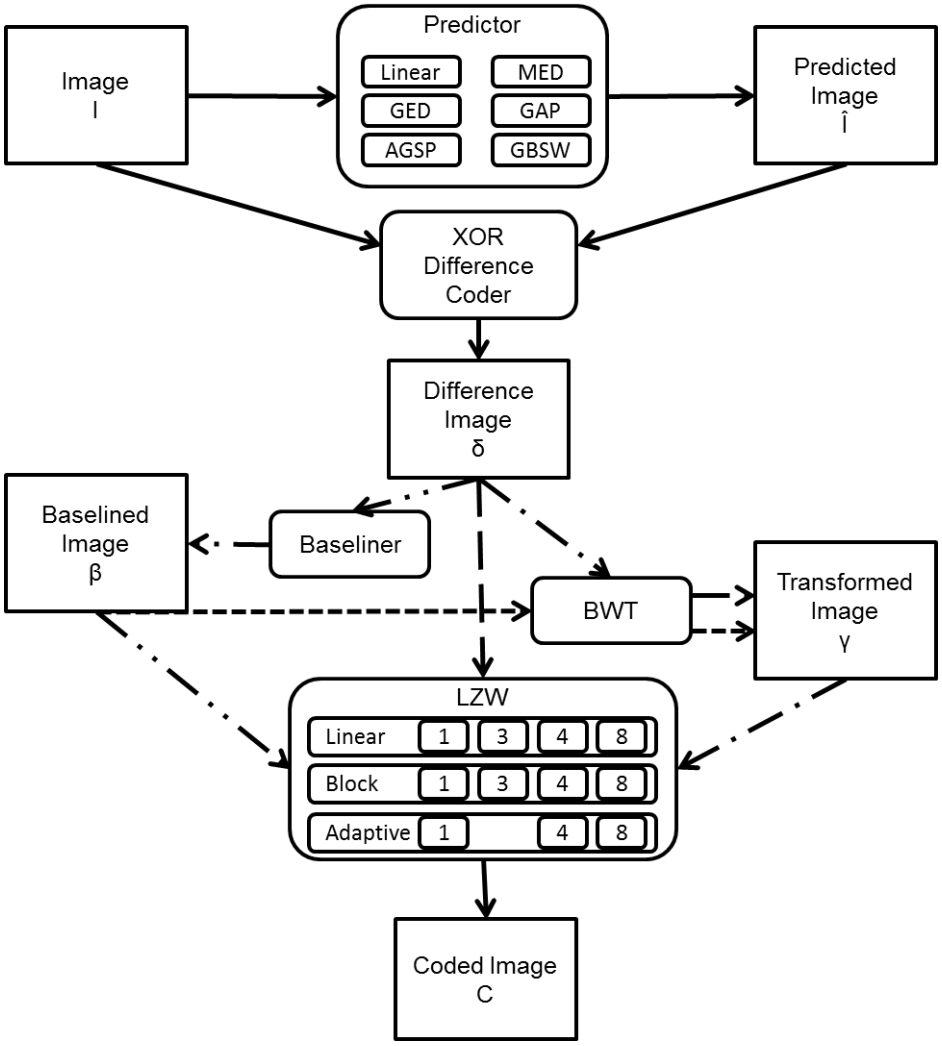

Figure 5-1: Component Experimentation

5.2 DISSERTATION ALGORITHM ENCODER

The dissertation algorithm is comprised of 4 components: Prediction, Difference Coding, Baselining, and Dictionary coding. The encoding process begins with prediction of the expected image values. Each of the predictors generates a data set of predicted pixel values. The predicted value set with the lowest mean absolute error is kept for further use and the identifier of the predictor is added to the output data set. The selected predicted value set, $\hat{I}$, is then XORed with the actual data set, I, to generate the difference coded data set, δ.

The difference coded data set, δ, is then adjusted through baselining. The resulting data set, β, will have a lower entropy and higher correlation than the original difference coded data set, δ. The baselined data set, β, is the input for the LZW dictionary encoder. Each of the LZW variations is applied to β and the variation that produces the best compression is retained. The identifier of the LZW variation is added to the output data set.

The encoding method described automatically determines the best performing variations of all components. In the prediction component, the best predictor is identified. The best variation of LZW coding is found in the dictionary coding component. The compressed data set, *C*, requires only 2 additional pieces of information for decoding. The identifier for the predictor and the identifier for the LZW variation can both be encoded using 2 sets of 2 bits respectively.

The dictionary coding component of the algorithm converts the image data from a fixed-length representation to a variable-length representation. The change in representation causes the dimensions of the image to be lost. If the size limitation of input images is 4096x4096, 2 sets of 12 bits will be required to represent the image row and column sizes respectively. The size of the image is added to the output data set followed by the compressed data set, *C*.

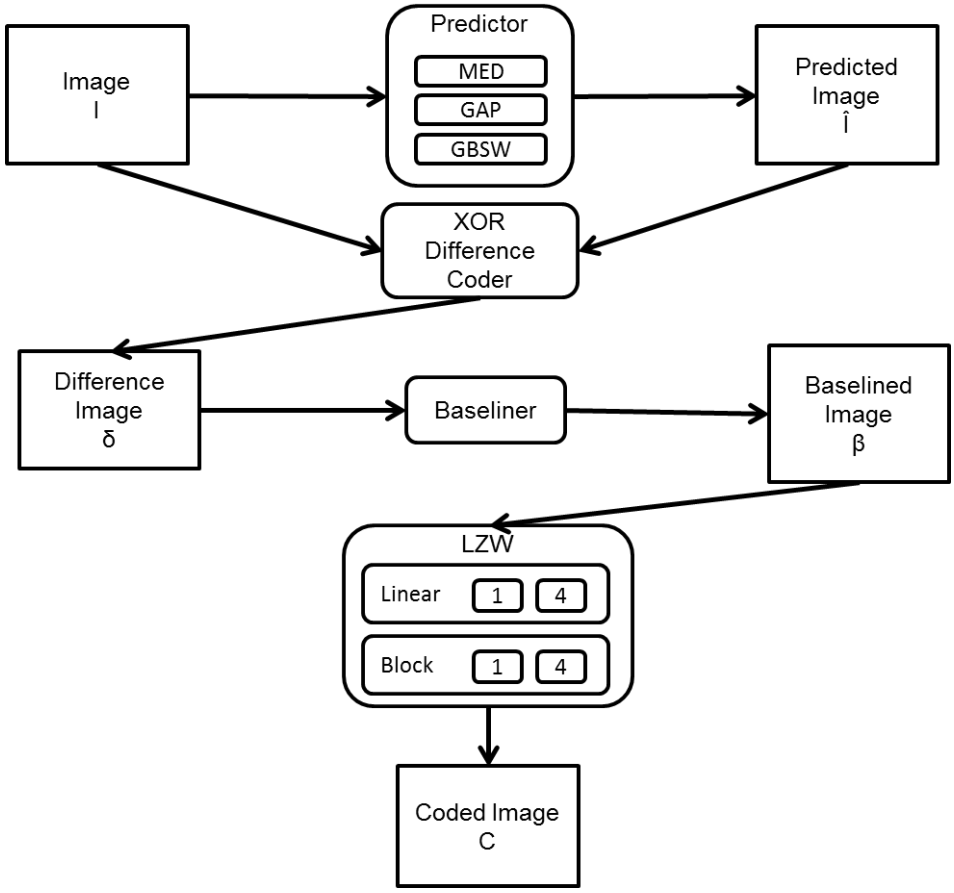

Figure 5-2: Dissertation Algorithm Encoder

5.3 DISSERTATION ALGORITHM DECODER

Prior to decoding the encoded image, the decoder must extract key data. The predictor is captured in the first 2 bits. The LZW encoder variation is listed in the next 2 bits. Image size is then provided in the next 24 bits. This gives the decoder all of the information needed to begin decoding the image.

The first step in decoding is recovering the baselined image, β. The LZW decoder selects the variation identified in the pre-decoding information. Only one decoder, the one previously identified is invoked. The resulting data set is reshaped to the original image size indicated in the pre-decoding information. Image β becomes the input for the next component of the decoder.

In the decoder, the baselining, predicting, and difference coding component operate as a joint component. The predictor identified in the pre-decoding information generates the original

prediction data set, $\hat{I}$. The baselined data set, β, is then converted into the XOR difference coded data set, δ. As difference image, δ, is processed the original image, I, is recreated. The pixel values of I are then used to create predictions for the following pixels. This cyclic process continues until all pixels of image I, are recovered.

The given algorithm is not symmetric. The encoder executes several predictors and LZW variations. The decoder only executes one of each. When encoding, the predictor is not combined with the difference coder and can use matrix operations to generate predictions and difference data sets. This helps to balance the time of encoding with the time of decoding because the several prediction and differencing matrix operations take a similar amount of time as the combined baselining-prediction-differencing sequential operations.

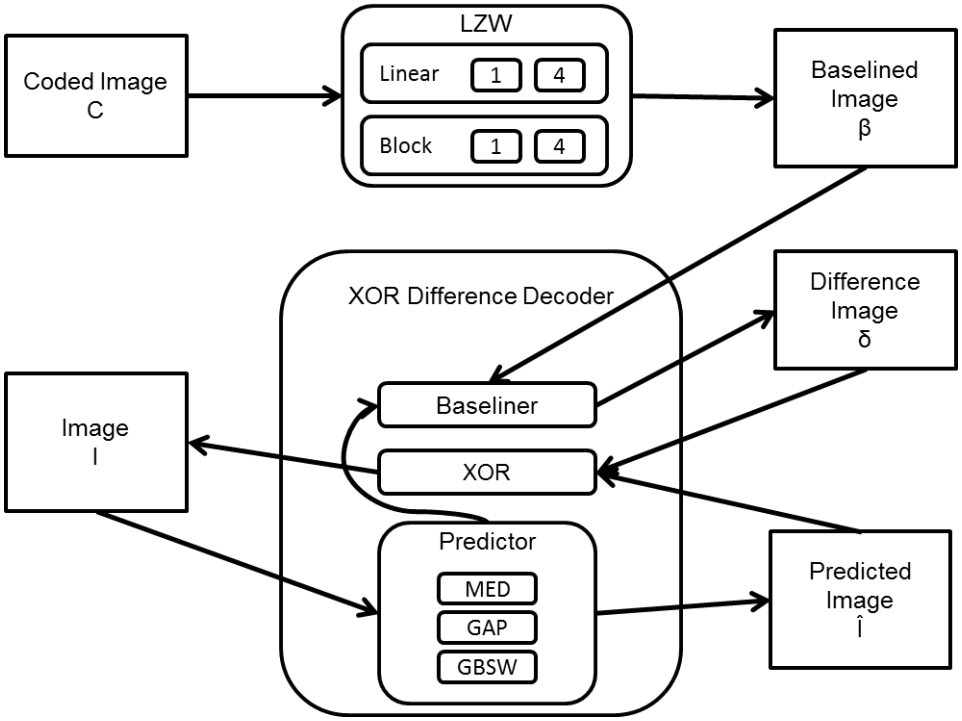

Figure 5-3: Dissertation Algorithm Decoder

5.4 DISSERTATION ALGORITHM DYNAMIC CODER

The object-oriented, component-based design of the dissertation algorithm also allows for more flexibility. There are 2 possible modes of operation. The first mode is a full mode. In this mode

each of the 4 components are used in the order described in the “Dissertation Algorithm Encoder” section and Figure 5-2. The second mode is a user directed. The variations of the core components of the algorithm, prediction and dictionary encoding, are specified by the user. The user directed mode allows for efficient execution of the algorithm because fewer variations of the predictor and dictionary coder are called.

CHAPTER 6

COMPRESSION PERFORMANCE ON VARIOUS IMAGE TYPES

The experiments described in the “Algorithm Components” chapter were used to guide the design of the final dissertation algorithm. This algorithm converts a fixed size image into a combination of fixed size metadata and variable-length encoded data sets. The metadata includes the row count, column count, predictor identifier, and LZW variation identifier. The compression ratio is defined using formula (42. It is the size of the original image divided by the full length of the combined metadata and encoded data set. The compression performance of the final dissertation algorithm is tested on 3 different types of images: natural, medical [58], and synthetic.

$$CR = \frac{bpp \times M \times N}{\left(b_m + b_n + b_p + b_c + b_d\right)};$$

$$\begin{aligned}
&Where\ bpp\ is\ the\ number\ of\ bits\ in\ each\ pixel,\\
&M\ is\ the\ number\ of\ rows\ in\ the\ image,\\
&N\ is\ the\ number\ of\ columns\ in\ the\ image,\\
&b_m\ is\ the\ number\ of\ bits\ required\ to\ represent\ the\ \max row\ count,\\
&b_n\ is\ the\ number\ of\ bits\ required\ to\ represent\ the\ \max column\ count,\\
&b_p\ is\ the\ number\ of\ bits\ required\ to\ represent\ the\ \text{identifier of the predictor},\\
&b_c\ is\ the\ number\ of\ bits\ required\ to\ represent\ the\ \text{identifier of the LZW coder},\\
&and\ b_d\ is\ the\ number\ of\ bits\ required\ to\ represent\ the\ coded\ data.
\end{aligned} \qquad (42)$$

6.1 COMPRESSION PERFORMANCE ON NATURAL IMAGES

Certain images have been longstanding test images in the field of image data compression. These images are members in the test sets of other image compression algorithms consistently. *Lena*, *Mandrill*, and *Peppers* are common images that have traditionally been used in the experimentation of existing compression methods. The test population of this study includes a total of 9 of these traditional images. The rest of the images in the natural image test set were randomly selected from a more current image database, the Kodak True-Color test image set. These images are a more accurate representation of the natural, vibrantly diverse images of

today. The combination of the traditional images and newer Kodak images, 21 images in total, form the natural image test set.

Table 6-1 : Natural Image Test Set

Traditional	Kodak
airplane.tiff	barn.png
house.tiff	bikes.png
JellyBeans.tiff	birds.png
Lena.tiff	building_front.png
lenna.jpg	buildings.png
man.tiff	door.png
Mandrill.tiff	flower_window.png
Peppers.tiff	girl.png
sailboat_lake.tiff	hats.png
	lighthouse.png
	mural_home.png
	statue.png

The dissertation algorithm is applied to each image in the natural image test set. The results are provided in Appendix B. The average compression ratio of natural images generated by the algorithm is 1.6310. Each variation of the 3 predictors and 4 compressors available in the algorithm is used to obtain optimum compression across the natural image test set. The most frequently used predictor is the Gradient Based Selection and Weighting predictor, GBSW. The most frequently used variation of the LZW dictionary coder is the 4-dictionary, linear entry formation, LZW Linear 4D.

6.2 COMPRESSION PERFORMANCE ON MEDICAL IMAGES

Medical images are often required to be compressed using lossless compression methodologies to preserve the information in the image when storing and transferring between usages. The current standard for storage and transmission along with print and display is Digital Imaging and Communications in Medicine (DICOM). DICOM images, which use the extension .DCM, have image data, but can also have other attribute data such as patient identifier, name, and birthdate.

For this study, the image attributes of a collection of DICOM images are used as the medical image test set. The medical images include brain, knee, and dental scans.

The medical image test set consists of 16-bit grayscale images. The dissertation algorithm is optimized for 8-bit images. Rather than expand the algorithm design to allow greater pixel bit-counts, the pixels of the medical images are partitioned into 2 components.

The first step in partitioning is to identify and use only the count of bits required. For example, if the maximum pixel value in the image is 3987, only 12 bits are required to represent the full range of values 0 to 3987. The second step of the process is to extract the most significant 8 bits of each 12-bit pixel. In our example, 3987 in decimal representation is 111110010011 in binary representation. The first 8 bits extracted are 11111001. This is equivalent to 249 when represented in decimal format. This value, 249, is saved as the pixel value of the first component. The second component consists of the remaining bits. The decimal value, 3, representing the remaining bits, 0011, is saved as the pixel value of the second component.

There are 2 benefits to using the previously described partitioning process. The first benefit is that the correlation of the image is retained in the first component. The second component is built from residual values and loses the correlation of the original image. The loss of correlation is counteracted by the second benefit. The second component of the image should have a reduced data set size. In the previously discussed example, the first 4 bits are discarded because there is no value greater than 4096. The next 8 bits are assigned to the first component. The second component formed from the remaining 4 bits of data. Only 16 of 256 values are possible. The truncation of the first 4 bits of pixel values automatically reduces the size of the second component by half.

Partitioning allows the 8-bit algorithm to process the 16-bit medical images effectively. The 2 components are processed sequentially. The image size is captured once as metadata in the compressed data output. The identifier of the optimal predictor and optimal LZW variation of the first component are then added to the encoded data. The LZW compressed image is then added

to the encoded data. The optimal predictor and optimal compressor of the second component are then added followed by the LZW compressed second component.

Table 6-2 : 16-Bit Compression Format

row count	column count	best predictor	best compressor	encoded data	best predictor	best compressor	encoded data
01-12	13-24	25-26	27-28	29-x	(x+1)-(x+2)	(x+3)-(x+4)	(x+5)-(x+y)

This format of the encoded data allows the image to be displayed in a lossy-to-lossless manner. The first and most significant component can be displayed independently. The image can then be completed by recovering the second component and adding the 2 components together.

The dissertation algorithm is applied to each image in the medical image test set. The results are provided in Appendix C. The average compression ratio of medical images generated by the dissertation algorithm is 3.0972. Each variation of the 3 predictors and 4 LZW dictionary coding variations available in the algorithm are used in the compression of the first component of images in the medical image test set. On the first component, the most frequently used predictor is the Gradient Based Selection and Weighting predictor, GBSW. The most frequently used variation of the LZW dictionary coder on the first component is the 4-dictionary, block entry formation, LZW Block 4D. The best predictor and LZW dictionary coder variation of the second component are the Gradient Based Selection and Weighting predictor, GBSW, and single dictionary, block entry formation, LZW Block 1D, respectively.

6.3 COMPRESSION PERFORMANCE ON SYNTHETIC IMAGES

The previous test image sets, the natural image test set and the medical image test set, have images that can be classified as real images. For this study, real images are defined as images of physical objects that are captured by a camera. Synthetic images are the opposite of real images. Synthetic images are images of objects not captured by camera that may or may not exist physically. Continuous-tone images can be created artificially as synthetic images. A third image test set, consisting of synthetic images, is included in the study.

Synthetic images can be created in various ways, but this study focuses on manually created images and computer generated images. Both of these image types, manually created and computer generated, are represented in animation. A collection of 24 animated images form the synthetic image test set. Images in the synthetic image test can vary in detail. Primitive images, such as bart_simpson.jpg in Figure 6-1, have lower variance and fewer colors. The creator of the animated series The Simpsons explains that “[it’s] still hand drawn and then it's digitally inked and painted” [59]. Advanced images, such as avatar.jpg in Figure 6-2, have higher variance and greater color depth. The avatar.jpg image features a computer generated character and background.

Figure 6-1: bart_simpson.jpg

Figure 6-2: avatar.jpg

The dissertation algorithm is applied to each image in the synthetic image test set. The results are provided in Table 6-3. The average compression ratio of synthetic images generated by the algorithm is 2.4869. As with the other image test sets, each variation of the 3 predictors and 4 LZW dictionary coding variations available in the algorithm is used on some image in the synthetic image test set. Although each option is used, the most frequently used predictor is the Gradient Based Selection and Weighting predictor, GBSW. The most frequently used variation of the LZW dictionary coder is the 4-dictionary, block entry formation, LZW Block 4D.

Table 6-3 : Synthetic Image Compression Ratios

Image	Best Predictor	Best Compressor	Compression Ratio
family_guy.jpg	3	4	2.3892
scooby.jpg	3	4	1.7577
wallace_gromit.jpg	3	4	3.0768
family_guy_1.jpg	3	4	2.2902
my_little_pony.png	1	3	10.9659
childhood_cartoons.png	1	1	1.4026
ratatouille.jpeg	3	3	1.6043
looney_tunes.jpg	3	4	1.4083
pooh.jpg	3	4	1.3771
fred_flintstone.jpg	2	3	1.4875
the_simpsons.jpg	3	4	2.2809
bugs_bunny.jpg	3	4	3.6352
bart_simpson.jpg	3	4	3.9268
classic_cartoons.jpg	3	2	1.2808
star_wars.jpg	3	4	2.2534
minion_banana.jpg	3	4	3.3458
avatar.jpg	2	3	1.6249
lion_king.jpg	3	4	1.6806
cartoon_network.jpg	3	2	1.2779
disney.jpg	3	4	1.4271
shrek.jpg	3	4	1.7364
despicable_me.jpg	3	4	1.9930
smurfs.jpg	3	4	3.2878
pixar.jpg	1	1	2.1754
Summary	3	4	2.4869

6.4 COMPRESSION PERFORMANCE ACROSS IMAGE TYPES

A summary of compression performance by image type is provided in Table 6-4. The algorithm provides an average compression ratio of 1.6310 on natural images. There is a 52% improvement in the average compression ratios of synthetic images as compared to natural images. The improvement is even greater when comparing natural image compression ratios to the compression ratios of medical images. Medical images have average compression ratios that are 90% better than the average compression ratios of natural images. The algorithm

successfully generates positive compression on all images and image types. There are no instances of compression ratios below 1.0.

Table 6-4 : Compression Ratios by Image Type

Image Type	Best Predictor	Best Compressor	Compression Ratio
Natural	3	3	1.6310
Medical (1st)	3	4	3.0972
Medical (2nd)	3	2	
Synthetic	3	4	2.4869

CHAPTER 7

ALGORITHM COMPARISON

The effectiveness of the dissertation algorithm can be determined through comparison with mainstream methodologies of lossless image compression. In the “Review of Related Literature” chapter, 3 different lossless compression algorithms are discussed: LOCO-I, Deflate, and CALIC. The LOCO-I algorithm is the lossless compressor used in the JPEG-LS image file format. File formats .ZIP and .PNG are both built upon the Deflate algorithm. The CALIC algorithm is not currently a part of any commonly used file types. The compression ratios of the dissertation algorithm will be compared to the compression ratios of various implementations of the 3 mainstream lossless compressors.

7.1 DISSERTATION ALGORITHM V. HUFFMAN

Preliminary research on the use of the exclusive-or operator as a difference coder was conducted to measure the direct impact of the exclusive-or operator on lossless compression. A set of images is difference coded. The original images and the difference coded images are then compressed using Huffman coding. Using the exclusive-or operator as a difference coder improves the compression of the Huffman coder by an average of 35%. The compression ratios of the combination of the exclusive-or operator and Huffman coding, XOR Huffman, are provided in Table 7-1. The same images are encoded using the dissertation algorithm. The compression results are included in Table 7-1 for comparison.

Table 7-1 : Dissertation Algorithm v. XOR Huffman and Huffman

Images	ORG Huffman	XOR Huffman	Dissertation Algorithm
airplane.tiff	1.1883	1.5666	**1.7430**
barn.png	1.1085	1.4349	**1.4684**
bikes.png	1.0819	1.2652	**1.3288**
birds.png	1.0991	1.7572	**1.9618**
building_front.png	1.1124	1.2312	**1.2525**
buildings.png	1.0458	1.2501	**1.2588**
door.png	1.4320	1.5989	**1.6626**
flower_window.png	1.1345	1.5245	**1.9175**
girl.png	1.0736	1.6328	**1.7533**
hats.png	1.1228	1.6342	**1.9913**
house.tiff	1.2264	1.4891	**1.6573**
JellyBeans.tiff	1.3913	2.4130	**2.9421**
Lena.tiff	1.0713	1.5327	**1.5942**
lenna.jpg	1.0406	1.6048	**1.8673**
lighthouse.png	1.1365	1.3573	**1.4365**
man.tiff	1.0598	1.4175	**1.4583**
Mandrill.tiff	1.0839	**1.1441**	1.0737
mural_home.png	1.1187	1.4146	**1.4426**
Peppers.tiff	1.0500	1.4360	**1.4833**
sailboat_lake.tiff	1.0644	**1.3182**	1.2934
statue.png	1.0979	1.5117	**1.6638**
Average	1.1305	1.5016	1.6310

The dissertation algorithm provides the best compression performance with an average compression ratio of 1.6310. The simplified combination of the exclusive-or operator and Huffman coding only generates an average compression ratio of 1.5016. The dissertation algorithm includes enhanced prediction in the difference coder, baselining prior to dictionary coding, and enhanced LZW dictionary coding. These modifications introduce an 8.61% performance increase over the XOR Huffman algorithm and a 44.28% performance increase over traditional Huffman coding.

7.2 DISSERTATION ALGORITHM V. CALIC & LOCO-I

The CALIC and LOCO-I algorithms were both created to be candidates for the international standardization effort on lossless compression of continuous-tone images [60]. The performance of the CALIC and LOCO-I algorithms is most commonly analyzed using the International Telecommunications Union's Telecommunications (UTI-T) Test Signals for Telecommunication Systems [61]. This test set includes a sampling of natural, medical, and combined graphics with text images. The dissertation algorithm is applied to several images of this UTI-T test set. The resulting compression ratios for the dissertation algorithm, CALIC, and LOCO-I are provided in Table 7-2.

Table 7-2 : Dissertation Algorithm v. CALIC and LOCO-I

Image	Dissertation Algorithm	LOCO-I	CALIC
BIKE	1.6294	2.2284	2.2857
CAFE	1.3946	1.6667	1.7058
WOMAN	1.7217	1.9185	1.9753
TOOLS	1.3125	1.5779	1.6162
BIKE3	1.5478	1.8307	1.8913
FINGER	1.1921	1.4210	1.4625
US	2.5551	2.9963	3.4188
CMPND1	5.1202	6.1538	6.4516
CMPND2	4.7647	5.9259	6.4516
AERIAL2	1.2648	1.9950	2.0888
AVERAGE	2.2503	2.7714	2.9348

The CALIC algorithm provides the best compression performance with an average compression ratio of 2.9348. The LOCO-I algorithm is the next best performing algorithm with an average compression ratio of 2.7714. The average compression ratio of the LOCO-I algorithm is only 4.57% below the average compression ratio of CALIC. The dissertation algorithm generates an average compression ratio below the CALIC and LOCO-I algorithms. The compression ratio of the dissertation algorithm performs 22.67% below the CALIC algorithm and 18.96% below the LOCO-I algorithm.

Although the dissertation algorithm performs lower than the CALIC and LOCO-I algorithms, the dissertation algorithm provides consistent performance. There is a .99 measure of correlation between the compression ratios of dissertation algorithm and compression ratios of both of the CALIC and LOCO-I Algorithms. The dissertation algorithm also provides positive compression on each of the images in the UTI-T test set. Just as the LOCO-I algorithm was selected as the foundation for the JPEG-LS algorithm due to its lower complexity, the dissertation algorithm may be preferred when low complexity is more important than maximum compression.

Table 7-3 : Correlation of Dissertation Algorithm, CALIC, and LOCO-I

	Dissertation Algorithm	LOCO-I	CALIC
Dissertation Algorithm	1		
LOCO-I	0.9943	1	
CALIC	0.9932	0.9986	1

7.3 DISSERTATION ALGORITHM V. PNG & ZIP

The CALIC and LOCO-I algorithms were selected for comparison with the dissertation algorithm because the CALIC and LOCO-I algorithms are high performing lossless image compression algorithms. The DEFLATE algorithm is another lossless compression algorithm that has been widely adopted. The .PNG and .ZIP image file formats use the DEFLATE algorithm to achieve compression. To compare the compression performance of the dissertation algorithm with DEFLATE algorithm, the same images in the original test set were converted to the .PNG and .ZIP image file formats. The bit count of each image when compressed using the dissertation algorithm, the .PNG file format, and .ZIP file format are provided in Table 7-4.

Table 7-4 : Dissertation Algorithm v. .ZIP and .PNG

Image	Dissertation Algorithm	ZIP	PNG
airplane	1.7430	1.3076	**1.8077**
barn	1.4684	1.2810	**1.6749**
bikes	1.3288	1.1728	**1.4747**
birds	1.9618	1.4236	**2.0401**
building_front	1.2525	1.2137	**1.4656**
buildings	1.2588	1.0915	**1.4525**
door	1.6626	1.5799	**1.9216**
flower_window	1.9175	1.4475	**1.9381**
girl	1.7533	1.4240	**1.8688**
hats	1.9913	1.5802	**2.0223**
house	1.6573	1.2101	**1.7711**
JellyBeans	**2.9421**	1.6005	2.2932
Lena	1.5942	1.1943	**1.7224**
lenna	1.8673	1.2290	**1.9147**
lighthouse	1.4365	1.3687	**1.6877**
man	1.4583	1.1687	**1.6128**
Mandrill	1.0737	1.0594	**1.2876**
mural_home	1.4426	1.2807	**1.6050**
Peppers	1.4833	1.1796	**1.6899**
sailboat_lake	1.2934	1.1591	**1.5211**
statue	1.6638	1.4102	**1.8569**
Average	1.6310	1.3039	**1.7442**

Although the .PNG and .ZIP file formats are both based on the DEFLATE algorithm, the compression ratios are not the same. The .PNG file format includes a prediction-based preprocessing step that reduces the entropy of the data set to be encoded and improves compressibility. The .PNG file format provides the best average compression ratio, 1.7442. The worst average compression ratio is generated by the .ZIP file format, 1.3039. The average compression ratio of the dissertation algorithm is between these competitors with a value of 1.6310. The dissertation algorithm is only 7.46% below the average compression performance of the .PNG file format and exceeds the .ZIP file format by 24.27%.

Table 7-5 : Correlation of Dissertation Algorithm, .ZIP, and .PNG

	Dissertation Algorithm	*ZIP*	*PNG*
Dissertation Algorithm	1		
ZIP	0.7570	1	
PNG	0.9355	0.8571	1

The compression performance of the dissertation algorithm is very close to performance of the .PNG file format. The .PNG file format and dissertation algorithm both include steps for decorrelation by prediction and dictionary encoding. The .ZIP file format does not include a decorrelation step prior to dictionary coding. In both the .PNG and .ZIP file format, the dictionary encoded data is further compressed using Huffman coding. The additional step of Huffman coding improves performance, but also increases complexity. The dissertation algorithm provides compression performance that is comparable to .PNG, but with a lower complexity that is comparable to .ZIP.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

8.1 DISSERTATION ALGORITHM DEVELOPMENT

The bulk of effort within this dissertation was given to the development of the dissertation algorithm. The fundamental goal was simply to decorrelate image data using the XOR filter prior to dictionary coding. There were several methods to implement both decorrelation and dictionary coding. The methodologies examined were selected from options already implemented in mainstream lossless compression algorithms as well as newer experimental research. The variety of methodology options allows a comparison of performance within each algorithm component and not just a comparison of the performance of the overall algorithm. This method of analysis is similar to a baker determining the best quality of individual cake ingredients rather than determining the best cake. The best algorithm component options can then combined to form a stronger overall algorithm.

The prediction component of the algorithm has a great impact on overall performance. Decorrelation with the XOR filter requires two elements: predicted values and actual values. The XOR filter captures the bit difference between the predicted and actual values. Ten methods of prediction were examined. Of the 10 methods, only 3 were found to be strong or significant performers. The Median Edge Detection predictor is a component of the LOCO-I algorithm which is embedded in the JPEG-LS algorithm. The predictor used in the CALIC algorithm is the Gradient Adjusted Predictor. The Gradient Based Selection and Weighting predictor is not currently incorporated in any mainstream lossless compression algorithm or image file format. The dissertation algorithm is able to dynamically leverage the best performing of the 3 predictors

based on the image being compressed. The prediction component can access the power of both existing and emerging predictors.

In the dissertation algorithm, the dictionary coding component is implemented using a series of Lempel-Ziv-Welch dictionary encoder variations. Of the 3 algorithms used for performance comparison, only the DEFLATE algorithm uses dictionary coding. The LOCO-I and CALIC algorithms use entropy coding instead. The dictionary encoder used in DEFLATE is the LZ77 dictionary coder which uses linear word formation and a single dictionary. Research in the dissertation introduced variance to dictionary coding in 2 ways: dictionary count and word entry formation. Variations of dictionary count included 1-dictionary, 3-dictionary, 4-dictionary, and 8-dictionary groupings. Experiments indicated that the 3-dictionary and 8-dictionary options did not provide better compression than the 1-dictionary and 4- dictionary options. Variations of word entry formation included linear, adaptive, and block. The adaptive word entry formation does not provide greater compression than either of the linear and block word entry formations. Eleven permutations of dictionary count and word entry formation were examined and only 4 options, linear 1-dictionary, linear 4-dictionary, block 1-dictionary, and block 4-dictionary, proved to be high performing.

Prior to dictionary coding, the data set is transformed to improve compressibility. Two transforms were examined: baselining and the Burrows-Wheeler transform. Baselining is a method created specifically for the dissertation algorithm. This method converts bit-plane differences which are generated by the XOR filter into an adjusted linear distance. This preserves some of the correlation lost in XOR filtering. The Burrows-Wheeler transform reorders data to group common values together. The BWT removes a significant portion of the correlation found in the original image data. Although decorrelation is of great value when entropy coding, some amount of correlation is a benefit when dictionary coding. Patterns in data are desirable for dictionary coding because the patterns provide more matches to dictionary entries and more efficient compression. Data with patterns will have greater correlation measures than random data. For

this reason, the goal of all steps prior to dictionary coding, should be entropy reduction rather than decorrelation.

Learnings from experimentation on each possible algorithm component were used to produce an efficient and effective algorithm. Prediction is selected from 3 high performing options: Median Edge Detection, Gradient Adjusted Prediction, and Gradient Based Selection and Weighting. The XOR Filter then uses the prediction and actual values to generate an entropy reduced data set. The data set is then baselined which further reduces entropy and restores some correlation. The baselined data set is then dictionary coded using the best performing of block and linear word entry and 1-dictionary and 4-dictionary count Lempel-Ziv-Welch dictionary coding variations. The final algorithm is avoids the use of higher complexity methods such as context-modeling, prediction error adjustments, or entropy coding while providing positive compression results.

8.2 DISSERTATION ALGORITHM PERFORMANCE

The performance of the dissertation algorithm is viewed from 3 perspectives: compression ratios, speed of execution, and complexity. The average compression ratios generated by the dissertation algorithm vary based on the image types that are being compressed. It is also evident that the speed of execution has a direct relationship to the compression ratio of the compressed image. The complexity of the dissertation algorithm is also an important factor when assessing the performance of the dissertation algorithm against other algorithms. Each of these 3 perspectives aid in the understanding of how the dissertation algorithm could fit in the landscape of algorithms designed for lossless compression of continuous-tone images.

The dissertation algorithm was applied to 3 different types of continuous-tone images. Each image type, natural, medical, and synthetic, has strong inter-pixel relationships. The medical image type differs from the other image types because of the consistency in the format of medical images. Medical images consist of a focused object on a solid background and there is little variation from this format. The lack of variation in medical images results in an image type that has high correlation more consistently than the other image types. The dissertation algorithm is

able to leverage the high correlation of medical images in both the exclusive-or filter and in the dictionary coder. The average compression ratio of medical images, 3.10, is 90% higher than the average compression ratios of natural images, 1.63.

Although the correlation of all image types in the continuous-tone image set is generally high, medical images typically have the greatest amount of correlation. Medical images offer the lowest change energy of all the image types. The images tend to be smooth, have fewer edges, and maintain more continuity between pixels. These traits promote greater performance in the prediction component of the algorithm.

The medical image type also differs in pixel bit-count from the natural and synthetic image types. The medical images processed are 16-bit images. The dissertation algorithm is optimized for 8-bit pixel images. Instead of expanding the dissertation algorithm to allow the 16-bits of data per pixel, medical images are partitioned into 2 data sets not greater than 8-bits per pixel each. This produces a highly correlated significant component and a lower entropy detail component. The most significant bits of the image have greater correlation than the full 16-bit image, which generates more patterns and improves compression. Most of the medical images in the test set do not require the use of 16 bits. The first 8 significant bits of information are extracted leaving a detail component that requires fewer bits. Having fewer bits in a data set decreases the range of data that must be represented which reduces the entropy of a data set. Reduced entropy allows for better compression. Due to the greater correlation of the significant component and reduced entropy of the detail component from image partitioning, the dissertation algorithm provides the best compression on medical images.

Compression performance is only one measure of algorithm performance. Another important factor is the speed of execution of an algorithm. An algorithm that provides superior compression, but requires extensive processing times, may not have practical real-world applications. It is possible to have predictable compression times in an algorithm when the processing steps of the algorithm are designed with a linear relationship to pixel counts. An example of a linear relationship in encoding is Huffman coding which individually replaces each

symbol in a data set with a code word. The dissertation algorithm uses dictionary coding which does not encode each symbol in the data individually, but can encode data in various sized symbol sets.

Like Huffman coding, dictionary coding converts fixed length symbol data to variable-length code words. Unlike Huffman coding, the LZW dictionary coder does not encode in a linear way. If the data set to be encoded is very random, there will be fewer occurrences of repeated patterns in the dictionary. This would require more entries to be inserted into the dictionary and require more code words to represent the data set. If the data set to be encoded is highly correlated, there will be many occurrences of repeated patterns in the dictionary. The repeated patterns allow multiple symbols in the data set to be represented using a single dictionary entry. The longer the set of symbols to be encoded the fewer dictionary entries that must be processed. This allows highly correlated data set to be processed in fewer transactions and in therefore in less time. For this reason, compression time is directly tied to correlation.

An additional algorithm performance measure is complexity. The dissertation algorithm has the linear complexity of O(n). There is a single set of operations executed for each pixel in an image. Algorithms with simple and low complexity design have the benefit of being able to be implemented on a wide variety of platforms. An algorithm with simple transactions, low memory usage, and simple data structures can be implemented on a platform with limited processing power. This flexibility allows for even greater adoptability of the algorithm.

The final dissertation algorithm includes 4 components. The first component, prediction, is the most complex of the 4 components. All of the predictors use addition and subtraction, with the GAP using division by powers of 2, and the GBSW using multiplication, division, and decimal values. The XOR filtering component is very low complexity and uses only exclusive-or logical bit operations. The last two components, baselining and dictionary coding, do not require mathematical transactions and use table look-up operations in processing. These 4 components are lower in complexity than Huffman which is lower in complexity than arithmetic coding.

The performance of the dissertation algorithm meets the needs of lossless image compression on continuous-tone images. Positive compression is made on each image in the test image set and the best compression is made on medical images. The speed of execution is directly tied to compressibility, but the dissertation algorithm is still an option for applications that do not require low processing speeds. The low complexity of the algorithm may also make it a preferred option when the desire for low complexity is a greater factor than the desire for low speed and low compression ratios. Although the dissertation algorithm does not outperform its competitors, it is able to provided comparable compression with lower complexity.

8.3 POSSIBLE EXPANSION OF THE DISSERTATION ALGORITHM

The algorithm presented in the dissertation can be expanded to provide enhancements that meet specific needs. In color image processing, inter-component pixel dependencies can be incorporated into the prediction component of the algorithm. Performance could be optimized by determining identifiers that have predictive relations to individual component performance. The success of algorithm performance on medical images can be extended by integrating the dissertation algorithm into the DICOM standard. Extending the algorithm into additional applications expands the impact of algorithm.

The individual color components of an image have correlation between each of the color components. Changes between pixels of one component frequently occur between pixels at the same location of the other components. This correlation may allow the prediction errors of one component to be used in predictions of the other components. This would introduce a prediction error correction step into the dissertation algorithm. CALIC and LOCO-I both use the context of a pixel to make prediction adjustments based on the errors that have previously occurred at similar contexts in the same component. This dissertation expansion uses the pixels of other components as context to make prediction adjustments. Any improvements in the prediction of image values will reduce the entropy of the data set to be compressed and therefore improve final compression ratios.

Another possible expansion of the dissertation would be to identify methods of predicting component performance based on image traits. Applying the dissertation algorithm to an expanded test image set would be required. Image traits such as entropy, correlation, and change energy should be captured for each image. The results of applying each option of the predictor and each option of dictionary encoder should be captured as well. The relationship between image traits and algorithm component performance can then be assessed for possible relationships. It may be possible to identify image conditions that predict the compression ratios of the dissertation algorithm. The predictions could be used to determine which algorithm components to apply and streamline the speed of algorithm execution.

The strong performance of the dissertation algorithm on medical images provides another possible expansion. The dissertation algorithm can be integrated into the DICOM file format. The DICOM file format can include multiple data sets in conjunction with an image pixel data set. Examples of additional data sets that are included in DICOM images include patient name, study name, creation date, and other information. The additional data sets are typically text data. The LZW dictionary coder was originally designed for text data. The LZW dictionary coder variants of the dissertation algorithm can easily be applied to the non-pixel image data sets. Applying the dissertation algorithm to medical image data and the associated attributes in a way that is consistent with DICOM file format guidelines could provide a new, low complexity variation of the medical imaging standard.

The 3 expansions described may help bridge the gap between the dissertation algorithm and its competitors. Information shared across color components of RGB images could allow for error corrections of prediction residuals and improve compression ratios. Identifiers of algorithm component performance could minimize processing and improve execution times. The dissertation algorithm also offers a potential low complexity variation of the DICOM standard. Each of the described expansions could allow the dissertation algorithm to have additional impact on the field of lossless image compression.

REFERENCES

[1] D. Wu and E. Tan, "Comparison of Lossless Image Compression Algorithms," *TENCON 99. Proceedings of the IEEE Region 10 Conference,* vol. 1, pp. 718-721, 1999.

[2] G. R. Robertson, M. F. Aburdence and R. J. Kozick, "Differential Block Coding of Bilevel Images," *Image Processing, IEEE Transactions on ,* vol. 5, no. 9, pp. 1368-1370, 1996.

[3] X. Wu and N. Memon, "Context-Based, Adaptive, Lossless Image Coding," *Communications, IEEE Transactions on ,* vol. 45, no. 4, pp. 437-444, 1997.

[4] X. Lu, M. Spear, K. Levitt and S. F. Wu, "Non-Uniform Entropy Compression for Uniform Energy Distribution in Wireless Sensor Networks," in *Sensor Technologies and Applications, 2008. SENSORCOMM '08. Second International Conference on* , Cap Esterel, 2008.

[5] S. I. Hussian, H. Javed, W. u. Rehman and F. N. Khalil, "CoXoH: Low Cost Energy Efficient Data Compression for Wireless Sensor Nodes using Data Encoding," in *Computer Networks and Information Technology (ICCNIT), 2011 International Conference on* , Abbottabad, 2011.

[6] J. Song, H. Yi, D. Hwang and S. Park, "A Compression Improvement Technique for Low-Power Scan Test Data," in *TENCON 2006. 2006 IEEE Region 10 Conference* , Hong Kong, 2006.

[7] B. Carpentieri, M. J. Weinberger and G. Seroussi, "Lossless Compression of Continuous-Tone Images," *Proceedings of the IEEE, Vol 88, No. 11 November,* pp. 1797-1809, 2000.

[8] "Wikipedia Halftone," 27 August 2014. [Online]. Available: http://en.wikipedia.org/wiki/Halftone. [Accessed 9 January 2015].

[9] P. Hao and Q. Shi, "Comparative Study of Color Transforms for Image Coding and Derivation of Integer Reversible Color Transform," in *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, Barcelona, 2000.

[10] I. Singh, P. Agathoklis and A. Antoniou, "Lossless Compression of Color Images using an Improved Integer-Based Nonlinear Wavelet Transform," in *Circuits and Systems, 1997. ISCAS '97., Proceedings of 1997 IEEE International Symposium on* , Hong Kong, 1997.

[11] W. Asdornwised, P. Sootaroj, P. Sa-nguansat, S. Jitapunkul and C. Chinrungrueng, "Context Partitions in Lossless Image Compression: An Enumerative Experiment," in *TENCON 2004. 2004 IEEE Region 10 Conference*, Chiang Mai, 2004.

[12] A. C. Hadenfeldt and K. Sayood, "Compression of Color-Mapped Images," in *Communications, 1993. ICC '93 Geneva. Technical Program, Conference Record, IEEE*

International Conference on , Geneva, 1994.

[13] A. Bruna, F. Vella, A. Buemi and S. Curti, "Predictive Differential Modulation for CFA Compression," in *Signal Processing Symposium, 2004. NORSIG 2004. Proceedings of the 6th Nordic*, Espoo, 2004.

[14] I. Avcibas, N. Memon, B. Sankur and K. Sayood, "A Successively Refinable Lossless Image-Coding Algorithm," *Communications, IEEE Transactions on ,* vol. 53, no. 3, pp. 445-452, 2005.

[15] M. Yang and N. Bourbakis, "An Overview of Lossless Digital Image Compression Techniques," in *Circuits and Systems, 2005. 48th Midwest Symposium on*, Cincinnati, 2005.

[16] C. TAŞKIN and S. K. SARIKOZ, "An Overview of Image Compression Approaches," in *Digital Telecommunications, 2008. ICDT '08. The Third International Conference on* , Bucharest, 2008.

[17] L. M. Kadlaskar and J. H. Pujar, "A New Lossless Method of Image Compression and Decompression using Huffman Coding Techniques," *Journal of Theoretical and Applied Information Technology,* vol. 15, no. 1, pp. 18-22, 2010.

[18] X. Wu, "An Algorithmic Study on Lossless Image Compression," in *Data Compression Conference, 1996. DCC '96. Proceedings*, Snowbird , 1996.

[19] X. Li and M. T. Orchard, "Edge-Directed Prediction for Lossless Compression of Natural Images," in *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on* , Kobe, 2001.

[20] R. Barequet and M. Feder, "SICLIC: a simple inter-color lossless image coder," in *Data Compression Conference, 1999. Proceedings. DCC '99*, Snowbird, 1999.

[21] X. Wu, W.-k. Choi and N. Memon, "Lossless Interframe Image Compression via Context Modeling," in *Data Compression Conference, 1998. DCC '98. Proceedings*, Snowbird, 1998.

[22] Y. Xue, M. M. Rees and X. Sheng, "A Simple Lossless Compression Method – Interval Number Method," in *Engineering in Medicine and Biology Society, 1998. Proceedings of the 20th Annual International Conference of the IEEE* , Hong Kong, 1998.

[23] M. Konecki, R. Kudelic and A. Lovrencic, "Efficiency of Lossless Data Compression," in *MIPRO, 2011 Proceedings of the 34th International Convention* , Opatija, 2011.

[24] J. Augustine, W. Feng and J. Jacob, "Logic Minimization Based Approach for Compressing Image Data," in *VLSI Design, 1995., Proceedings of the 8th International Conference on*, New Delhi, 1995.

[25] P. Mateu-Villarroya and J. Prades-Nebot, "Sequential Logic Compression of Images," in *Image Processing, 2001. Proceedings. 2001 International Conference on*, Thessaloniki, 2001.

[26] A. Spira and D. Malah, "Improved Lossless Compression of Color-Mapped Images by an Approximate Solution of the Traveling Salesman Problem," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on* , Salt

Lake City, 2001.

[27] X. Chen, S. Kwong and J.-f. Feng, "A New Compression Scheme for Color-Quantized Images," *Circuits and Systems for Video Technology, IEEE Transactions on ,* vol. 12, no. 10, pp. 904-908, 2002.

[28] A. Akimov, A. Kolesnikov and P. Franti, "Lossless Compression of Color Map Images by Context Tree Modeling," *Image Processing, IEEE Transactions on,* vol. 16, no. 1, pp. 114-120, 2007.

[29] N. Kuroki, T. Yamane and M. Numa, "Lossless Coding of Color Quantized Images Based on Pseudo Distance," in *Circuits and Systems, 2004. MWSCAS '04. The 2004 47th Midwest Symposium on*, Hiroshima, 2004.

[30] B. Koc and Z. Arnavut, "Application of Pseudo-distance to Lossless Coding of Color-Mapped Images," in *System of Systems Engineering (SoSE), 2011 6th International Conference on* , Albuquerque, 2011.

[31] S. A. Khayam, "The DiscreteCosine Transform (DCT): Theory and Application," Michigan State University, East Lansing, 2003.

[32] K. Cabeen and P. Gent, "Image Compression and the Discrete Cosine Transform," College of the Redwoods, Eureka.

[33] X. Hou, K. S. Gurumoorthy and A. Rajwade, "Color Image Compression using a Learned Dictionary of Pairs of Orthonormal Bases," in *Data Compression Conference (DCC), 2011*, Snowbird, 2011.

[34] B. Aiazzi, L. Alparone and S. Baronti, "Fuzzy Logic-Based Matching Pursuits for Lossless Predictive Coding of Still Images," *Fuzzy Systems, IEEE Transactions on,* vol. 10, no. 4, pp. 473-483, August 2002.

[35] N. Strobel, S. K. Mitra and B. Manjunath, "Progressive-Resolution Transmission and Lossless Compression of Color Images for Digital Image Libraries," in *Digital Signal Processing Proceedings, 1997. DSP 97., 1997 13th International Conference on* , Santorini, 1997.

[36] K. Ratakonda and N. Ahuja, "Segmentation Based Reversible Image Compression," in *Image Processing, 1996. Proceedings., International Conference on*, Lausanne, 1996.

[37] B. Aiazzi, P. Alba, L. Alparone and S. Baronti, "Lossless Compression of Multi/Hyper-Spectral Imagery Based on a 3-D Fuzzy Prediction," *Geoscience and Remote Sensing, IEEE Transactions on,* vol. 37, no. 5, pp. 2287-2294, 1999.

[38] T. Richter, "Evaluation of Floating Point Image Compression," in *Quality of Multimedia Experience, 2009. QoMEx 2009. International Workshop on*, San Diego, CA, 2009.

[39] R. Xu, S. N. Pattanaik and C. E. Hughes, "High Dynamic Range Image And Video Data Compression," University of Central Florida, Orlando, 2005.

[40] D. Marpe, H. Schwarz and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *Circuits and Systems for Video Technology,*

IEEE Transactions on, vol. 13, no. 7, pp. 620-636, 2003.

[41] Y. Zhu and Z. Xu, "Adaptive Context Based Coding for Lossless Color Image Compression," in *Computational Engineering in Systems Applications, IMACS Multiconference on* , Beijing, 2006.

[42] C. Serrano, B. Acha and R. M. Rangayyan, "Segmentation-Based Lossless Compression for Color Images," in *Image Analysis and Processing, 1999. Proceedings. International Conference on* , Venice, 1999.

[43] L. Zhe-lin, X. Qin-xiang, J. Li-jun and W. Shi-zi, "Full Color Cartoon Image Lossless Compression Based on Region Segment," in *Computer Science and Information Engineering, 2009 WRI World Congress on* , Los Angeles, 2009.

[44] M. J. Weinberger, G. Seroussi and G. Sapiro, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS," *Image Processing, IEEE Transactions on ,* vol. 9, no. 8, pp. 1309-1324, 2000.

[45] J. I. Larrauri, "A New Algorithm for Lossless Compression Applied to Two-Dimensional Static Images," in *Browse the Proceedings of the 6th International Conference on Communications and Information Technology (CIT '12), the 10th International Conference on Applied Electromagnetics, Wireless and Optical Communications (ELECTROSCIENCE '12) and the 3rd World Co*, Athens, 2012.

[46] A. E. Savakis, "Evaluation of Lossless Compression Methods for Gray Scale Document Images," in *Image Processing, 2000. Proceedings. 2000 International Conference on* , Vancouver, 2000.

[47] I. Blanes and J. Serra-Sagrista, "Clustered Reversible-KLT for Progressive Lossy-to-Lossless 3d Image Coding," in *Data Compression Conference, 2009. DCC '09.*, Snowbird, 2009.

[48] S. Yerva, S. Nair and K. Kutty, "Lossless Image Compression based on Data Folding," in *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on* , Chennai, 2011.

[49] H. Pan, W.-C. Siu and N.-F. Law, "Lossless Image Compression using Binary Wavelet Transform," *Image Processing, IET,* vol. 1, no. 4, pp. 353-362, 2007.

[50] P. Deutsch, "DEFLATE Compressed Data Format Specification version 1.3," Aladdin Enterprises, 1996.

[51] "Wikipedia Portable Network Graphics," 8 January 2015. [Online]. Available: http://en.wikipedia.org/wiki/Portable_Network_Graphics. [Accessed 8 January 2015].

[52] A. Avramic, "Lossless compression of medical images based on gradient edge detection," *Telecommunications Forum (TELFOR),* pp. 1199 - 1202, 2011.

[53] H. Tang and S.-i. Kamata, "A Gradient Based Predictive Coding for Lossless Image Compression," *IEICE Transactions on Information and,* Vols. E89-D, no. 7, 2006.

[54] J. Knezovic and M. Kovac, "Gradient Based Selective Weighting of Neighboring Pixels for Predictive Lossless Image Coding," *Information Technology Interfaces, 2003. ITI 2003.*

Proceedings of the 25th International Conference on, pp. 483-488, 2003.

[55] S. Batzoglou, "http://web.stanford.edu/class/cs262/presentations," 15 January 2015. [Online]. Available: http://web.stanford.edu/class/cs262/presentations/lecture4.pdf. [Accessed 14 March 2015].

[56] D. Gusfield, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, New York: Cambridge University Press, 1997.

[57] T. A. Welch, "A Technique for High-Performance Data Compression," *Computer,* vol. 17, no. 6, pp. 8--18, 1984.

[58] A. Rosset, L. Spadola and O. Ratib, "OsiriX: an open-source software for navigating in multidimensional DICOM images," J Digit Imaging, Los Angeles, 2004.

[59] M. Groening, Interviewee, *What You're Watching.* [Interview]. 20 February 2012.

[60] M. Barni, Document and Image Compression, CRC Press, 206.

[61] International Telecommunication Union, "Recommendation ITU-T T.24 : Standardized digitized image set," ITU-T Publications, Geneva, 2006.

[62] E. Janssen, E. Knapen, D. Reefman and F. Bruekers, "Lossless Compression of One-Bit Audio," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on* , Montreal, 2004.

APPENDIX A

Natural Image Compression Ratios

		LZW Linear 1D	LZW Block 1D	LZW Adaptive 1D	LZW Linear 4D	LZW Linear 8D	LZW Linear 3D	LZW Block 4D	LZW Block 8D	LZW Block 3D	LZW Adaptive 4D	LZW Adaptive 8D
airplane .tiff	XOR	1.4942	**1.4947**	1.4805	1.4818	1.4718	1.4623	1.4813	1.4722	1.4619	1.4708	1.4591
	BSLN	1.6989	1.6955	1.6868	***1.7065***	1.6819	1.6660	1.7023	1.6768	1.6631	1.6913	1.6705
	BWT XOR	**1.4555**	1.4387	1.4326	1.4290	1.4132	1.4286	1.4110	1.3958	1.4118	1.4049	1.3882
	BWT BSLN	**1.6559**	1.6398	1.6331	1.6438	1.6186	1.6249	1.6268	1.6025	1.6124	1.6196	1.5954
barn .png	XOR	**1.3132**	1.3115	1.3032	1.2931	1.2810	1.2842	1.2909	1.2794	1.2819	1.2842	1.2721
	BSLN	***1.4684***	1.4648	1.4605	1.4679	1.4552	1.4343	1.4653	1.4522	1.4304	1.4593	1.4475
	BWT XOR	**1.2833**	1.2737	1.2690	1.2558	1.2410	1.2597	1.2446	1.2298	1.2489	1.2392	1.2253
	BWT BSLN	**1.4427**	1.4322	1.4275	1.4271	1.4100	1.4126	1.4161	1.3992	1.4017	1.4122	1.3949
bikes .png	XOR	1.1895	**1.1919**	1.1831	1.1760	1.1629	1.1668	1.1780	1.1639	1.1676	1.1712	1.1577
	BSLN	1.3213	1.3221	1.3112	1.3275	1.3149	1.2939	***1.3288***	1.3175	1.2949	1.3193	1.3075
	BWT XOR	**1.1698**	1.1608	1.1579	1.1469	1.1315	1.1496	1.1362	1.1219	1.1408	1.1342	1.1184
	BWT BSLN	**1.2893**	1.2798	1.2764	1.2787	1.2615	1.2653	1.2681	1.2516	1.2557	1.2645	1.2486
birds .png	XOR	**1.6907**	1.6905	1.6707	1.6773	1.6652	1.6565	1.6774	1.6648	1.6540	1.6565	1.6454
	BSLN	1.9449	1.9428	1.9329	***1.9619***	1.9532	1.9101	1.9596	1.9501	1.9064	1.9486	1.9401
	BWT XOR	**1.6375**	1.6203	1.6123	1.6057	1.5858	1.6051	1.5863	1.5675	1.5867	1.5803	1.5601
	BWT BSLN	**1.9108**	1.8947	1.8889	1.9034	1.8908	1.8798	1.8879	1.8744	1.8630	1.8811	1.8667
building _front .png	XOR	**1.1449**	1.1424	1.1365	1.1278	1.1196	1.1252	1.1256	1.1170	1.1229	1.1205	1.1130
	BSLN	***1.2525***	1.2466	1.2405	1.2421	1.2268	1.2280	1.2364	1.2234	1.2232	1.2313	1.2176
	BWT XOR	**1.1249**	1.1169	1.1131	1.1008	1.0864	1.1056	1.0915	1.0781	1.0983	1.0878	1.0740
	BWT BSLN	**1.2300**	1.2216	1.2177	1.2123	1.1981	1.2088	1.2029	1.1890	1.1992	1.1986	1.1846
buildings .png	XOR	**1.1484**	1.1437	1.1367	1.1289	1.1177	1.1256	1.1239	1.1134	1.1219	1.1176	1.1077
	BSLN	***1.2588***	1.2536	1.2474	1.2511	1.2337	1.2339	1.2460	1.2309	1.2289	1.2398	1.2244
	BWT XOR	**1.1287**	1.1209	1.1176	1.1023	1.0888	1.1085	1.0934	1.0797	1.1013	1.0904	1.0766
	BWT BSLN	**1.2379**	1.2299	1.2265	1.2183	1.2005	1.2156	1.2100	1.1927	1.2074	1.2061	1.1888
door .png	XOR	1.5129	**1.5156**	1.5009	1.4929	1.4826	1.4790	1.4951	1.4846	1.4803	1.4819	1.4732
	BSLN	1.6567	1.6576	1.6435	1.6614	1.6489	1.6231	***1.6626***	1.6487	1.6235	1.6488	1.6368
	BWT XOR	**1.4762**	1.4629	1.4565	1.4451	1.4287	1.4448	1.4315	1.4152	1.4314	1.4255	1.4114
	BWT BSLN	**1.6282**	1.6154	1.6083	1.6111	1.5946	1.5949	1.5978	1.5803	1.5814	1.5918	1.5756
flower _window .png	XOR	1.6336	**1.6397**	1.6242	1.6212	1.6052	1.5997	1.6257	1.6083	1.6050	1.6117	1.5944
	BSLN	1.8882	1.8930	1.8752	1.9112	1.9028	1.8536	***1.9175***	1.9051	1.8564	1.8991	1.8902
	BWT XOR	**1.5888**	1.5703	1.5618	1.5544	1.5349	1.5563	1.5353	1.5143	1.5339	1.5269	1.5055
	BWT BSLN	**1.8413**	1.8245	1.8164	1.8376	1.8225	1.8083	1.8189	1.8039	1.7908	1.8113	1.7967
girl .png	XOR	1.5432	**1.5438**	1.5247	1.5306	1.5199	1.5054	1.5322	1.5212	1.5061	1.5155	1.5062
	BSLN	1.7234	1.7212	1.7090	***1.7436***	1.7320	1.6870	1.7424	1.7322	1.6870	1.7286	1.7181
	BWT XOR	**1.4911**	1.4713	1.4639	1.4598	1.4430	1.4582	1.4397	1.4231	1.4378	1.4322	1.4155
	BWT BSLN	**1.6827**	1.6645	1.6564	1.6773	1.6606	1.6492	1.6592	1.6427	1.6292	1.6506	1.6335
hats .png	XOR	1.6863	**1.7012**	1.6836	1.6742	1.6611	1.6500	1.6866	1.6719	1.6629	1.6719	1.6589
	BSLN	1.9570	1.9615	1.9514	1.9848	1.9757	1.9195	***1.9914***	1.9806	1.9243	1.9790	1.9691
	BWT XOR	**1.6376**	1.6176	1.6066	1.6004	1.5815	1.6043	1.5794	1.5592	1.5827	1.5673	1.5484
	BWT BSLN	**1.9180**	1.8991	1.8913	1.9170	1.9010	1.8837	1.8968	1.8810	1.8661	1.8896	1.8712
house .tiff	XOR	**1.4571**	1.4558	1.4485	1.4352	1.4241	1.4214	1.4332	1.4220	1.4186	1.4260	1.4148
	BSLN	***1.6574***	1.6534	1.6465	1.6490	1.6244	1.6177	1.6465	1.6157	1.6114	1.6397	1.6134
	BWT XOR	**1.4294**	1.4138	1.4042	1.3900	1.3725	1.3942	1.3726	1.3544	1.3765	1.3627	1.3439
	BWT BSLN	**1.6240**	1.6062	1.5991	1.5992	1.5681	1.5863	1.5786	1.5522	1.5692	1.5730	1.5439

JellyBeans .tiff	XOR	2.5003	2.5008	2.4373	2.5044	2.4763	2.4354	**2.5064**	2.4778	2.4308	2.4554	2.4188
	BSLN	2.8655	2.8543	2.8330	***2.9425***	2.9135	2.7929	2.9355	2.9022	2.7826	2.9176	2.8854
	BWT XOR	**2.3731**	2.2990	2.2589	2.3312	2.2952	2.3154	2.2598	2.2297	2.2423	2.2250	2.1981
	BWT BSLN	2.7746	2.7174	2.6957	**2.7942**	2.7583	2.7023	2.7373	2.7069	2.6475	2.7102	2.6851
Lena .tiff	XOR	**1.4071**	1.4025	1.3905	1.3896	1.3745	1.3791	1.3844	1.3700	1.3730	1.3731	1.3595
	BSLN	1.5937	***1.5942***	1.5843	1.5924	1.5792	1.5675	1.5908	1.5770	1.5664	1.5823	1.5707
	BWT XOR	**1.3731**	1.3619	1.3572	1.3444	1.3241	1.3480	1.3317	1.3108	1.3363	1.3262	1.3065
	BWT BSLN	**1.5745**	1.5649	1.5599	1.5600	1.5436	1.5490	1.5493	1.5324	1.5384	1.5450	1.5291
lenna .jpg	XOR	**1.5485**	1.5424	1.5267	1.5369	1.5193	1.5186	1.5295	1.5120	1.5110	1.5145	1.4990
	BSLN	1.8093	1.8082	1.7833	***1.8325***	1.8112	1.7732	1.8302	1.8102	1.7687	1.8112	1.7933
	BWT XOR	**1.5029**	1.4845	1.4780	1.4764	1.4553	1.4750	1.4573	1.4364	1.4578	1.4512	1.4303
	BWT BSLN	1.7417	1.7244	1.7160	**1.7427**	1.7199	1.7102	1.7229	1.6986	1.6924	1.7165	1.6915
lighthouse .png	XOR	1.2835	**1.2882**	1.2823	1.2688	1.2550	1.2529	1.2732	1.2596	1.2562	1.2686	1.2554
	BSLN	1.4300	1.4273	1.4221	***1.4365***	1.4260	1.3953	1.4353	1.4252	1.3931	1.4298	1.4195
	BWT XOR	**1.2541**	1.2410	1.2347	1.2257	1.2089	1.2280	1.2118	1.1959	1.2149	1.2071	1.1908
	BWT BSLN	**1.4006**	1.3884	1.3828	1.3885	1.3735	1.3711	1.3775	1.3609	1.3585	1.3723	1.3568
man .tiff	XOR	1.3072	**1.3082**	1.2982	1.2990	1.2861	1.2871	1.2992	1.2861	1.2872	1.2900	1.2777
	BSLN	1.4509	***1.4516***	1.4445	1.4486	1.4408	1.4321	1.4497	1.4411	1.4332	1.4425	1.4345
	BWT XOR	**1.2822**	1.2751	1.2717	1.2623	1.2465	1.2644	1.2537	1.2381	1.2567	1.2504	1.2353
	BWT BSLN	**1.4348**	1.4284	1.4260	1.4218	1.4107	1.4171	1.4145	1.4028	1.4112	1.4125	1.4001
Mandrill .tiff	XOR	0.9986	**1.0003**	0.9978	0.9773	0.9613	0.9669	0.9773	0.9604	0.9666	0.9750	0.9589
	BSLN	1.0729	***1.0737***	1.0715	1.0606	1.0516	1.0426	1.0613	1.0517	1.0429	1.0593	1.0490
	BWT XOR	**0.9890**	0.9834	0.9829	0.9629	0.9443	0.9598	0.9571	0.9400	0.9548	0.9564	0.9389
	BWT BSLN	**1.0642**	1.0594	1.0569	1.0433	1.0302	1.0376	1.0370	1.0240	1.0311	1.0354	1.0228
mural _home .png	XOR	1.2900	**1.2911**	1.2802	1.2778	1.2638	1.2607	1.2777	1.2643	1.2605	1.2690	1.2566
	BSLN	1.4293	1.4316	1.4184	1.4405	1.4296	1.3933	***1.4426***	1.4317	1.3952	1.4305	1.4216
	BWT XOR	**1.2528**	1.2393	1.2354	1.2278	1.2128	1.2283	1.2140	1.1986	1.2145	1.2092	1.1925
	BWT BSLN	**1.3904**	1.3767	1.3708	1.3803	1.3667	1.3616	1.3668	1.3524	1.3465	1.3610	1.3472
Peppers .tiff	XOR	**1.3218**	1.3207	1.3127	1.3038	1.2921	1.2964	1.3019	1.2898	1.2933	1.2954	1.2827
	BSLN	***1.4833***	1.4782	1.4738	1.4799	1.4658	1.4596	1.4753	1.4609	1.4554	1.4704	1.4569
	BWT XOR	**1.2966**	1.2876	1.2828	1.2674	1.2507	1.2731	1.2594	1.2408	1.2627	1.2549	1.2382
	BWT BSLN	**1.4711**	1.4616	1.4587	1.4586	1.4393	1.4481	1.4496	1.4315	1.4400	1.4469	1.4287
sailboat _lake .tiff	XOR	**1.1803**	1.1802	1.1734	1.1620	1.1508	1.1507	1.1608	1.1498	1.1496	1.1563	1.1450
	BSLN	***1.2934***	1.2901	1.2867	1.2900	1.2801	1.2659	1.2866	1.2764	1.2631	1.2829	1.2726
	BWT XOR	**1.1603**	1.1514	1.1479	1.1321	1.1155	1.1327	1.1213	1.1054	1.1234	1.1185	1.1027
	BWT BSLN	**1.2757**	1.2684	1.2640	1.2589	1.2463	1.2510	1.2481	1.2357	1.2424	1.2455	1.2323
statue .png	XOR	1.4557	**1.4578**	1.4449	1.4383	1.4268	1.4226	1.4383	1.4281	1.4239	1.4282	1.4175
	BSLN	1.6526	1.6525	1.6420	***1.6638***	1.6519	1.6212	1.6636	1.6517	1.6195	1.6531	1.6422
	BWT XOR	**1.4195**	1.4052	1.3985	1.3892	1.3733	1.3911	1.3740	1.3568	1.3760	1.3680	1.3521
	BWT BSLN	**1.6239**	1.6093	1.6051	1.6105	1.5951	1.5929	1.5953	1.5791	1.5772	1.5895	1.5743
Average	XOR	1.4337	**1.4344**	1.4208	1.4189	1.4056	1.4022	1.4190	1.4056	1.4017	1.4073	1.3940
	BSLN	1.6147	1.6130	1.6031	***1.6235***	1.6095	1.5815	1.6224	1.6077	1.5795	1.6126	1.5991
	BWT XOR	**1.3965**	1.3807	1.3735	1.3671	1.3492	1.3681	1.3506	1.3329	1.3519	1.3437	1.3263
	BWT BSLN	**1.5815**	1.5670	1.5608	1.5707	1.5529	1.5510	1.5553	1.5378	1.5363	1.5492	1.5318

APPENDIX B

Natural Image Compression Ratios

Image	Best Predictor	Best Compressor	Compression Ratio
airplane.tiff	GAP	LZW Linear 4D	1.7430
barn.png	GAP	LZW Linear 1D	1.4684
bikes.png	GBSW	LZW Block 4D	1.3288
birds.png	GBSW	LZW Linear 4D	1.9618
building_front.png	MED	LZW Linear 1D	1.2525
buildings.png	MED	LZW Linear 1D	1.2588
door.png	GBSW	LZW Block 4D	1.6626
flower_window.png	GBSW	LZW Block 4D	1.9175
girl.png	GAP	LZW Linear 4D	1.7533
hats.png	GBSW	LZW Block 4D	1.9913
house.tiff	MED	LZW Linear 1D	1.6573
JellyBeans.tiff	GBSW	LZW Linear 4D	2.9421
Lena.tiff	GBSW	LZW Block 1D	1.5942
lenna.jpg	MED	LZW Linear 4D	1.8673
lighthouse.png	GAP	LZW Linear 4D	1.4365
man.tiff	GAP	LZW Block 1D	1.4583
Mandrill.tiff	GBSW	LZW Block 1D	1.0737
mural_home.png	GBSW	LZW Block 4D	1.4426
Peppers.tiff	GBSW	LZW Linear 1D	1.4833
sailboat_lake.tiff	GBSW	LZW Linear 1D	1.2934
statue.png	GBSW	LZW Linear 4D	1.6638
Mode	GBSW	LZW Linear 4D	

APPENDIX C

Medical Image Compression Ratios

		First Component		Second Component		
	Image	Best Predictor	Best Compressor	Best Predictor	Best Compressor	Compression Ratio
KNEE1	IM-0001-0001.dcm	1	4	3	2	2.0080
	IM-0001-0006.dcm	1	4	3	2	2.1858
	IM-0001-0011.dcm	1	4	3	2	2.1022
	IM-0001-0016.dcm	1	4	3	2	2.1740
KNEE2	IM-0001-0001.dcm	1	2	3	2	1.8764
	IM-0001-0006.dcm	1	2	3	3	1.8494
	IM-0001-0011.dcm	1	2	3	2	1.8821
	IM-0001-0016.dcm	1	2	3	2	1.9585
KNEE3	IM-0001-0001.dcm	1	2	3	2	1.6215
	IM-0001-0006.dcm	1	2	3	2	1.5796
	IM-0001-0011.dcm	1	2	3	2	1.6041
	IM-0001-0016.dcm	1	2	3	2	1.7031
KNEE4	IM-0001-0001.dcm	3	3	3	2	2.4107
	IM-0001-0006.dcm	3	3	3	2	2.3625
	IM-0001-0011.dcm	1	4	3	1	2.4097
	IM-0001-0016.dcm	3	4	3	1	2.2425
KNEE5	IM-0001-0001.dcm	1	1	3	1	1.9436
	IM-0001-0006.dcm	1	1	3	1	1.7981
	IM-0001-0011.dcm	1	1	3	1	1.8223
	IM-0001-0016.dcm	1	3	3	2	1.8361
KNEE6	IM-0001-0001.dcm	2	3	3	2	1.7812
	IM-0001-0006.dcm	1	1	3	2	1.6641
	IM-0001-0011.dcm	3	1	3	2	1.6193
	IM-0001-0016.dcm	1	1	3	2	1.6762
BRAIN1	IM-0001-0001.dcm	2	3	1	1	4.5922
	IM-0001-0006.dcm	3	3	3	2	3.4197
	IM-0001-0011.dcm	3	3	3	2	3.1033
	IM-0001-0016.dcm	3	3	3	2	3.0073

BRAIN2	IM-0001-0001.dcm	2	3	3	1	6.6105
	IM-0001-0006.dcm	3	4	3	2	4.2846
	IM-0001-0011.dcm	3	4	3	2	3.8373
	IM-0001-0016.dcm	3	4	3	2	3.7766
BRAIN3	IM-0001-0001.dcm	2	3	3	2	5.4901
	IM-0001-0006.dcm	3	4	3	2	3.6867
	IM-0001-0011.dcm	3	4	3	2	3.3474
	IM-0001-0016.dcm	3	4	3	2	3.2681
BRAIN4	IM-0001-0001.dcm	3	3	3	2	5.5836
	IM-0001-0006.dcm	3	4	3	2	4.5737
	IM-0001-0011.dcm	3	4	3	2	3.8455
	IM-0001-0016.dcm	3	4	3	2	3.5414
BRAIN5	IM-0001-0001.dcm	1	3	3	1	6.7563
	IM-0001-0006.dcm	3	4	3	2	4.7404
	IM-0001-0011.dcm	3	4	3	2	4.2769
	IM-0001-0016.dcm	3	4	3	2	4.1276
BRAIN6	IM-0001-0001.dcm	1	3	3	2	5.6754
	IM-0001-0006.dcm	3	4	3	2	4.1713
	IM-0001-0011.dcm	3	4	3	2	3.7824
	IM-0001-0016.dcm	3	4	3	2	3.6593
BRAIN7	IM-0001-0001.dcm	2	3	3	3	6.3744
	IM-0001-0006.dcm	2	3	3	1	4.0917
	IM-0001-0011.dcm	3	4	3	1	3.5817
	IM-0001-0016.dcm	3	2	3	2	3.6565
DENTAL	IM-0001-0001.dcm	2	3	3	2	1.6333
	IM-0001-0006.dcm	2	3	3	1	1.6179
	IM-0001-0011.dcm	2	3	3	2	1.6068
	IM-0001-0016.dcm	2	3	3	2	1.6125
Summary		3	4	3	2	3.0972

CURRICULUM VITA

NAME Takiyah K. Cooper

ADDRESS 1363 Darlene Circle
Radcliff, KY 40160

EDUCATION

B.S., Computer Engineering and Computer Science
University of Louisville
2000-2004

M. Eng., Computer Engineering and Computer Science
University of Louisville
2004-2005

WORK EXPERIENCE

Eli Lilly & Co.	
Manager-IT	2015 - present
Assoc. Consultant	2012 - 2015
Sr. Analyst	2009 - 2012
Humana, Inc.	
Oracle Applications Engineer	2006 – 2009
Motorola	
Software Engineer	2005
United Parcel Service	
Business Systems Analyst	2002 - 2003
United States Army	
Unit Administrative Assistant	2000 - 2006
Petroleum Supply Specialist	1998 - 2000