SEARCH AND OPTIMIZATION ALGORITHMS FOR BINARY IMAGE COMPRESSION

By

REETU HOODA

A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in
The Department of Electrical and Computer Engineering
to
The School of Graduate Studies
of
The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA

2018

In presenting this thesis in partial fulfillment of the requirements for a master's degree from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department or the Dean of the School of Graduate Studies. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Reetu Hooda (Date)

THESIS APPROVAL FORM

Submitted by Reetu Hooda in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and accepted on behalf of the Faculty of the School of Graduate Studies by the thesis committee.

We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate on the work described in this thesis. We further certify that we have reviewed the thesis manuscript and approve it in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering.

Committee Chair

Dr. W. David Pan (Date)

Dr. B. Earl Wells (Date)

Dr. Seong-Moo Yoo (Date)

Department Chair

Dr. Ravi Gorur (Date)

College Dean

Dr. Shankar Mahalingam (Date)

Graduate Dean

Dr. David Berkowitz (Date)

ABSTRACT

School of Graduate Studies
The University of Alabama in Huntsville

Degree: Master of Science in Engineering College/Dept: Engineering/Electrical and Computer Engineering

Name of Candidate Reetu Hooda

Title Search and Optimization Algorithms for Binary Image Compression

The creation of and demand for large-scale image data grows increasingly fast, resulting in a need for efficient image compression. This work focuses on improving the efficiency of lossless compression of binary images. To this end, we propose the use of the following two optimization algorithms, which search for either the best combination of scan directions for uniform blocks, or search for the best partitions of the input image into non-uniform blocks, with the goal of allowing for more efficient compression of the resulting sequences of intervals between successive symbols of the same kind. The first algorithm is the Binary Particle Swarm Optimization (BPSO) algorithm, which is shown to offer increasingly better image compression with additional iterations. The other algorithm is the Tree-based Search algorithm, which searches for the best grid structure for adaptively partitioning the image into blocks of varying sizes. Extensive simulations of these two search algorithms on various datasets demonstrated that we can achieve significantly higher compression on average than various standard binary image compression methods such as the JPEG 2000 and JBIG2 standards.

Abstract Approval: Committee Chair _____
Dr. W. David Pan

Department Chair _____
Dr. Ravi Gorur

Graduate Dean _____
Dr. David Berkowitz

ACKNOWLEDGEMENTS

First and foremost, I would like to express my heartfelt thanks to Dr. Pan for his patience, motivation and immense knowledge. His guidance helped me throughout my research and writing of this thesis. He has always been open to new ideas and steered me in the right direction whenever there was a need. I could not have imagined having a better advisor and mentor in my course of study in UAH.

Besides my advisor, I would like to thank my committee members Dr. Wells and Dr. Yoo for their time and insightful comments. I thank Dr. Ganesh Rao for his sincere and valuable guidance extended to me throughout my bachelor's degree.

I would like to express my very profound gratitude to my parents, my brother Erash, my close friends Satyaki Goswami and Aamira Indikar for providing me with unwavering support and love. This accomplishment would not have been possible without their belief in me. I am also grateful to Aditi Singh for reviewing my thesis, guiding me at every step and keeping me motivated.

Finally, I also place on record, my sense of gratitude to one and all, who directly or indirectly, have lent their hand in this venture.

Table of contents

List of Figures

List of Table

Chapter 1

Introduction

We have been witnessing a massive growth of data related to multimedia. Astonishingly, more data have been created in the past two years than in the entire history of human race and they are growing faster than ever before. With the popularity of smart phones, we are creating data in the form of images, videos, and audio recordings, at an incredibly increasing rate. According to a study in 2015, Cisco predicts that video will account for 80 percent of all consumers Internet traffic in 2019, up from 64 percent in 2014 [1]. Also, we have many photo and video sharing websites. “Facebook” features album creation and tagging photos by date, location or people. “Flickr”, which provides 1 TB storage and editing tools, has more than 40 million monthly visitors and 2 billion photos uploaded; in fact, in a single day, a few million photos are uploaded [2]. “500 pixels”, which is an image focused design, allows 20 photos a week. “Photo bucket” have 2GB of storage to store thousands of photos which we can edit using tools like smart color brush. “Canon” which has uploaded programs for MAC and Windows to quickly share photos to Facebook, “Twitter”, and many others like “DropBox” for public access. Colossal amount of image data is surrounding us. From our smart phones, digital cameras to computers, this data are stored either in personal, public, or commercial collections. All this yet very small part of it is analyzed and/or organized.

1.1 Motivation

Living in an information-centric world, we have an ever-growing reliance on creating data and its consumption, resulting in generation, processing, transferring, and

storage of exponential amount of data altogether which might be retained in multiple copies for longer period. In addition to that, advent of affordable devices in past decade led to increasing availability of imaging and visualization equipments instigating a huge increment in the need for algorithm to process image data [3]. Its current applications are numerous, namely in the fields of medicine, industrial inspection, video communication, remote sensing, robot vision, etc.

1.1.1 Overview of the problem

With new and advanced high-resolution cameras, we capture images with all the intricate details, leading to a big increment in overall file size, which occupies a significant amount of space. Images contribute to huge part of data and information. Storing and retrieving large quantities of image data is an extremely challenging and increasingly topical problem for both industry and academia [4]. Digital image processing is concerned with the development of computer algorithms working on digitized images. In general, image processing problems are solved by chain of tasks. This research focuses on two algorithms we proposed to solve one of the biggest image processing problems, lossless compression of binary images.

1.1.2 Need of Binary Image Compression

To reduce storage and transmission-bandwidth needs, the requirement for image compression has increased drastically. This work is aimed at improving the efficiency of lossless compression of binary images. A binary image is a digital image that has only two possible values for each pixel - either “0” or “1” as shown in Figure 1.1.

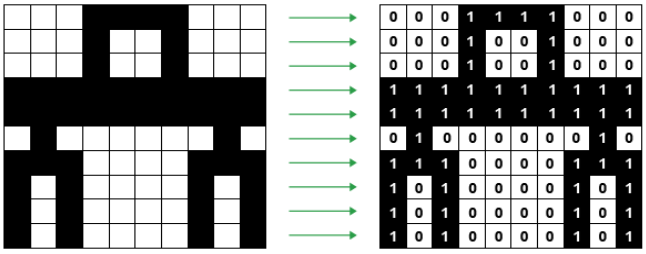


Figure 1.1: Binary images.

Binary image is a black and white image without color or without shades of gray. The image becomes a series of black dots on a white background. This may not seem very useful, however, there are several good uses of such images. Binary images are used in black and white laser printer, input/output devices, such as fax machines, and some bilevel computer displays which can handle only binary images. Such images are also processed by banks, government agencies, credit card companies, airlines and insurance companies.

1.2 Objective and Approaches

The image and video compression algorithm has become a critical business with the demand and development of multimedia product growth, resulting in insufficient storage in various devices and demand for higher transmission rates [7]. Typically, the images found on the web are compressed in some or the other format. The Compression techniques used in these images can be classified as:

Lossless Compression: This type of compression technique allows complete reconstruction of compressed data. Lossless algorithms are typically used for text where loss of any data is not affordable.

Lossy compression: Lossy algorithms can only reconstruct an approximation of the original data. They are typically used for images and sound where a bit of loss is often undetectable or at least acceptable [8].

Basic Binary Image compression: Binary image is also known as bi-level image which has only two possible values for each pixel. An entire binary image is a pixel array (matrix) with M rows and N columns comprising of zeroes and ones. A basic image compression algorithm stores the image into a bit-stream, in as compact way as possible, and displays the decoded image as exactly as possible [6].

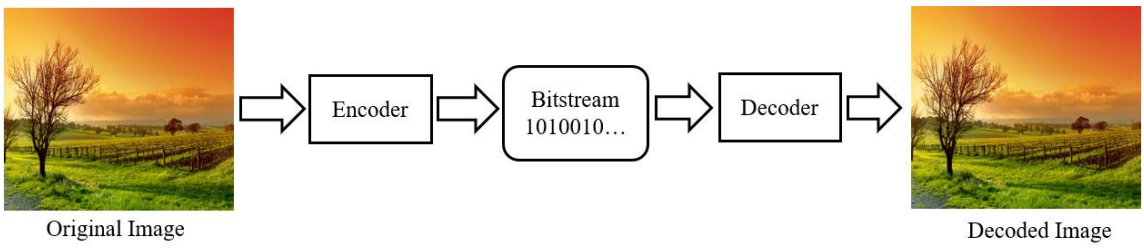


Figure 1.2: Basic image compression flow.

Figure 1.2 depicts a basic image compression flow. It is an encoding-decoding system where the encoder takes in the image and converts it into a coded form, having the series of binary data, only to produce the compressed form of image with smaller file size. Then the decoder decodes the compressed image for reconstruction.

This research is aimed at improving lossless compression of binary images. We propose Lossless Binary Image Compression using a block-based method, where the image is divided into blocks which are then scanned horizontally (i.e., from left to right and from top to bottom), or vertically (i.e., from top to bottom and from left to right). The best scan direction is determined by the optimization algorithm called BPSO (Binary Particle

Swarm Optimization). Since there is a need to decide for each block if it should be scanned in either horizontal or vertical direction for maximum compression, the search turns into a combinatorial problem. BPSO is one of the most robust population based stochastic optimization technique, which was extended from the PSO (Particle Swarm Optimization), which was developed by James Kennedy and Russell Eberhart in 1995. In the proposed framework, it finds the optimal combination of scan direction for all the blocks of a binary image in order to generate a sequence of intervals (the intervals being the distance between the current and previous occurrences of the same bit) and finally combining all the details in a data file which can be encoded. This data file is compressed by a data compression utility. The proposed method is shown to increase the compression ratio with every iteration, depending on the choice of various parameters such as the block size, compression utility used, and the number of particles.

We also proposed a tree-based algorithm, which uses dynamic structure to make the division of image into shape adaptive blocks. An entire binary image is initially divided into four equal sized blocks and further division of the first block, again into four equal sized blocks, is performed to examine if it improves the compression. This process is continued in order to decide the layout for the division of other sub-blocks. Two levels are explored to get a grid like structure with multiple block sizes. In this approach, further improved compression was achieved.

1.3 Organization of the Thesis

The outline of the thesis is as follows:

Chapter 2 gives an overview of the literature on application of image processing scheme

to compress the binary images. To achieve maximum compression ratio the lossless data compressor was applied on a sequence of intervals which represents distance between identical source symbols-either zeroes or ones in binary images. We also introduce the theoretical background of the optimization algorithm, Particle Swarm Optimization (PSO). In addition, we describe the PSO's extension to solve problems with discrete solution using Binary Particle Swarm Optimization (BPSO) algorithm.

Chapter 3 gives a detailed description about our proposed method of using the Binary Particle Swarm Optimization to compress binary images. We also discuss performance parameters and their effect on the simulation results.

Chapter 4 introduces a proposed tree-based search and optimization algorithm and explain how the algorithm operates in order to compress binary images. We show that the algorithm allows for dynamic structures (grid structures) and support multiple block size.

Chapter 5 compares the performance of optimization algorithm BPSO and Tree based search algorithm in maximizing the compression of binary image sequences. Also, we briefly discuss the types of compressor used. Comparison of the proposed framework with some standard compression schemes is also provided including the documentation of our findings.

Chapter 6 summarizes the results of our work and discuss further extension to this work.

Chapter 2

Literature Review

In recent years, the storage capacity and transfer bandwidth has grown accordingly but many applications still require compression. A digital image compression algorithm exploits the redundancy in an image to reduce the amount of data needed to represent an image with an acceptable subjective quality. The correlation amongst the neighboring pixel leads to redundancy of stored information, which is a common characteristic shared by most of the images. Redundancy reduction in image processing is focused on eliminating the duplication in the data (images) [13].

In general, three basic redundancies exist in digital images that follow:

Inter-pixel Redundancy: It is a redundancy corresponding to statistical dependencies among pixels, i.e., correlation between neighboring pixels. It is also known as Spatial Redundancy.

Temporal Redundancy: The prediction of value of a pixel can be done using the values of its neighboring pixels. Temporal redundancy focuses on identifying the correlation between adjacent frames in an image sequence. It is applied in video applications [14].

Coding Redundancy: An image is coded with a certain specified length for each pixel. Variable length code schemes such as, Huffman Coding, Run-length Coding, Arithmetic Coding, etc., can be used for compression. For example, a gray level image needs less information for each pixel to code. The average code bits assigned to the gray level values is given by:

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k)p(r_k) \tag{2.1}$$

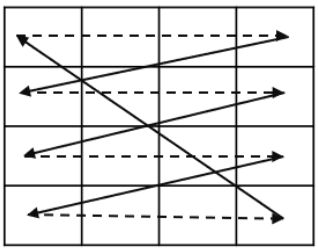
where r_k is a discrete random variable in the interval [0,1] that represents gray levels. Each r_k occurs with probability p_k. $l(r_k)$ is the number of bits used to represent each value of r_k.

The basic data redundancies can be dealt by different methods, as discussed in the following sections.

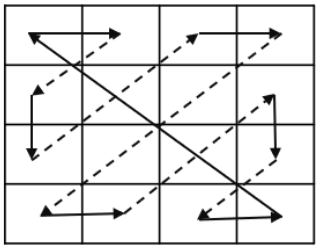
2.1 Coding Schemes for Lossless Compression

Lossless compression is employed for applications with stringent requirements such as medical imaging and text compression. Some decomposition techniques are often used in lossless compression to eliminate redundancy, followed by some entropy coding scheme [18]. Scanning patterns are used to traverse the pixels prior to coding. As the values of the pixels are non-homogenous in real images, non-linear scanning patterns are used [16].

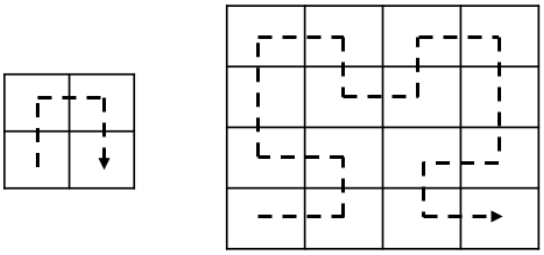
Following scanning patterns are used in image compression:



(a) Raster scan



(b) Zig-Zag scan



(c) Hilbert Scan

Figure 2.1: Scanning patterns.

Raster Scan: A Raster Scan is the rectangular pattern, whereby images are scanned from left to right and top to bottom according to the natural order of the object images. The line-by-line scanning creates a 'raster', which is a systematic process of covering area progressively one line at a time as shown in Figure 2.1(a). In raster scan, the image is subdivided into horizontal lines, called "Scan Lines" and each scan line is transmitted as a signal [16].

Hilbert Scan: As depicted in figure 2.1(c), the Hilbert Scan is a well-known space filling curve that visits every point of the grid exactly once. The quadrants on the square are mapped to segments on the line in a Hilbert scan [15]. The analytical solution of a space filling curve is as follows:

Let R^n be an n-dimensional space. A *"Peano"* curve is a locus of points $(y_1, y_2, \ldots\ldots y_n) \epsilon R^n$ defined by continuous function, $y_1 = \chi_1(v), y_2 = \chi_2(v), \ldots\ldots, y_n = \chi_n(v)\,, (v \epsilon R^1)$ where $0 \leq y_1, y_2, \ldots\ldots, y_n < 1 \; and \; 0 \leq v < 1$.

Hilbert curve has a space filling characteristic in R^2 and accommodate position through which the curve has passed [19].

Random scan: The order of the scan is chosen uniformly at random from all possible orders to scan. It is also known as Vector Scan and used for smooth line drawing applications. Each random combination of pixel positions in an image represents a possible random scan [15].

Zig-Zag scan: The zig-zag scanning is depicted in the figure 2.1(b) where it starts from topmost left corner and traverses in zig-zig fashion allowing to find the zero coefficients much faster and equivalently increasing coding efficiency [14].

Image pixels are traversed following a certain scanning patters to encode the image using coding schemes discussed in the following section.

2.2 Image Compression Methods

Existing coding schemes for Lossless Compression include:

1. Run length Coding
2. Huffman Coding
3. Arithmetic Coding
4. LZW Coding
5. Distance Coding

Run-Length Coding: It is used for sequential data to exploit the spatial/temporal redundancy in an image. It works reasonably well for images which involves similar patterns like repetitive intensities [17]. Runs of identical intensities are represented by *Run-length pairs* $\{V_i, R_i\}$, where each run length pair corresponds to the start of new intensity $\{V_i\}$ and count of consecutive pixels with same intensity $\{R_i\}$. Although few or no runs of identical pixels result in data expansion. For instance, images with extensive variations in different region may lead to large files after coding using this scheme [18]. The approximate run-length entropy of images is given as:

$$H_{RL} = \frac{H_0 + H_1}{L_0 + L_1} \tag{2.2}$$

Where H_0 and H_1 corresponds to average entropy of black and white run lengths. L_0 and L_1 corresponds to the average values of black and white run-lengths respectively.

In recent work [10], a novel based run length (BRL) coding scheme was introduced in which the most probable symbol block is coded using variable length code such as Huffman code to generate the codewords. The generated codewords have lower entropies compared to the conventional run-length coding. Biased run-length coding was analyzed on statistical models rather than relying on empirical data [12]. Run length coding performs better on binary images while employing variable length coding makes additional compression possible.

Huffman Encoding: It is a statistical method based on the occurrence of symbols (the pixels in images are interpreted as symbols). Shorter codewords are assigned to more frequent symbols and longer codewords are assigned to less frequent symbols. Huffman code is known to be optimal for some symbol distributions [18]. Huffman code, being a prefix code, implies that the binary code of any symbol is not the prefix code of any other symbol as it should be uniquely decodable.

Some image coding standards use run-length coding on the original image and then apply Huffman coding. Typically, it gives higher compression but for some cases, sequential application of these two-coding schemes can get compression ratio lower than the alternate method of applying one of them alone or applying Huffman coding alone [17]. It works well for text and fax transmission. However, text compression symbol frequencies vary with context. In the case of ensemble change, frequency and probability change as well, resulting in variation of optimal code. Another drawback of Huffman coding scheme is that it requires two passes, one to build a statistical model of the data and a second to encode it. So, it is a relatively slow process. Thus, lossless encoding technique that use Huffman coding are notably slower than other techniques when reading or writing files [21].

Arithmetic Coding: Entire sequence of source symbols is assigned a single arithmetic code. It takes a message composed of symbols and converts it to a floating-point number greater than or equal to zero and less than one. It generates non-block codes and if the length of the sequence increases, the interval used to represent it becomes smaller and the number of information units required to represent the interval [0,1) becomes larger [22].

LZW Coding: The idea of using address information to designate a piece of message can be used in image compression. Lempel-Ziv-Welch is a dictionary-based algorithm. It follows the design of a dictionary where a word can also be referred to as page and order number. In static coding,

dictionary is fixed during encoding and decoding processes, unlike dynamic dictionary encoding. It is used as a compress command on UNIX [17].

Distance Coding: As proposed in [3,15], in distance coding scheme, the image is adaptively chosen to be traversed either in horizontal or vertical direction within a block. The image is first subdivided into blocks of different size and choice of direction to be scanned is based on entropy. Theoretically, lower entropy corresponds to higher compression and vice-versa. The entropy is calculated using the formula $H(R) = \sum_{k=0}^{L} P(r_k) \log P(r_k)$, where L is the number of distinct symbols r_k (with $1 \leq k \leq L$) in the interval generated. $P(r_k)$ is the probability of each symbol r_k occurring in the sequence. Finally, the direction bits and intervals with the header is combined in a data file and compressed using a compressor.

2.3 Optimization Algorithms

Optimization algorithms focuses on minimizing mathematical functions, that are used in every field of engineering. Particle swarm Optimization (PSO) is one of the most popular nature-inspired metaheuristics optimization algorithm developed by James Kennedy and Russell Eberhart in 1995 [22]. It has been successfully applied to find optimal solution by simulating social behavior of the movement of an organism in a flock of bird or school of fishes. It is an extremely simple swarm intelligence algorithm that is effective for solving problems in many fields, like machine learning, data mining, operational research, image processing, etc. Since its development, it has been gaining worldwide popularity as a powerful global optimization tool [25].

The PSO algorithm consists of swarm and each member of the swarm is called “particles”. Each particle represents a potential solution of the problem (objective function). The particles fly around the search space with certain velocity till an optimum solution is found. It is guided by

personal experience (P_{best}), overall experience (G_{best}) and the present location of the particles in the search space [24]. Each particle keeps track of the best position it has reached so far (P_{best}). The best solution achieved by the whole swarm after every iteration is G_{best}. P_{best} and G_{best} are updated every time a better solution is found. The working of PSO can be explained using a simple model shown in figure 2.2.

Let us consider a search space X of size n and dimension D. Position of each particle is denoted by $\vec{x_i}(t) \in X$ initialized with velocity $\vec{v_i}(t)$. The velocity of the particle $\vec{v_i}(t)$, describes the movement of the particle $\vec{x_i}(t)$. $\vec{P_i}(t)$ is personal best experience of particle i and $g(t)$ is common best experience which is not particle specific.

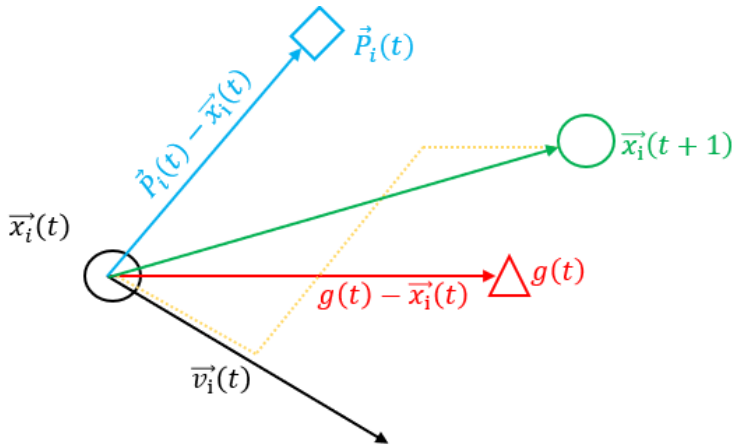


Figure 2.2: PSO search mechanism.

Let us define two vectors, $\vec{P_i}(t) - \vec{x_i}(t)$ *and* $g(t) - \vec{x_i}(t)$. Particles move to the new position $\vec{x_i}(t+1)$ with new velocity $\vec{v_i}(t+1)$ which are defined as,

$$\vec{v_i}(t+1) = \omega\vec{v_i}(t) + c_1 r_1 \left(\vec{P_i}(t) - \vec{x_i}(t)\right) + c_2 r_2 \left(g(t) - \vec{x_i}(t)\right) \quad (2.3)$$

$$\vec{x_i}(t+1) = \vec{x_i}(t) + \vec{v_i}(t+1) \quad (2.4)$$

where $c_1\ and\ c_2$ are positive acceleration constants; $r_1\ and\ r_2$ are the random numbers uniformly distributed over the interval $[0,1]$, and ω is called inertia weight [24].

The PSO model can also be used for motion estimation with some modification to improve the speed of convergence and the accuracy of the optimal solution. Models with very low computational cost and a good balance between local and global searches are suitable for the context of video coding using this algorithm because of the time constraint in motion estimation. Larger values of $c_1\ and\ c_2$ causes the particles to move rapidly along their trajectories. Thus, there is a possibility that the particle might bounce off the optimal solution. On the other hand, smaller values result in slow movement. Consequently, the particle takes considerable amount of time to reach the optimal solution. In [25], time varying coefficients approach (strategy promoting adaptively choosing acceleration coefficient to extend exploration) for block-based motion estimation was proposed in which particles are attracted towards the global best positions. In the proposed method, the velocity and position are updated using the following equations:

$$\overrightarrow{v_{\mathrm{i}}}(t+1) \;=\; \omega\overrightarrow{v_{\mathrm{i}}}(t) + c_1(t)r_1\left(\vec{P}_i(t) - \overrightarrow{x_{\mathrm{i}}}(t)\right) + c_2(t)r_2\big(g(t) - \overrightarrow{x_{\mathrm{i}}}(t)\big) \tag{2.5}$$

$$\overrightarrow{x_{\mathrm{i}}}(t+1) \;=\; \overrightarrow{x_{\mathrm{i}}}(t) + \overrightarrow{v_{\mathrm{i}}}(t+1) \tag{2.6}$$

Where,

$$c_1(t) \;=\; (c_{1,min} - c_{1,max})\frac{t}{Iter_{max}} + c_{1,max}, \tag{2.7}$$

$$c_2(t) \;=\; (c_{2,max} - c_{2,min})\frac{t}{Iter_{max}} + c_{2,min} \tag{2.8}$$

where ω is set to be 0.9, $c_{1,min}$, $c_{1,max}$, $c_{2,min}$ and $c_{2,max}$ can be set to 0.5, 2.5, 0.5, 2.5, respectively. $Iter_{max}$ is the number of maximum iterations [24].

PSO is used for optimization of continuous non-linear functions. It was reworked to operate in various search spaces based on the problem. Some of the improvements and modifications are as follows:

Binary Particle Swarm Optimization (BPSO): A considerable amount of optimization problems are set in a space where distinctions, both discrete and qualitative, between variables as well as levels between variables are present. BPSO was developed to operate in discrete binary search space. In discrete version, the particle swarm formula remains unchanged, but the position update is defined by the following rule:

$$if\big(rand(\,) < S(v_{id})\big) \tag{2.9}$$

$$then\ x_{id} = 1$$

$$else\ x_{id} = 0$$

where the function $S(v_{id})$ is a sigmoid limiting transformation defined as

$$S(v_{id}) = \frac{1}{1+e^{-v_{id}}} \tag{2.10}$$

and $rand(\,)$ is a quasirandom number selected from a uniform distribution in [0,1].

$$\text{Now, } \lim_{v_{id}\to 0} S(v_{id}) = 1 \text{ and } \lim_{v_{id}\to \infty} S(v_{id}) = 0 \tag{2.11}$$

Hence, $S(v_{id}) \in [0,1]$.

Consequently, v_{id} is limited to V_{max}.

When it comes to binary space, a particle tends to move to nearer and farther corners of the hypercube. It does that by flipping various number of bits. With all the bits reversed, a particle

moves the “farthest” while it does not move if there are zero bits flipped. Hence, the number of bits changed with each iteration gives value of the velocity of the particle.

MBPSO: The modified BPSO is an improved algorithm to extend the exploration of the original BPSO. In MBPSO, a new probability function is introduced which maintains the diversity of the swarm, making it more effective and efficient in solving optimization problems. After updating the velocity, the original BPSO does not include the particle’s previous position. However, in MBPSO the position-update equation is an explicit function of previous position and previous velocity like in continuous optimization in the case of PSO [25]. The movement of particle is sensitive to V_{max}. Thus, fine tuning V_{max} controls the particle behavior.

Besides these PSO models, many variations to the models have been introduced to represent in states [20,22,23]. Instead of involving a high dimensional bit vector as in BPSO, the proposed methods involve states of variables involved. Representation of basic PSO models in states performs consistently in solving the discrete combinatorial optimization problem.

Chapter 3

Block Scan Pattern Search using Binary Particle Swarm Optimization

Logic and mathematics are nothing but linguistic structure.

Digital images are typically large, consequently occupying larger storage space. The purpose of image compression is to eliminate the redundancy of the image data using coding schemes. This chapter discusses how to use Binary Particle Swarm Optimization (BPSO) to compress binary images. The technique proposed minimizes the size of a binary image without modifying the quality of the image [31]. Before we describe the algorithm implementation, we present a brief introduction of the basic BPSO model, the scanning pattern employed in the algorithm and the overview of the algorithm.

3.1 Basic BPSO Model

The main goal of optimization algorithms is to find optimum value of a given mathematical function. Among other things, optimization algorithms are used to evaluate design tradeoffs, to assess control systems, and to find patterns in data. Binary Particle Swarm Optimization is based on bird flocking or fish schooling behavior. In a BPSO system, each particle represents the candidate solution to the optimization problem and swarm of particles is considered to fly through the search space. The position or movement of the particles is guided by the experience of the neighboring particles and experience of the entire swarm. A particle's best position is termed as $''P_{best}''$ and the global best of the entire swarm is termed as $''G_{best}''$. Fitness functions are used to measure the performance of each particle which is problem specific [30]. In the proposed

method, BPSO is used to search for the best scanning patterns for image blocks, such that highest compression can be achieved for binary images. The scanning patterns used are discussed in the next section.

3.2 Scanning Patterns

There are many possible scanning patterns in which one can traverse the symbols in an image. The following scanning patterns are used in the proposed algorithm based on the best scanning directions discovered by the BPSO algorithm.

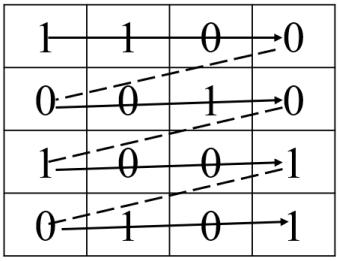
Horizontal Raster Scan: The image blocks are scanned following the horizontal direction, i.e., from left to right and from top to bottom.

Vertical Raster Scan: In vertical scanning, the order of scan is from top to bottom and left to right, following a vertical direction.

If the image block is scanned using Horizontal Raster Scan, then the direction bit for the block is set to "1" otherwise if the block is scanned in vertical direction, then the direction bit is set to "0".

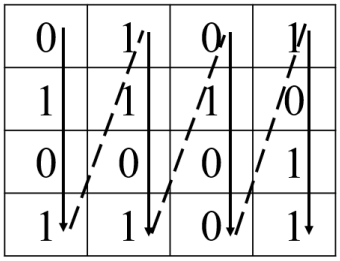
Interval Generations: In conventional distance coding method, the pixels are scanned following a fixed raster scan and calculates the distance between the previous and next occurrence of the same bit referred to as *"Intervals"*. The interval sequence generated affects the degree of compression. Due to the complementary nature of the bits in binary images, distance coding focuses on the symbol "1" to generate the intervals. Different intervals are generated by following different scans. The problem of coding the original image changes to the problem of coding the intervals after generating them.

Figure 3.1 illustrates Horizontal and Vertical Raster scan to generate different interval sequences.



Horizontal Scan

Interval sequence : [1 1 5 2 3 2 2].



Vertical Scan

Interval sequence : [2 2 1 1 2 2 3 2 1].

Figure 3.1: Scanning patterns for the search algorithm.

3.3 Overview

In the proposed method, the image is divided into uniform sized blocks. Each block is scanned either following a Horizontal Raster Scan or Vertical Raster Scan as determined by a search algorithm, to generate the interval sequences. Let us consider an illustrative example, where an image of size 8 × 8 divided into 4 × 4 blocks, to give a total of 4 sub-blocks as depicted in figure 3.2.

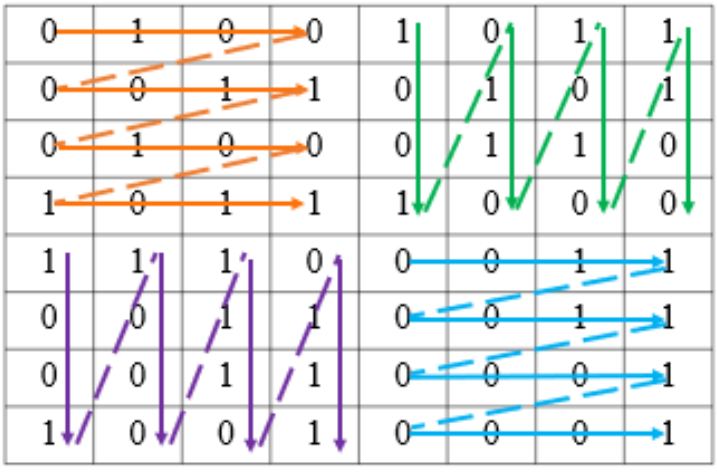


Figure 3.2: An 8×8 image divided into 4×4 blocks.

The scan path for all the blocks are referred to as '*Direction Bits'* as summarized in Figure 3.3(b). Now, let us assume the direction bits $\begin{bmatrix} H & V \\ V & H \end{bmatrix}$ as the optimum one for this image. The direction bit is set to "1", if horizontal direction is chosen for a block; otherwise, a direction bit of "0" will be set. As a result, direction bit of "1" is assigned for the first block. The interval sequence is generated while scanning the first block following a Horizontal Scan Path. Similarly, block 2 and 3 are scanned in vertical direction and block 4 is scanned in horizontal direction to generate different sequences as shown in Figure 3.3(a). Finally, a bitmap of the image is generated to view the scan path of the whole image as depicted in Figure 3.3(c).

Block 1: [2 5 1 2 3 2 1]	Block 2: [1 3 2 1 2 2 2 1]
Block 3: [1 3 1 4 1 1 3 1 1]	Block 4: [3 1 3 1 4 4]

(a) Interval sequences

1 Block 1	0 Block 2
0 Block 3	1 Block 4

(b) Direction Bits

(c) Bitmap

Figure 3.3: Interval generation and direction bits.

Search Space: In the illustrated example, there are total of 4 blocks and each block can be scanned in either horizontal or vertical direction. Thus, there are $2^4 = 16$ different combinations in which image can be scanned. It's a 4-dimensional search space optimization problem. However, the dimensionality increases with larger images.

3.4 Algorithm

In general, an image of size $M \times N$, when divided into a uniform block size of $p \times q$, gives r blocks given below:

$$No.of\ Blocks\ = \frac{Image\ Size}{Block\ Size}$$

$$r\ = \frac{M \times N}{p \times q} \tag{3.1}$$

The value of r gives the number of dimensions of the search space, and thus the number of parameters involved in the optimization problem. With, larger images the search complexity increases as 2^r. For instance, an image of size 256×256, divided into 8×8 block, gives a total of 1024 blocks and complexity of 2^{1024}. A full search method to find the optimum combination out of 2^{1024} combinations for maximum compression is an exhaustive process and time consuming. BPSO algorithm can be used to find the best combination of the scanning directions to get maximum compression on the intervals generated according to the combination.

3.4.1 Parameters Initialization

Data compression ratio is defined as the ratio of the original file size and the compressed file size. The main goal of image compression is to maximize the degree of compression by minimizing the file size. In the proposed method, the task is to find the best combination of the scan patterns to minimize the compressed file size. The values of the vector representing the scanned path are discrete, since a block can be scanned in only two directions i.e., $H\ or\ V$.

Parameters initialization is the very first step in any PSO model. The initial position and velocities of the particles are chosen randomly. The particles in the BPSO model are initialized with a vector, which represents a specific scan path. The dimension of the vector depends on the total number of blocks an image is partitioned into. The data structure of a particle may contain several sections each recording its current position, best position so far and velocity respectively as coordinates of current position, coordinates of best position so far, and velocity vectors in a D-dimensional space. If the initial positions are already close to global optima, then the iterative BPSO process might converge comparatively quickly [18]. A random combination of scanning patterns for the blocks represents a candidate solution.

After initializing the particles with random initial positions and velocity, evaluation of all the candidate solutions is performed, which becomes the candidate solution fitness, known as P_{best} and the best among all the candidate solution is the global best G_{best}. Hence, the first swarm is generated.

3.4.2 BPSO Model

BPSO is initialized with a swarm size of *N* particles. Each particle is treated as a point in the *D-dimensional space*, where *D* is specified based on the number of blocks. The i^{th} particle is represented as $x_i = (x_{i1}, x_{i2}, \ldots\ldots, x_{iD})$, where x_{ij} limited between two levels i.e.,[0,1]. The velocity of particle i is represented as $v_i = (v_{i1}, v_{i2}, \ldots\ldots, v_{iD})$. The previous best position of the i^{th} particle is represented as $p_i = (p_{i1}, p_{i2}, \ldots\ldots, p_{iD})$ and the best among the entire swarm is represented by $p_g = (p_{g1}, p_{g2}, \ldots\ldots, p_{gD})$, where $i = 1,2, \ldots\ldots, N$.

Now, at each step the idea of BPSO is to change the velocity of each particle towards its P_{best} and G_{best} location. The velocity of a particle can be updated according to the following equation:

$$v_{id} = \omega v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \quad (3.2)$$

And the position of the particle can be updated using a sigmoid function:

$$if(rand() < S(v_{id})) \quad (3.3)$$

$$then\ x_{id} = 1$$

$$else\ x_{id} = 0$$

$$\text{where, } S(v_{id}) = \frac{1}{1+e^{-v_{id}}} \quad (3.4)$$

The velocity of a particle is influenced by three factors:

- ωv_{id} is a momentum term to avoid oscillations in the search space. ω represents inertial weight attached to the particle's previously attained position. The momentum term incorporates the effect of previous velocity on current velocity of the particle [37]. ω controls the effect which the last iteration speed has on the current speed. Larger value of ω improves global search capability and smaller value of ω improves the partial search capability of the BPSO algorithm [41].
- $c_1 r_1 (p_{id} - x_{id})$ is cognitive component which represents the distance that a particle is from the best solution found by itself.
- $c_2 r_2 (p_{gd} - x_{id})$ is referred as social component which represents the distance that a particle is from the best position found by its neighborhood [38].

c_1 and c_2 are called acceleration constants, r_1 and r_2 represent random numbers in the range of [0,1]. The flowchart of BPSO model is shown in Figure 3.4.

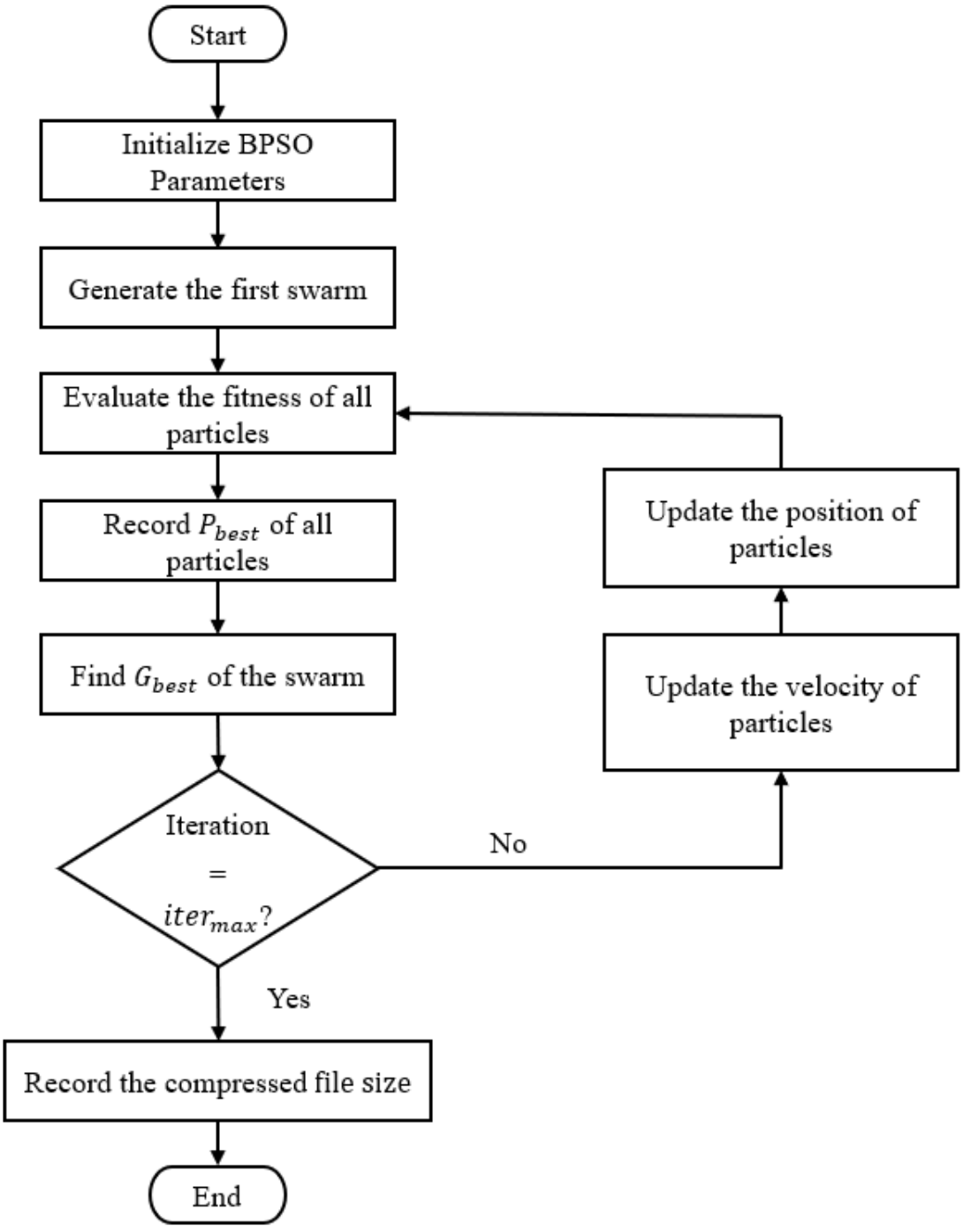


Figure 3.4: Flowchart of the BPSO algorithm.

3.4.3 Particle History Preservation

Due to the involvement of random numbers in evaluation of velocity in equation (3.2), there are high chances that same positions can be encountered, which were visited in previous iterations. Calculation of fitness value of the same position which was already visited increases computational complexity and time consumption. Thus, avoiding re-evaluation of the fitness value for the same position would reduce overall computational complexity quite significantly. The proposed method records the history of all the previously visited positions in a structure having two sections for visited positions and their corresponding cost value.

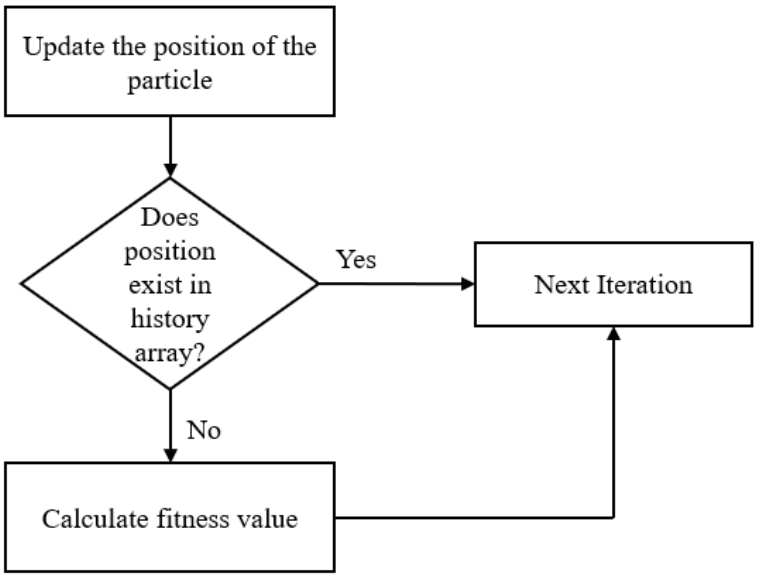


Figure 3.5: Flowchart of history preservation scheme.

Before evaluating the fitness of the new position, the history array is checked in next iterations. If the position exists in the history array, the position will be skipped and if it is not visited, the fitness value will be calculated and both position and its cost value will be appended to the history array. Therefore, particle history preservation scheme would

effectively reduce the overall computational complexity by avoiding redundant fitness calculations associated with candidate position that have been visited previously [18]. The flowchart in Figure 3.5 explains the particle history preservation scheme.

3.4.4 Fitness Function

The *objective function,* also referred to as *"fitness function"* maps the search space to the function space. The function space has only one output. Hence, it is one-dimensional. The function space is then mapped to the one-dimensional fitness space, providing a single fitness value for each set of parameters. The single fitness value determines the optimality of the set of parameters for the desired task [35].

For a known function f, which is differentiable, mathematical methods like calculus can easily provide us with minima and maxima of f. However, in real-life optimization tasks, this objective function f, is often not directly known. Instead, the objective function is a "black box" to which we apply parameters (the candidate solution) and receive an output value [41].

In the proposed method, we apply different scanning paths and determine the compressed file size as its output value. Figure 3.6 depicts the calculation of the fitness function. The input image to the fitness function is a binary image. Binary images are 2-D matrices, which have only two possible values for each pixel. Binary nature of the symbols in the image offers an advantage, where the location of either 0's or 1's completely determines the original binary image. We perform block-based scan on the input image. The input image is divided into blocks and blocks are scanned based on the direction recommended by the BPSO algorithm in its current iteration.

If the direction bit is ‘1’, the block is scanned in horizontal direction and vice-versa, to generate sequence of intervals. Finally, the sequence of intervals and direction bits are combined into a data file and compressed by a data compression utility. A header is also attached to the data file to synchronize the direction bit sequence with the sequence of intervals. The output of the fitness function represents the compressed file size following a fixed scan path chosen by BPSO algorithm in its current iteration.

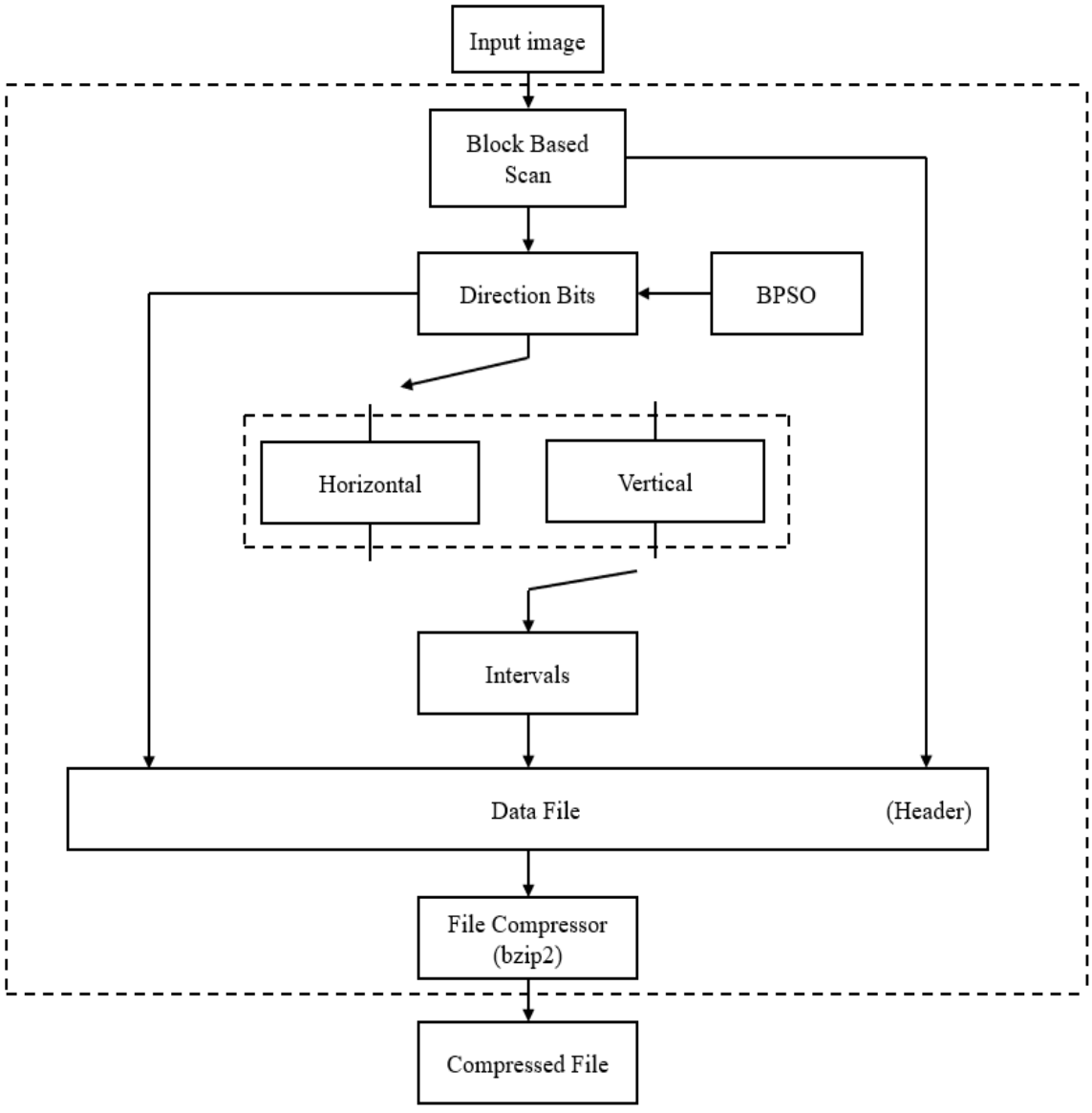


Figure 3.6: Flowchart of the fitness function calculation.

With every iteration, new direction bits are generated by the PSO to calculate the fitness function. The ultimate fitness value is calculated at the final step which is the compressed file size we can achieve finally after the BPSO algorithm is terminated.

3.4.5 Decoder

A compressed file is the input to the decoder, with reconstructed image as its output. After decompressing the file, extra information such as Block Size and Number of Blocks in the image are fetched. The direction bits obtained from the data file help to decode the intervals in order to convert back to location indices of the 1s in the corresponding block within the image.

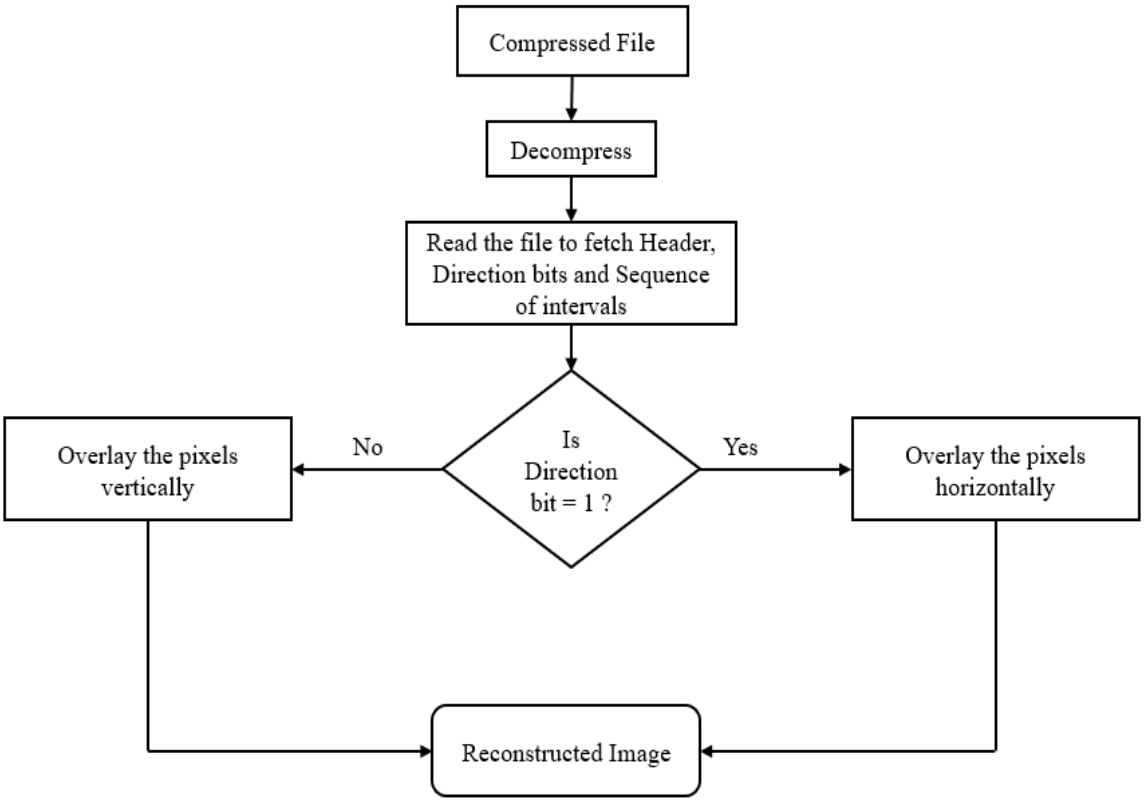


Figure 3.7: Flowchart of the decoder.

Figure 3.7 shows the flowchart of the decoder. If direction bit is “1”, the interval sequences are overlaid horizontally otherwise pixels are overlaid vertically to generate a reconstructed image losslessly.

Lossless Check: In this step, we compare the reconstructed image to the original binary image pixel by pixel. So, Lossless Check is performed at the end of the decoder to make sure that the reconstructed binary image matches the original binary image pixel by pixel.

Chapter 4

Tree Based Search Algorithm

The value of an idea lies in the using of it.

The BPSO based search method is limited by the fixed block size generating uniform grid structure. Several regions of an image are less compressible when compared to other regions. To adapt to the changing statistics of the image, we proposed to use variable sized blocks as determined by the tree-based search algorithm. It divides the image into variable sized blocks to generate non-uniform grid structure. This chapter explains the Tree Based Algorithm we proposed for compression of binary images, which uses full search of sub-block in an image and supports variable grid size structure. We also present two level recursive splitting of the image for implementation.

4.1 Introduction

Various studies in image and video coding show that decomposition of the original 2D signal (image) into a set of contiguous regions help in increasing the compression. This process is called "s*egmentation"*. Its application includes redundancy reduction and source coding [48]. For low bit-rate compression applications, segmentation-based coding methods provide, in general, higher compression when compared with traditional transform, vector quantization (VQ), and sub band (SB) coding approaches. The partitioning paradigm used in algorithms with segmentation keeps the computational cost and complexity reasonably low and still achieve superior compression efficiency [42].

A natural image consists of regions of smoothness as well as the ones with sudden variations. Thus, it can be segmented into regions having widely different perceptual importance. Typically, high frequency content is considered a reasonable measure, as extra bits are allocated to highly detailed area(s) in the image. Such regions are intrinsically less compressible than less detailed regions. By allocating uniform number of bits across an image, regions with less variations are coded with far more bits than needed. This provides motivation to examine image coder which uses variable size blocks according to the importance of the region being coded [50].

The proposed technique dwells on the idea of exploiting the smoothness in portion of an image by splitting it into variable length segments. Portions dominated by change are retained as smaller blocks and the smooth segments are not chosen to be divided further. In other words, smooth regions are grouped as a single partition. Thus, decomposition of only high concentration region is performed.

The next section illustrates the tree-based search algorithm chain using full search of sub-images and two level recursive splitting.

4.2 Tree Based Search Algorithm Chain

The Tree Based Search Algorithm Chain can be broken down into the following three main steps to generate the final compressed file:

- Full search of image sub-block
- Optimal tree structure in the set of all possible tree structures
- Two-level splitting of the original image

4.2.1 Full Search

Full search is a repeated step in the algorithm which is used to examine the change in compression after partitioning a high frequency content region in an image. In general, Full Search (FS) refers to techniques for searching a single element/document or collection of elements/documents in full database. Full search algorithm is the most simplistic search method with few tradeoffs. The advantage of full search lies in the fact that we can find the absolute optimal solution. It guarantees to explore all the search paths to solve a given problem. However, its high computational complexity makes it difficult for real time implementation. Since full search on small data has low time complexity, the tree-based search algorithm uses full search on image blocks instead of using on the whole image itself.

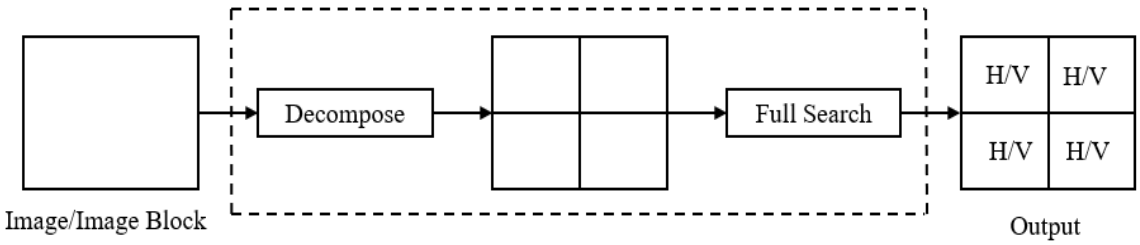


Figure 4.1: Flow chart of the full search.

Figure 4.1 shows the flow of full search operation used in the proposed method. Either the image or a section of it, is the input to the full search block, which divides it into 4 equally sized blocks. Horizontal and Vertical scanning schemes are used to traverse the pixels of the input image block. Each of the four blocks can be scanned in two direction which makes total of $2^4 = 16$ combinations to be explored. Division of the input image is followed by the full search, to examine all the possible scanning patterns and

consequently finding out the best combination for highest compression. In summary, full search is used in the following operations.

- Divide the image into 4 equal blocks
- Find the best combination of scanning patterns out of 16 combinations

The next section describes the tree structures explored and the basis for selection of a structure is stated.

4.2.2 Adaptive Grid Structure

The aim is to obtain a compact representation of the information content in each image and the content of an image can be described in terms of regions contained in the image. Thus, the amount of compression depends on concentration of pixels in a portion of the image [48]. The compression of 2D signal is obtained by partitioning the original signal into complex grid structure, which consequently results in division of pixel concentration. To divide the regions of image, we use a tree-based search method to find an overall optimal tree structure for decomposing the image. The method involves decomposition of the image into variable size blocks using a tree. The image is referred as "*image tree"* or *"original tree",* which is then segmented. The image will be decomposed into larger blocks for smooth regions and smaller ones for regions with largely varying content. Adaptive grid structure takes advantage of short and long-range correlations present in an image through image segmentation. It examines non-uniform areas and isolates them from the remaining parts of the image, which can be coded at a lower rate than it would be otherwise possible. As depicted in Figure 4.2, a variable block size scheme is used to partition the image into rather complex grid structure. There are several ways to perform

segmentation. In tree-based search algorithm, segmentation is performed with variable size grid structure by isolating the perceptually more important areas of the image into smaller blocks and separately identifying larger “uniform” blocks. In performing such segmentation, the tree-based search algorithm takes series of binary decisions based on the full search performed on the sub-blocks. A two-level tree structure has a total of 16 structures as shown in the figure below and all of them are explored to find out the best among the whole set.

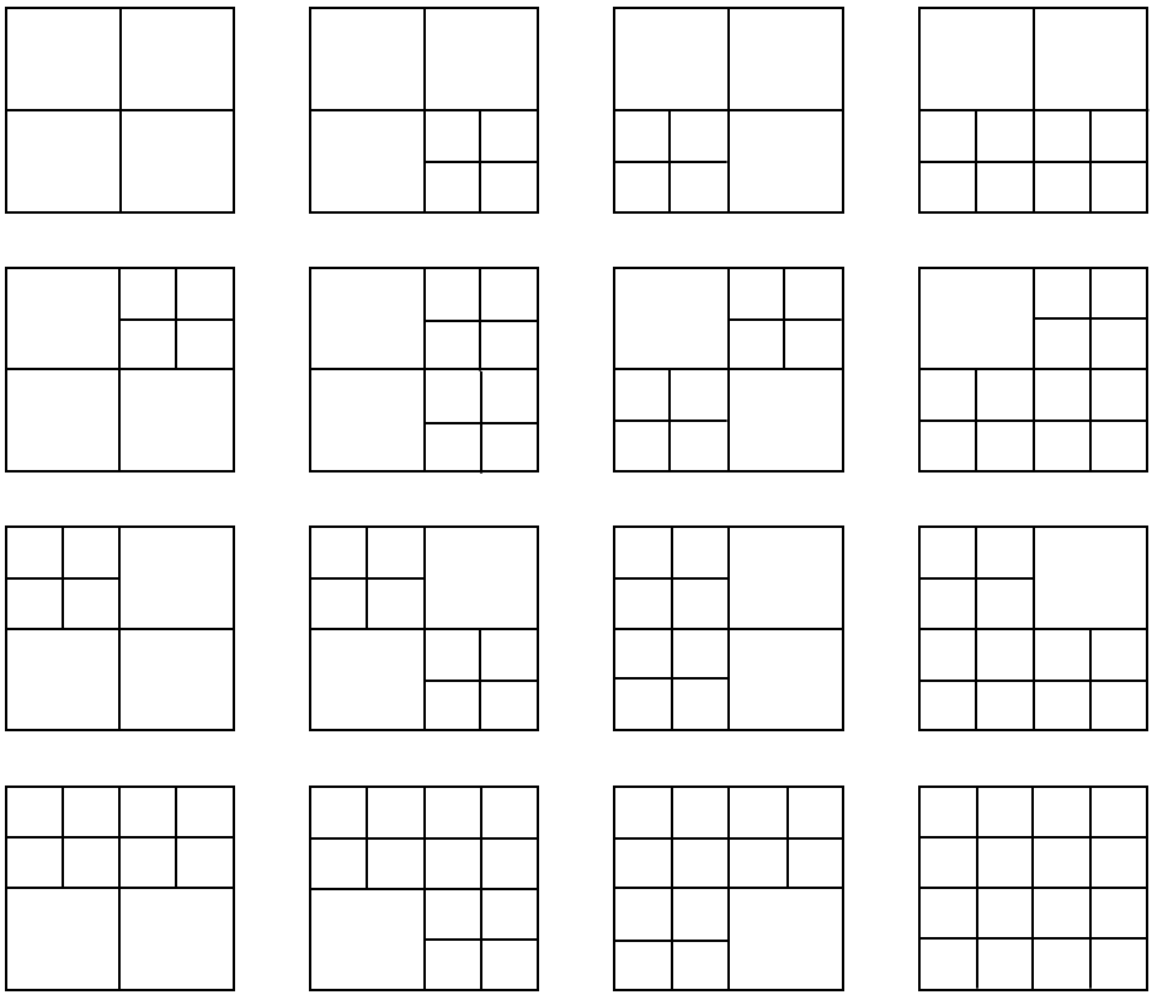


Figure 4.2: Adaptive grid structures.

4.2.3 Two Level Recursive Splitting

Tree based search approach is a very effective and computationally simple technique for image compression. The workflow of Tree Based Algorithm is summed up by the flowchart shown in Figure 4.3. It starts with decomposing the original image, also known as *"original tree"*, into two levels by partitioning the image recursively into four sub images. At level one, the original tree is divided into four blocks and at level two, the image blocks are further divided into four blocks. Therefore, as depicted in Figure 4.3, an image can be represented by a binary tree structure. The segmentation is performed iteratively and controlled at each step to ensure that the segmented image improves compression. Tree structures are constructed by repeated splits of the image.

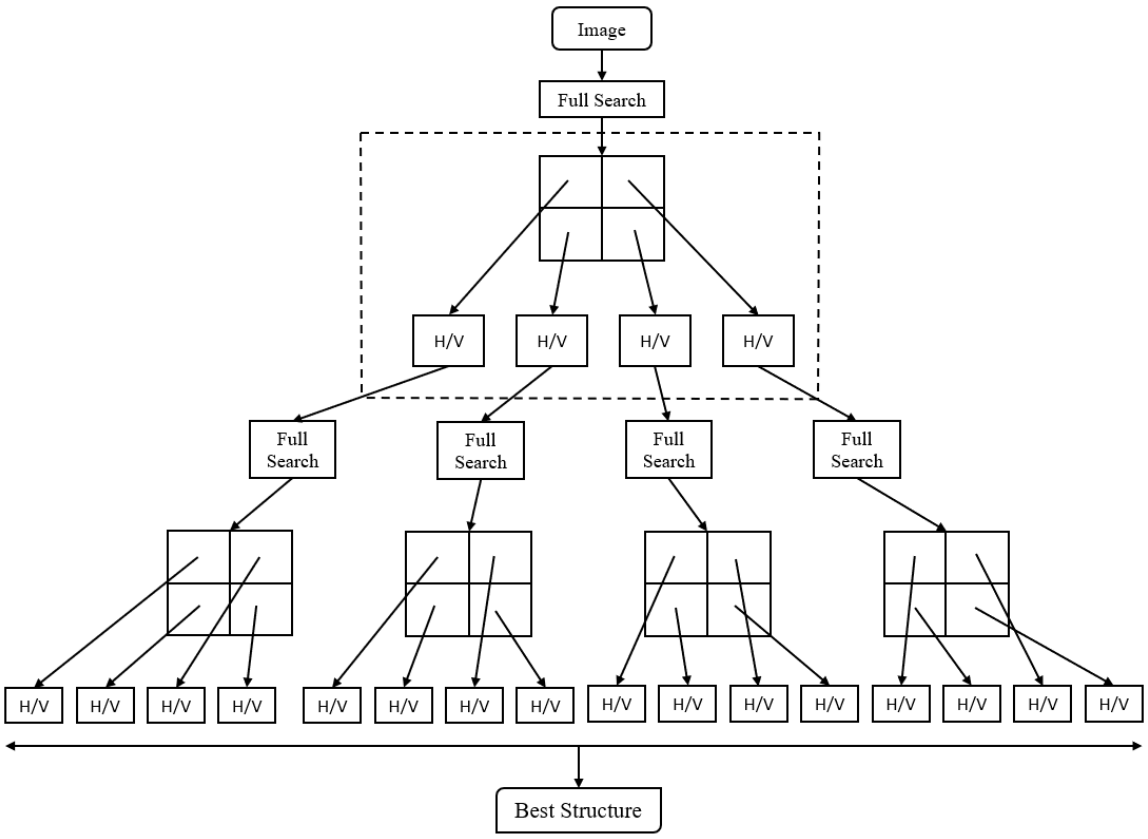


Figure 4.3: Tree based search algorithm.

The proposed algorithm uses a tree structure in which the original image is the root node. The original image is split into four equal-sized blocks, emanating four branches from it. These branches point to nodes that are the children of the parent node. The child nodes are obtained after splitting parent block. The blocks are called nodes, which can either be a leaf, i.e., it is not further subdivided, or can branch into four blocks, each being a child node [49]. The tree represents recursive splitting of the space (image), and a node at any stage represents one quarter sub block of the parent image block from the previous stage. Therefore, every node of interest corresponds to one region in the original space. Four child nodes will occupy four different regions. If put them together, we get the same region as that of the parent node.

The tree structure can be designated by a series of bits that indicate termination by a leaf with a "0" and branching into child nodes with a "1". In the end, every leaf node is assigned with a class ('H' or 'V') based on the full search test [45]. Full search test is performed to determine whether segmentation of a block improves compression or not. If the test is positive, then the node becomes a terminal node with division, otherwise it becomes a leaf node and only one bit is required to indicate the information about the division. We refer to this bit as "binary tree structure bit" or "decision bit" since it describes the shape of the final tree selected to be encoded [44].

Let us consider the structure shown in Figure 4.4(a) as the optimal tree structure. Figure 4.4(b) corresponds to the tree path followed and figure 4.4(c) represents the division decision bits. The tree is defined in such a way that each node has either no offspring or four offsprings [46].

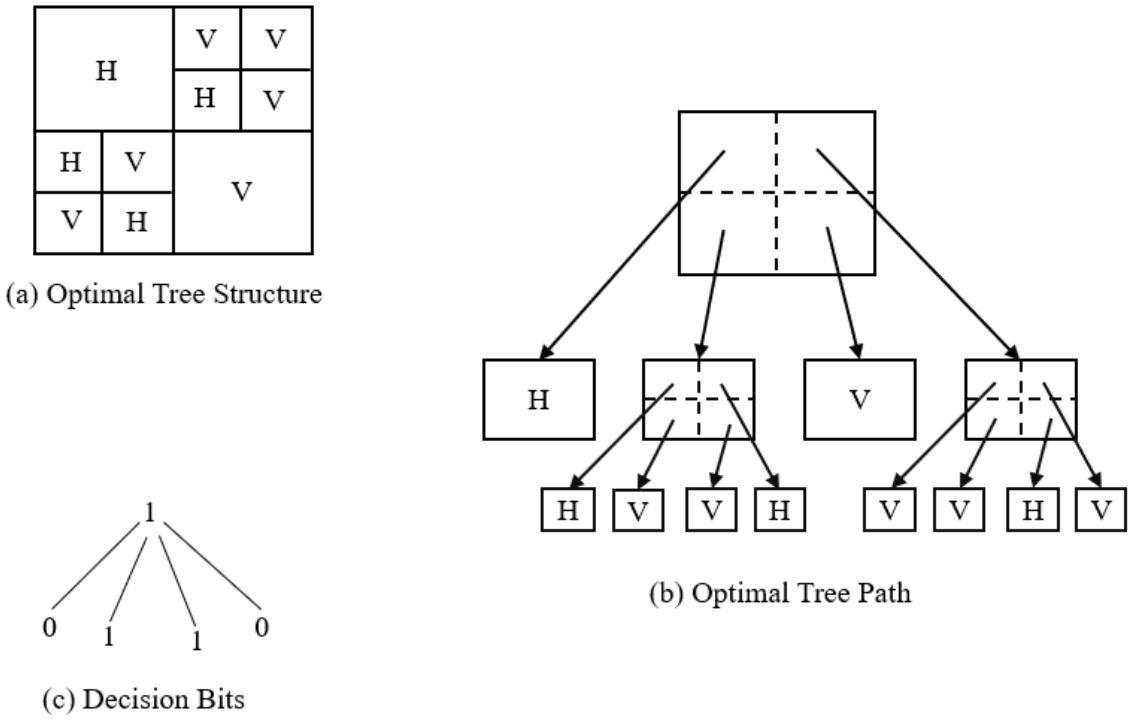


Figure 4.4: Final structure.

A judgement is first made as to whether the entire block can be represented by a single leaf or whether it must be divided into four sub blocks. If the block is divided, then a binary decision is made for selection of scanning direction. The process continues for rest of the blocks. To identify the optimal tree, the structures shown in Figure 4.2 are examined. Hence, the procedure terminates after two-level recursive splitting. The final non-uniform grid structure in tree-based algorithm is represented by side information, which indicates if the division of sub block is performed or not.

The tree structure and sequence of intervals are combined into a data file with the header and compressed by a data compression utility as the final step.

Lossless Check: lossless check is performed at the end of the decoder to match reconstructed image with the original image pixel by pixel.

Chapter 5
Simulation Results

This chapter discusses the simulation results of proposed algorithms and its comparison to six other lossless bi-level image compressors. MATLAB R2017a was used for code implementation and result analysis. MATLAB provides a comprehensive environment for code development, result visualization, image and video processing. To test the proposed methods, a set of binary images were generated by converting grayscale images into binary images using thresholding on frames of the video sequence [46]. Figure 5.1 shows 9 frames of the Table Tennis video sequence for demonstration.

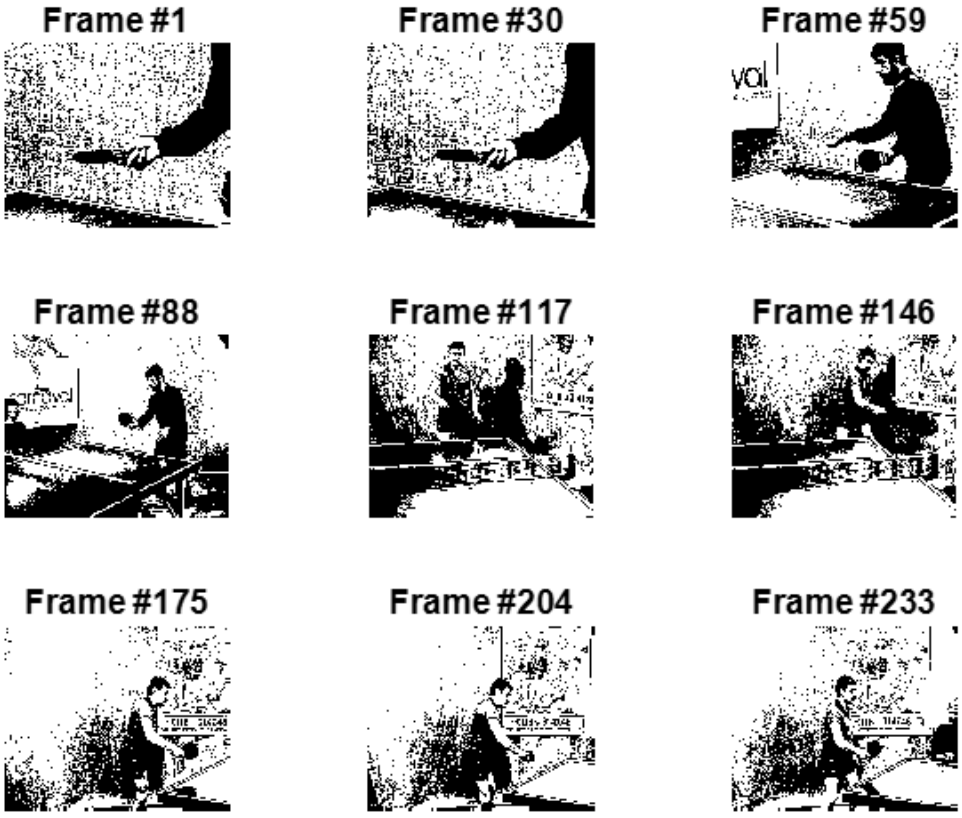


Figure 5.1: Binary images obtained from a video sequence.

5.1 Simulation Results of the Proposed Framework

The images are extracted from the YUV video sequence and after converting the grayscale images to binary images, we fed these binary images to the proposed BPSO algorithm. All the particles are initialized with random positions (i.e., scan direction), in the first iteration. Figure 5.2 shows reduction in the compressed file sizes with every iteration, corresponding to the frames in Figure 5.1. 100 iterations were run on each image to find out the best file size. Some of the images reach the optimal solution rather quickly (in less than 50 iterations), whereas some of the images go through 70 iterations to find out the best solution. At the 100th iteration, we get one of the unique scan path for a given image found to be the most optimized one.

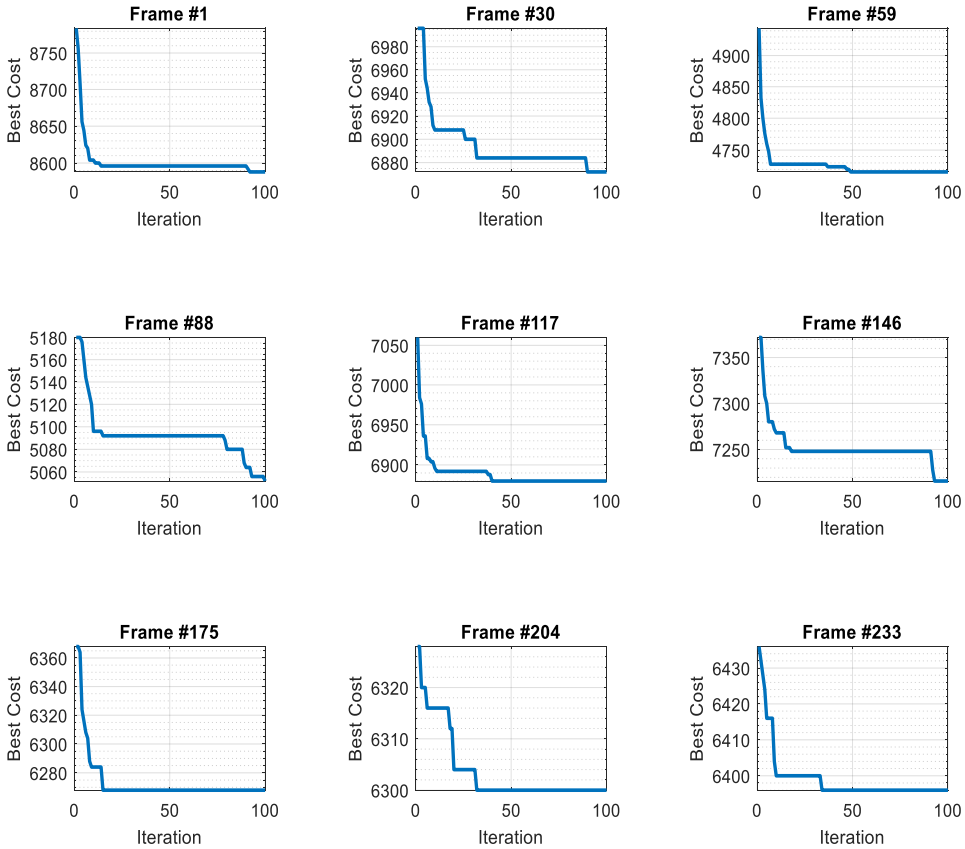


Figure 5.2: The cost function value (the compressed file size) changes with each iteration.

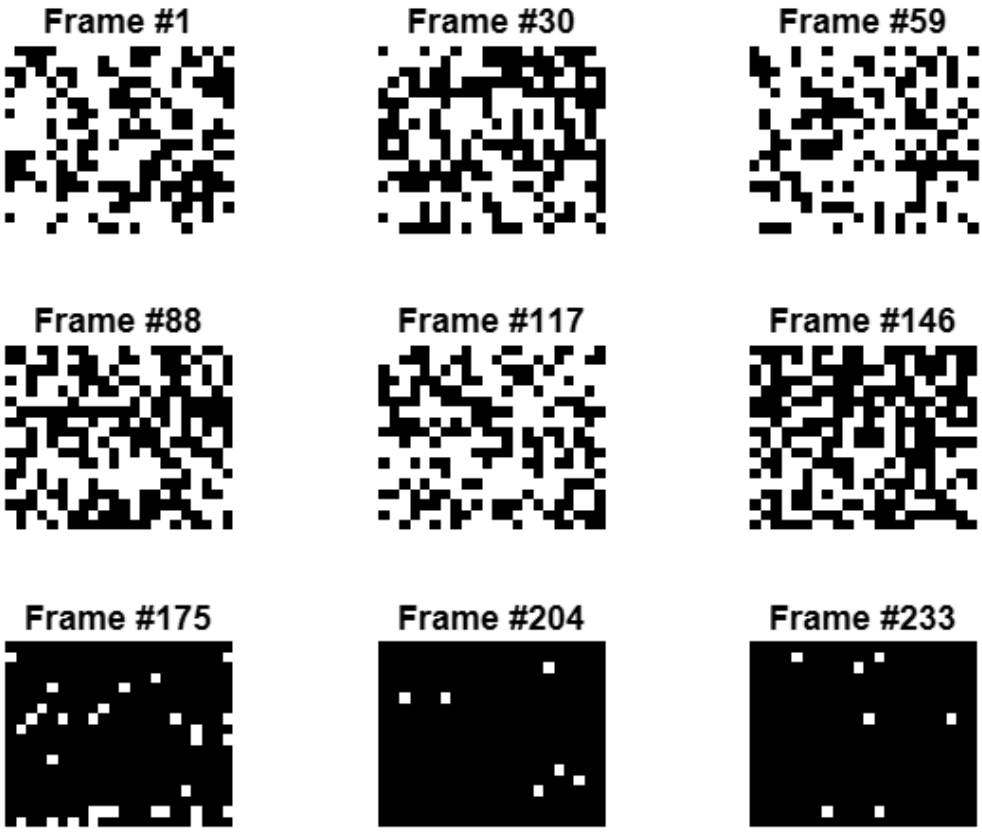


Figure 5.3 Bitmaps of the "Tennis" sequence using BPSO.

In the BPSO algorithm, at 100^{th} iteration, we generate a bitmap ("1" i.e., white for horizontal and "0" i.e., black for vertical scan) for viewing the scan path of the whole image. Typically, the bitmap gives a global view of the direction bits of the entire image. Figure 5.3 shows the bitmaps of the corresponding Table Tennis Sequence frames shown in Figure 5.1. Every image in a video sequence is slightly/significantly different from one another. Therefore, similar frames give similar bitmaps, as demonstrated for frame number 175, 204 and 233 and different bitmaps are generated for significantly different frames. But either more or less, every frame has different bitmap which implies that each image follows a different scan order for better compression. Similar bitmaps were obtained using tree-based algorithm as shown in Figure 5.4.

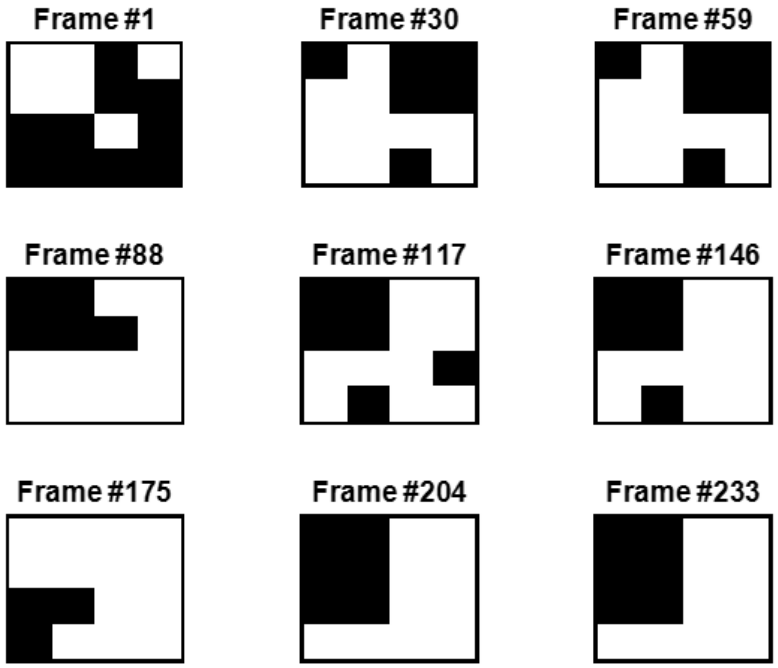


Figure 5.4 Bitmaps of the “Tennis” sequence using tree-based search method.

5.2 Comparison of Proposed Algorithms with Other Techniques

In this section, we provide simulation results of the proposed algorithms. Figure 5.5 shows the compression results for 9 frames in table tennis sequence. Test image index 1, 2, 3, 4, 5, 6, 7, 8 and 9 refers to frame 1, 30, 59, 88, 117, 146, 175, 204, 233 in the sequence, respectively. First, the optimization algorithm, BPSO was tested to examine its performance on binary images. Simulation results showed the scheme provides significantly higher compression than several existing lossless compression methods such as CCITT, FAX3 and JPEG 2000.

However, the BPSO method has lower compression than JBIG2 standard method. On the other hand, tree-based search algorithm achieves the highest compression with significant difference for most of the images (frame 5 to frame 9) in the sequence.

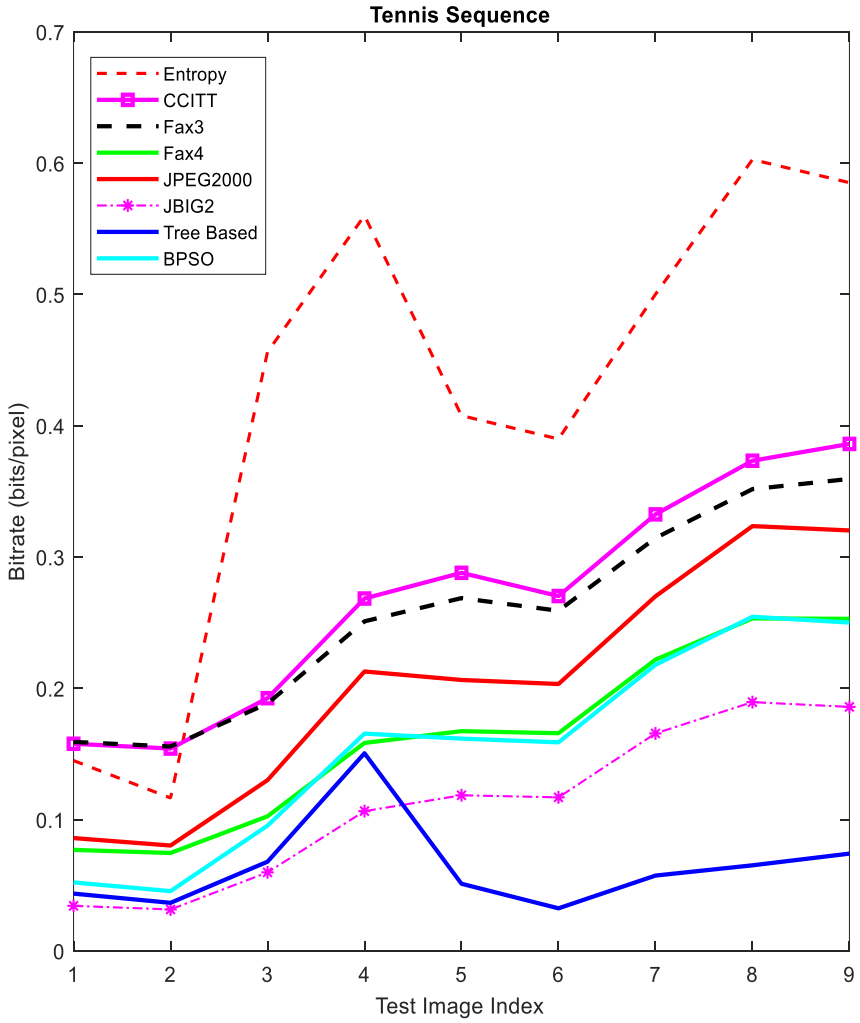


Figure 5.5: Compression results for tennis sequence.

The proposed methods were tested on several other data sets as well. Similar improvements were obtained for different types of images, demonstrating the advantage of dividing the image into variable size blocks. The “Bus” sequence (15 frames) acquired from video test media [46] was tested. Figure 5.6 shows selected frames of the bus sequence. Figure 5.7 shows the comparison of the BPSO and the tree-based search methods with other lossless coding schemes. Here the tree-based search method beats the JBIG2 method for frames 1 to frame 3 and frame 7 to frame 10.

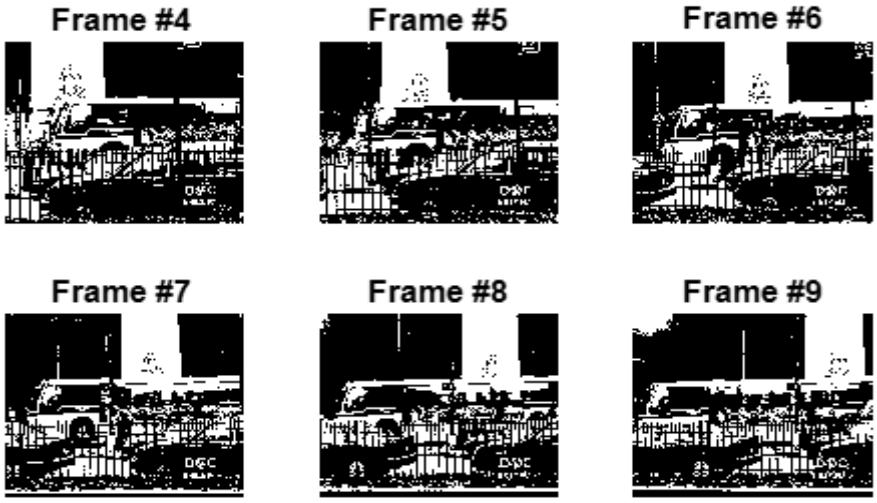


Figure 5.6: Binary images obtained from the “Bus” video sequence.

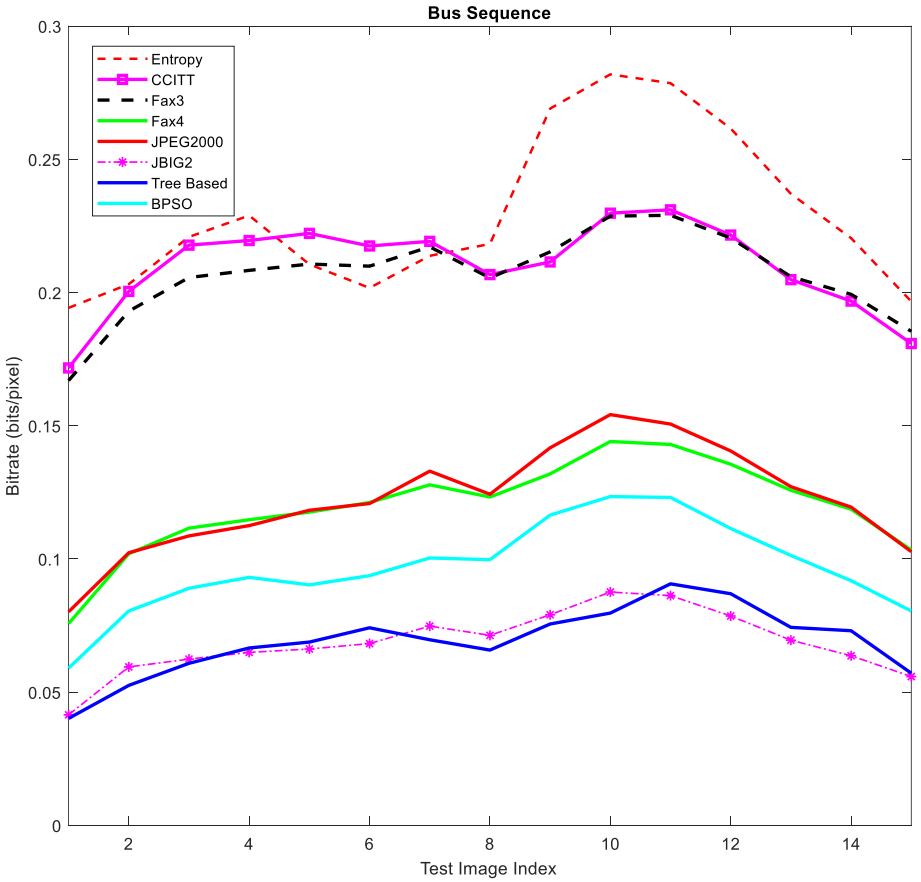


Figure 5.7: Compression results for the “Bus” sequence.

Hyperspectral images: Engineers build hyperspectral sensors and processing systems for application in astronomy, agriculture, biomedical imaging, geosciences, physics, and surveillance. A hyperspectral image is often organized as a three-dimensional data set with two spatial dimensions and one spectral dimension. One can achieve very large size reduction on hyperspectral image dataset by preserving some region-of-interest (ROI's) [52]. Figure 5.8 shows the ROI maps of hyperspectral sample dataset ("Indian Pines").

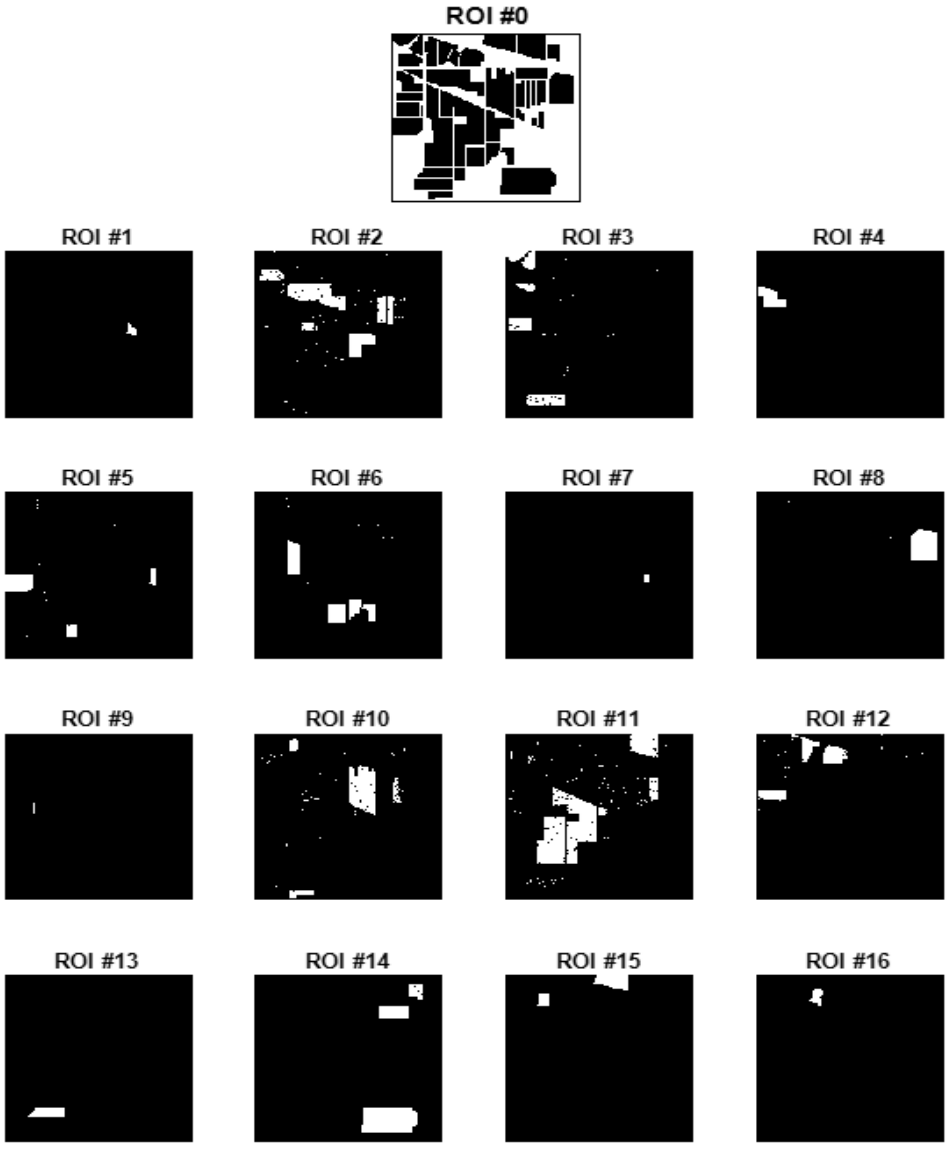


Figure 5.8: Classification label maps of the "Indian Pines (IP)" hyperspectral image dataset.

Region of interests are samples within a data set identified for a purpose. The bi-level map describes the location of the ROI pixels. ROIs can be defined by a user or they can be automatically identified by machine learning methods such as Support Vector Machine (SVM). We tested the proposed algorithms on ROI maps shown in Figure 5.8. We compressed all 17 ROI in the sample dataset (Indian Pines) and tested five other compressors. The results are plotted in Figure 5.9 for easy comparison and summarized in Table 5.1.

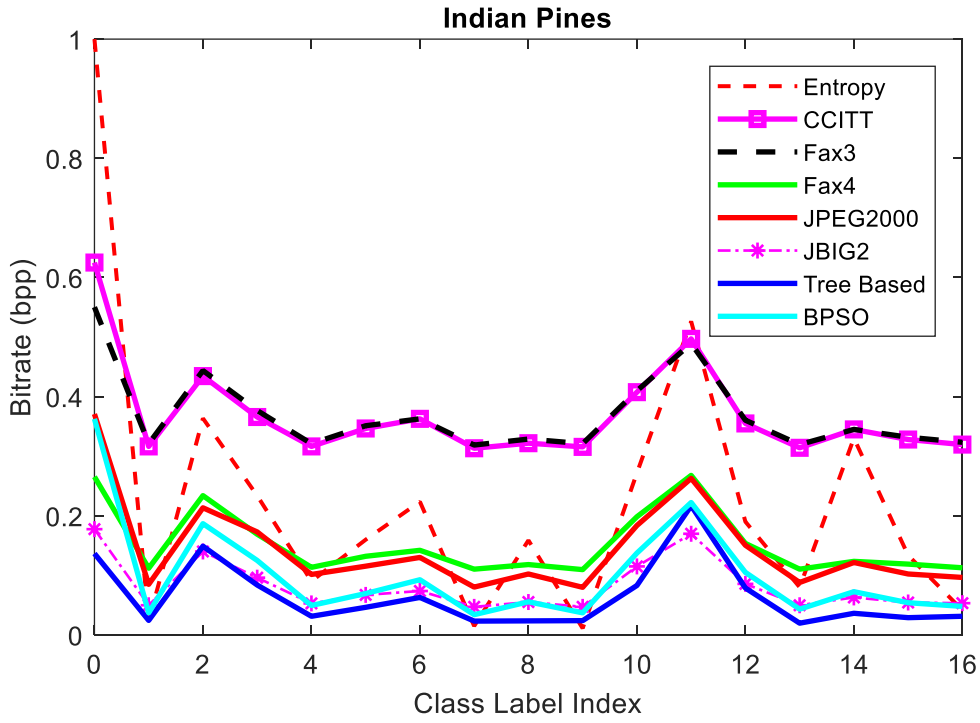


Figure 5.9: Compression results for bi-level IP ROI maps.

We observe that for ROI maps 1, 4, 7, 9, 13, and 16, BPSO outperforms all other compressors including JBIG2 standard. However, BPSO is slightly less efficient than JBIG2 for the rest of the ROI maps while still compressing more efficiently than other lossless compression methods, such as JPEG2000, FAX4, FAX3 and CCITT.

ROI #	CCITT	FAX3	FAX4	JPEG 2000	JBIG2	*BPSO*	*Tree-based*
0	0.625	0.5502	0.2654	0.3708	0.1775	*0.3627*	***0.1366***
1	0.3164	0.3210	0.1119	0.0849	0.0498	*0.037*	***0.0243***
2	0.4344	0.4437	0.2338	0.2133	**0.14**	*0.1867*	*0.1489*
3	0.3657	0.3765	0.1690	0.1728	0.0961	*0.125*	***0.0837***
4	0.3164	0.3210	0.1134	0.1015	0.0529	*0.0494*	***0.0313***
5	0.3465	0.3511	0.1319	0.1154	0.0671	*0.0694*	***0.0463***
6	0.3627	0.3627	0.142	0.13	0.0737	*0.0926*	***0.0629***
7	0.3133	0.3187	0.1103	0.0802	0.0471	*0.0340*	***0.0231***
8	0.3218	0.3287	0.1181	0.1022	0.0552	*0.0556*	***0.0235***
9	0.3156	0.3210	0.1096	0.0799	0.0471	*0.0370*	***0.0239***
10	0.4074	0.4097	0.1983	0.1836	0.115	*0.1373*	***0.0829***
11	0.4969	0.4892	0.2677	0.2627	**0.1698**	*0.2222*	*0.2168*
12	0.3549	0.3603	0.1535	0.1505	0.0856	*0.1049*	***0.0787***
13	0.314	0.3194	0.1103	0.088	0.0498	*0.0432*	***0.0197***
14	0.3449	0.3449	0.1235	0.1215	0.0629	*0.0725*	***0.0363***
15	0.3279	0.3318	0.1188	0.1022	0.0544	*0.0540*	***0.0289***
16	0.3194	0.3233	0.1127	0.0965	0.0532	*0.0478*	***0.0313***

Table 5.1: Bitrate of compressed files.

Tree-based algorithm achieves the best compression for all ROI maps 0 through 16, except for ROI maps 2 and 11, followed by either BPSO or JBIG2. With BPSO and JBIG2 being the second-best compression scheme, it is followed by either JPEG2000 or FAX4. Therefore, the proposed method achieved highest compression on 15 out of 17 ROI maps in hyperspectral images.

To evaluate the BPSO and tree-based search schemes, we tested them on another hyperspectral image dataset (600×340). Figure 5.10 shows the ROI maps of hyperspectral sample dataset (“Pavia University”) and Figure 5.11 shows simulation results for comparison.

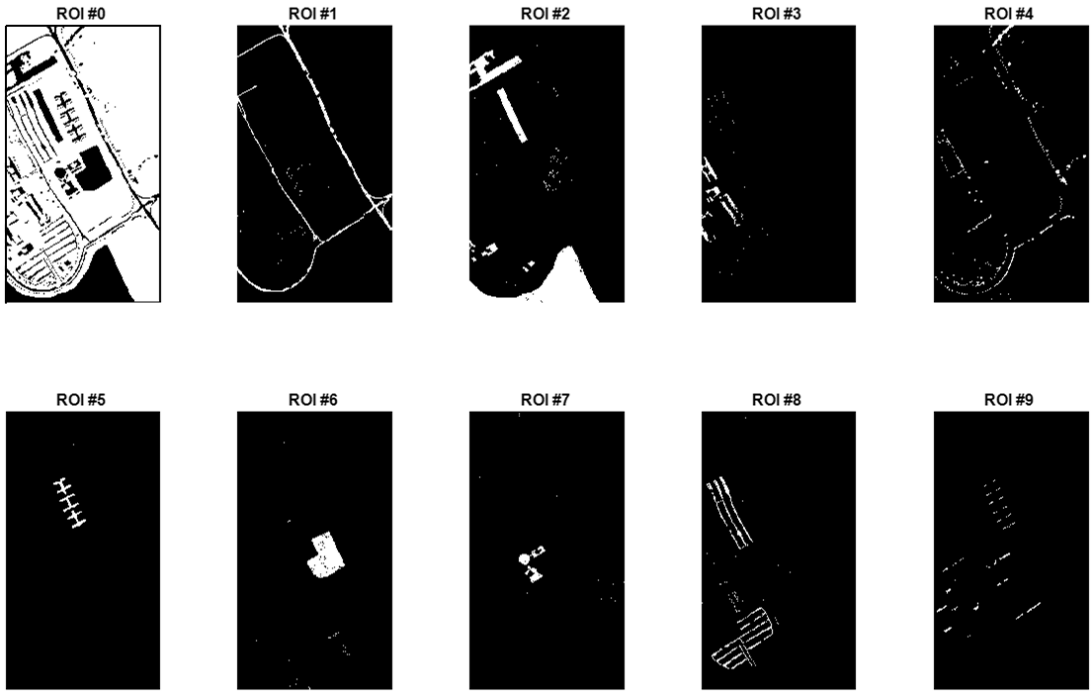


Figure 5.10: Classification label maps of the "Pavia University (PU)" dataset.

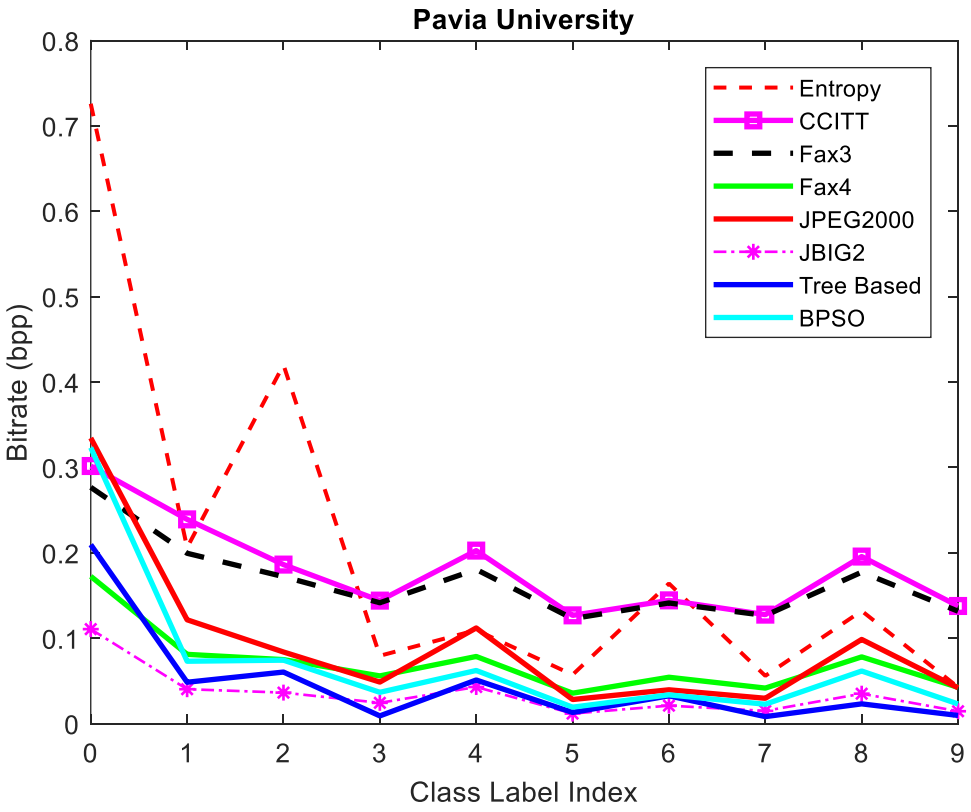


Figure 5.11: Compression results for bi-level PU ROI maps.

Also, for the “Pavia University” dataset we can observe that BPSO is performing very closely to JBIG 2 standard method, and the tree-based search algorithm performs the best on average for all the images.

Chapter 6

Conclusion and Future Expansion

In this work, we proposed two optimization and search methods for lossless compression of binary images. Chapter 3 describes the search and optimization algorithm BPSO, employed to improve compression. The algorithm explores different search paths to reach the most optimal one. There are several theoretical and practical issues that were addressed during the development of this work. This approach uses uniform block size to exploit redundancies in an image. We observed reduction in file size with every iteration resulting in improvement of compression.

A new compression scheme based on variable size blocks was introduced in Chapter 4, which divides the image using an adaptive grid structure. Non-uniform block size exploits different regions of the image based on its intrinsic nature. The advantage of dividing more concentrated regions have been demonstrated in the proposed tree-based algorithm. Simulation results in Chapter 5 reveal that the proposed coding method achieved highest compression in the case of most of the images when compared to several other efficient lossless compression standards such as CCITT, FAX3, FAX4, JBIG2 and JPEG 2000. Thus, relaxing the constraint of fixed sized blocks seems to be able to further improve the compression efficiencies.

The proposed methods used two scan paths, hence operating in low-dimension space. A general extension to this work could be to increase the number of scan paths to choose from (for instance, by using the Hilbert scan or zig-zag scan, etc.), which will result in

generating different interval sequences. This will result in much larger search space to explore with the potential to achieving additional compression gains. Alternatively, we can explore using more advanced BPSO models.

In addition, the tree-based search scheme has only two levels, which limits the scope of this research. The two-level tree-based scheme can be extended to more levels, with the goal of increasing the compression significantly.

REFERENCES

[1] Cisco Visual Networking Index: Forecast and Methodology, Cisco, May 2015. [Online] Available: www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-next-generation-network/white_paper_c11-481360.html.

[2] E. Auchard, "Flickr Maps the World's Photo hotspots," *Reuters*, 19 Nov. 2007.

[3] D. D. Ridder, "Adaptive Methods of Image Processing," Delft University of Technology, Netherlands, page 5, 2001.

[4] J. Z. Wang, D. Geman, J. Luo, and R. M. Gray, "Real-World Image Annotation and Retrieval: An Introduction to the Special Section," in *Proc. of IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, pp. 1873- 1876, 2008.

[5] L. Liaghati, and W. D. Pan, "An Adaptive Interval Generation Method for Efficient Distance Coding of Binary Images," in *Proc. of SPIE 9443, Sixth International Conference on Graphic and Image Processing (ICGIP 2014)*, March 2015.

[6] W. Y. Wei, "An Introduction to Image Compression," National Taiwan University, Taipei, Taiwan, ROC, 2008.

[7] P. Y. Lin, "Basic Image Compression Algorithm and Introduction to JPEG Standard", National Taiwan University, Taipei, Taiwan, ROC, 2009

[8] G. E. Blelloch, "Introduction to Data Compression,*"* Computer Science Department, Carnegie Mellon University, 2013.

[9] A. L. Liaghati, and W. D. Pan, "Evaluation of Biased Run-Length Coding Method on Binary Image Generated by Modified Ising model," in *Proc. of IEEE SoutheastCon 2016*, Norfolk, VA, March-April 2016.

[10] A. Liaghati, W. D. Pan, and Z. Jiang, "Biased Run-Length Coding of Bi-Level Classification Label Maps of Hyperspectral," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (JSTARS)*, vol.10, no.10, pp. 4580-4588, Oct. 2017.

[11] A. L. Liaghati, H. Shen, and W. D. Pan, "An Efficient Method for Lossless Compression of Bi-Level ROI Maps of Hyperspectral Images," in *Proc. of IEEE Aerospace Conference*, Big Sky, Montana, March 2016.

[12] L. Liaghati, and W. D. Pan, "Improved Distance Coding of Binary Images by Run Length Coding of the Most Probable Interval," in Proc. of *IEEE SoutheastCon 2015*, Fort Lauderdale, FL, April 2015.

[13] J. A. Storer, "Data Compression: Methods and Theory " in *Computer Science Press*, ISBN 978-0716781561, Pergamum, 1994.

[14] S. Majumder, M. Madhusudhan and A. Ray, "A Comparative study of different coding schemes for Image Compression using Wavelet Transform," in *Proc. of National Conference on Emerging Trends in Engineering (NCETE)*, 2008.

[15] N. Memon, D. L. Neuhoff, and S. Shende, "An analysis of Some Common Scanning Techniques for Lossless Image Coding*," IEEE Trans. Image Process.,* vol. 9, no. 11, pp. 1837–1848, Nov. 2000.

[16] K. S. Thyagarajan and S. Chatterjee, "Fractal scanning for image compression" in *Proc. of Conference Record of the 25th Asilomar Conference on Signals, Systems and Computers*, pp. 467–471, Pacific Grove, CA, 1992.

[17] D. Wu and E. C. Tan, "Comparison of Lossless Image Compression Algorithms," in *Proc. of IEEE Conference on TENCON*, vol.1, pp.718 – 721, 1999.

[18] K. D. Sonal, "A Study of Various Image Compression Techniques," in *Proceedings of COIT, RIMT Institute of Engineering & Technology*, pp. 799-803, Pacific, 2000.

[19] S. Biswas, "Hilbert Scan and Image Compression," in *Proc. of 15th IEEE International Conference on Pattern Recognition*, pp. 201–220, Barcelona, Spain, 2000.

[20] A. Mulla, J. Baviskar et al., "Image Compression Scheme Based on Zig-Zag 3D-DCT and LDPC Coding," in *Proc. of International Conference on Advances in Computing, Communications and Informatics (ICACCI),* pp.2380-2384, 24-27 Sept, Delhi, India, 2014.

[21] Y. C. Hu and C.C. Chang, "A new lossless compression scheme based on Huffman coding scheme for image compression," *Signal Processing: Image Communication,* vol. 16, no. 4, pp. 367–372, 2000.

[22] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," in *Proceedings of the IEEE International Conference on Neural Networks,* vol. 4, Perth, Australia, 1995.

[23] J. Kennedy and R. C. Eberhart, "A Discrete Binary Version of the Particle Swarm Algorithm," *in Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, pp. 4104–4109, 1997.

[24] J. Cai and W. D. Pan, "On Fast and Accurate Block Based Motion Estimation Algorithms Using Particle Swarm Optimization," *Information Sciences,* vol. 197, pp. 53-64, August 2012.

[25] J. C. Bansal and K. Deep, "A Modified Binary Particle Swarm Optimization for Knapsack Problems," *Appl. Math. and Comput*, vol. 218, pp.11042-11061, 2012.

[26] I. Ibrahim, Z. Ibrahim, H. Ahmad, Z. Md. Yusof, "An Improved Multi-State Particle Swarm Optimization for Discrete Optimization Problem," in *7th Int'l Conference on Computational Intelligence, Communication Systems and Networks (CICSyN),* June. 2015.

[27] B. Jarboui, M. Cheikh, P. Siarry, A. Rebai, "Combinatorial particle swarm optimization (CPSO) for Partitional Clustering Problem," *Appl. Math. Computation*, vol. 192, pp. 337–45, 2007.

[28] I. Ibrahim et al., "A Novel Multi-State Particle Swarm Optimization for Discrete Combinatorial Optimization Problems," in *4th Int'l Conference on Computational Intelligence, Modelling and Simulation (CIMSiM),* pp.18-23, Sept. 2012.

[29] I. Ibrahim, H. Ahmad, Z. Ibrahim, M. F. Mat Jusoh, Z. Md. Yusof, S. W. Nawawi, K. Khalil, and M. A. Abdul Rahim, "Multi-State Particle Swarm Optimization for Discrete Combinatorial Optimization Problem," *International Journal of Simulation - Systems, Science and Technology,* vol. 15, no. 1, pp. 15-25, Feb. 2014.

[30] L. M. Palanivelu and P. Vijayakumar, "Effective Image Segmentation Using Particle Swarm Optimization for Image Compression in Multi Application Smart Cards," in *Proceedings of the World Congress on Information and Communication Technologies*, pp. 535–539, 2011.

[31] S. Keerthika and S. Vidhya, "Fractal Image Compression with Advanced Particle Swarm Optimization and Sorting", *International Journal of Computer Science Trends and Technology (IJCST)*-vol. 4 Issue 5, Sep-Oct 2016.

[32] G. Vahdati, M. Yaghoobi, and M. R. Akbarzadeh-T, "Fractal Image Compression Based on Particle Swarm Optimization and Chaos Searching," in *International Conference on Computational Intelligence and Communication Networks (CICN),* pp. 62-67, 2010.

[33] C. C. Tseng, J. G. Hsieh, and J. H. Jeng, "Fractal image compression using visual-based particle swarm optimization*," Image Vis. Comput*., vol. 26, no. 8, pp. 1154–1162, 2008.

[34] A. Muruganandham, R. S. D. W. Banu, "Adaptive Fractal Image Compression using PSO," *Proc. Comput. Sci.*, vol. 2, pp. 338–344, 2010.

[35] J. Blondin, Particle Swarm Optimization: A Tutorial. [Online]. Available from: (www.cs.armstrong.edu/ saad/csci8100/pso_tutorial.pdf), 2009.

[36] K. Zielinski and R. Laur, "Adaptive Parameter Setting for a Multi-Objective Particle Swarm Optimization Algorithm," in *Proc. of IEEE Congress on Evolutionary Computation*, Singapore, pp. 3585–3592, 2008.

[37] Mudita Juneja, S. K. Nagar, "Particle swarm optimization algorithm and its parameters: A review," in *International Conference on Control, Computing, Communication and Materials (ICCCCM)*, pp. 1-6, 2016.

[38] F.V.D. Bergh, A.P. Engelbrecht, "A study of particle swarm optimization particle trajectories," in *Information Sciences*, vol. 176, pp. 937–971, 2006.

[39] R. Kalatehjari, A. S. A Rashid, N. Ali, and M. Hajihassani, "The Contribution of Particle Swarm Optimization to Three-Dimensional Slope Stability Analysis," *The Scientific World Journal*, vol. 2014, Article ID 973093, 12 pages, 2014.

[40] Q. Bai, "Analysis of Particle Swarm Optimization Algorithm," *Computer and Information Science*, vol. 3, no. 1, pp. 180-184, 2010.

[41] J. Liang, P. Suganthan, "Dynamic Multi-Swarm Particle Swarm Optimizer," in *Proc. of IEEE Swarm Intelligence Symposium, 2005.*

[42] W. A. Pearlman, "Trends of Tree-Based, Set Partitioning Compression Techniques in Still and Moving Image Systems," in *Proc. of Picture Coding Symposium.*, pp. 1-8 Apr. 2001.

[43] S. Verma, A.K. Pandit, "Quad tree based Image compression," *TENCON 2008-2008 IEEE Region 10 Conference,* 19-21 Nov. 2008

[44] H. Radha, M. Vetterli, R. Leonardi, "Image Compression Using Binary Space Partitioning Trees," *IEEE Transactions on Image Processing,* vol. 5, Dec 1996.

[45] W. Y. Loh, "Classification and Regression Trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* vol. 1, no. 1, pp. 14-23, 2011.

[46] A. Said, A. Pearlman. "A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, June 1996.

[47] M. Kunt, M. Benard, and R. Leonardi, "Recent Results in High Compression Image Coding," *IEEE Trans. Circuits Syst.*, vol. CAS-34, no. 11, pp. 1306-1336, Nov. 1987.

[48] R. Leonardi and M. Kunt, "Adaptive Split-and-Merge for Image Analysis and Coding," in *Proc. of SPIE 0594, Image Coding,* vol. 594, 1985.

[49] G. J. Sullivan and R. L. Baker, "Efficient Quadtree Coding of Images and Video," *IEEE Transactions on Image Processing*, vol. 3, no. 3, pp. 2661-2664, May 1991.

[50] J. Vaisey and A. Gersho, "Image Compression with Variable Block Size Segmentation," *IEEE Trans. on Signal Processing*, vol. 40, no. 8, pp. 2040-2060, Aug. 1992.

[51] Video test media. [Online]. Available: http://media.xiph.org/video/derf/

[52] H. Shen, W. D. Pan, Y. Dong, and Z. Jiang, "Golomb-Rice Coding Parameter Learning Using Deep Belief Network for Hyperspectral Image Compression," in Proc. of *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Fort Worth, TX, July 2017.