# LOSSLESS COLOR IMAGE COMPRESSION WITH BIT-ERROR AWARENESS

by

**Xuan Peng**

**A Thesis**

*Submitted to the Faculty of Purdue University*

*In Partial Fulfillment of the Requirements for the degree of*

**Master of Science in Electrical and Computer Engineering**

Department of Electrical and Computer Engineering

Hammond, Indiana

December 2019

**THE PURDUE UNIVERSITY GRADUATE SCHOOL**

**STATEMENT OF COMMITTEE APPROVAL**

**Dr. Lizhe Tan, Chair**

Department of Electrical and Computer Engineering

**Dr. Quamar Niyaz**

Department of Electrical and Computer Engineering

**Dr. Xiaoli Yang**

Department of Electrical and Computer Engineering

**Approved by:**

Dr.  Vijay Devabhaktuni

*Dedicated to my parents Hui Peng and Huanping Ren*

## ACKNOWLEDGMENTS

Here, I would like to express my gratitude to all those who helped me during the writing of this thesis. I gratefully acknowledge the help of my academic advisor, Dr. Lizhe Tan, who has offered me valuable suggestions in the academic studies. Under his guidance, I have learned much knowledge and many methods for the research. Without his patient instruction, insightful criticism and expert guidance, the completion of this thesis would not be possible.

I also owe a special debt of gratitude to my other committee members, Dr. Quamar Niyaz and Dr. Xiaoli Yang for their feedbacks. Finally, I like to express my gratitude to my group members in my research for their help and brainstorming.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

## ABBREVIATIONS

| | |
|---|---|
| CR | Compression Ratio |
| PSNR | Peak Signal to Noise Ratio |
| ARQ | Automatic Repeat Request |
| FEC | Forward Error Correction |
| HEC | Hybrid Error Correction |
| GA | Genetic Algorithm |
| PSO | Particle Swarm Optimization |
| BER | Bit-error rate |

# ABSTRACT

Image compression is widely applied to medical imaging, remote sensing applications, biomedical diagnosis, multimedia applications and so on [1]-[4]. In many cases, considering the factor of image quality, we use a lossless compression method to compress the image.

In this thesis work, we propose bit-error aware lossless compression algorithms for color image compression subject to bit-error rate during transmission. Each of our proposed algorithms includes three stages. The first stage is to convert the RGB images to YCrCb images, and the second stage predicts the transformed images to generate the residue sequences. Optimization algorithms are used to search the best combination of the image conversion and prediction. At the last stage, the generated residue sequences are encoded by several residue coding algorithms, which are 2-D and 1-D bi-level block coding, interval Huffman coding and standard Huffman coding algorithms. Key parameters, such as color transformation information, predictor parameters and residue coding parameters, are protected by using (7,4) Hamming code during image transmission,

The compression ratio (CR) and peak signal to noise ratio (PSNR) are two significant performance indicators which are used to evaluate the experimental results. According to the experimental results, the 2-D bi-level block coding algorithm is verified as the best coding method.

# CHAPTER 1. INTRODUCTION

## 1.1 Motivations

Image compression means reducing the amount of data needed to represent digital images. Image data can be compressed because of the redundancy in the data. The redundancy of image data mainly manifests in the following three aspects: spatial redundancy caused by correlation between adjacent pixels in the image; temporal redundancy caused by correlation between different frames in the image sequence; spectral redundancy caused by correlation of different color planes or spectral bands. Since the amount of image data is huge, and it is very difficult to store, transfer, and process, compression of image data is demanded.

Due to the growing size of color image datasets, it is necessary to research more efficient ways to compress the images [5]-[8], where the methods for image compression are divided into lossy compression and lossless compression. The lossy compression methods can achieve high compression ratio at the cost of poor reconstruction images [9], [10]. When an exact recovery of a compressed image is required, we use lossless image compression algorithms [11], [12]. The lossless compression algorithm is especially useful for dealing with ECG signals, 2-D, 3-D or even 4-D medical images in [13], [14]. In the image compression process, the image quality and compression ratio are better after preprocessing of the original images. In this paper, we adopt two steps to preprocess the images. The first step is to convert RGB images to YCrCb images so that we can get a better image compression ratio. The second step is to use predictive encoders to reduce the correlation between pixels, thereby increasing the image compression ratio [15]-[17]. Through an optimization algorithm, that is, genetic algorithm or particle swarm optimization algorithm, the most suitable color conversion equations and predictors are selected. Finally, the residues produced from the best color space conversion equations and predictors are further compressed via a lossless compression algorithm. Using lossless image compression could improve transmission throughput if the compressed image data is transmitted over a noiseless communication system. However, if bit errors occur in a noisy channel during transmission or in the storage media, the recovered image can be damaged and become useless if the instantaneous coder such as standard Huffman coding is directly applied. Although this problem can be cured by applying a forward error control scheme, adding additional bits required by the error correction coding can significantly degrade the performance of the compression ratio and may even cause the expansion of image files. A strategy for applying the error control coding needs to be addressed and the corresponding performance needs to be verified.

## 1.2 Objectives

We develop bit-error aware lossless compression algorithms using color transformation, prediction, and four different residue encoding methods, which are 1-D and 2-D bi-level block coding, interval Huffman coding, and standard Huffman coding. The compression ratio (CR) and peak signal to noise ratio (PSNR) are used to evaluate their performances.

Bit errors are unavoidable during data transmission. The error-correcting codes are essential. In this paper, we use (7,4) Hamming code reported in [18], [19] to protect key information of the compression algorithm in order to prevent images from being destroyed.

## 1.3 Organization of Thesis

This thesis is organized as follows. Chapter 2 introduces the framework of bit-error aware lossless color image compression. In Chapter 3, the two optimization algorithms, genetic algorithm and particle swarm optimization algorithm, are highlighted and compared. Then, we compare and analyze experimental results in Chapter 4. Finally, Chapter 5 presents the conclusion, contribution, and future work.

## 1.4 Contribution of Thesis

This thesis has four contributions. Firstly, we propose conversion methods from RGB to YCrCb, and predictors on YCrCb images to improve the CR and PSNR. Secondly, genetic algorithm and particle swarm optimization algorithm are proposed to select the best lossless conversion from RGB to YCrCb along with the best predictors to obtain the minimum entropy of the residues. Thirdly, the theory of 1-D bi-level block coding is extended to N-D bi-level block coding. As a special case, 2-D bi-level block coding is applied for residue coding. Finally, the performances of using new 2-D bi-level block coding are validated in comparisons with using 1-D bi-level block coding, interval Huffman coding, and standard Huffman coding.

# CHAPTER 2. BIT-ERROR AWARE LOSSLESS COLOR IMAGE COMPRESSION

## 2.1 Framework

In the image transmission and compression, color images are typically transformed, encoded, and compressed under certain conditions of fidelity. Besides, it is necessary to remove redundant data and reduce the amount of extra data when compressing color images to facilitate image storage and transmission. Figure 2.1 shows a block diagram of our bit-error aware lossless compression algorithms.

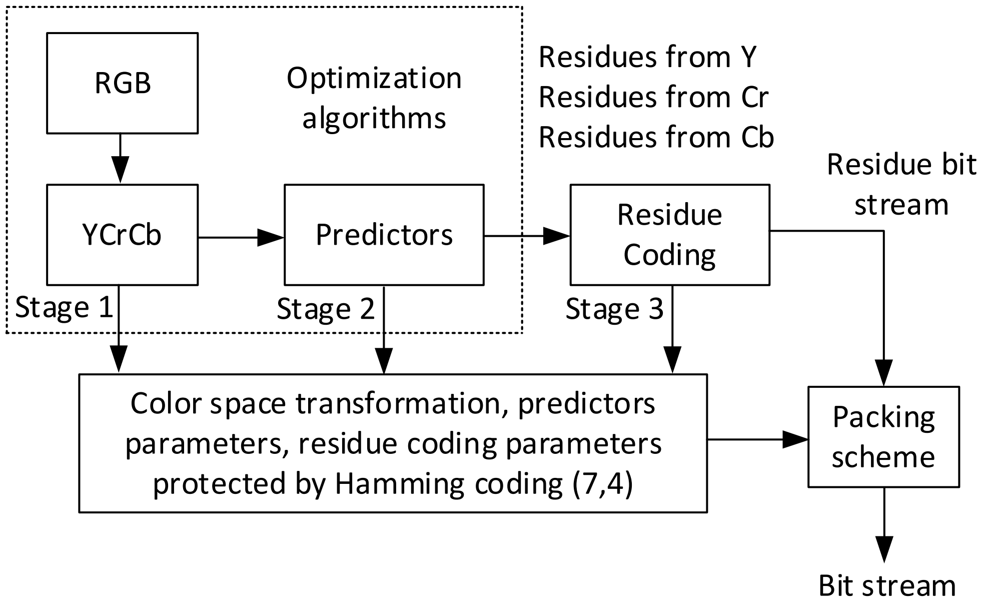


Figure 2.1 Bit-error aware three-stage lossless color image compression.

As shown in Figure 2.1, an RGB image is transmitted and compressed into a bit stream through three stages. At the first stage, the image becomes a YCrCb image by color space conversion, where Y is the luminance component, and Cr and Cb are the red-difference and blue-difference chrominance components. Next, the color-converted image components of YCrCb are predicted using the selected predictors and the residues of Y, Cr and Cb are produced. In order to effectively remove redundancy in the residues, an optimization algorithm is adopted to search the best combination of color space conversion for image in the first stage and linear predictor in the second stage. Then, the residue sequence produced by the first two stages is further encoded using one of four different residue coding methods. Consider the fact that the compressed bit stream may be interfered by bit errors during transmission. (7,4) Hamming code is effective in preventing image damage by protecting key information, that is, color space transformation, predictors parameters and residue coding parameters. Finally, a bit stream is produced for transmission or storage by packing these key parameters and unprotected residue bit stream. Through the reverse process, we can decode the bit stream back to obtain the restored image.

## 2.2 Color Space Transformation

YCrCb is mainly used to optimize the transmission of color video signals, making them backward compatible with old-fashioned black and white TVs. Compared with RGB signal transmission, its biggest advantage is that it requires only a small amount of bandwidth (RGB requires three independent signals to be transmitted simultaneously). “Y” indicates brightness (Luminance or Luma), which is the grayscale value. In RGB images, brightness is established by RGB input signals by superimposing specific parts of the RGB components. On the other hand, “Cr” and “Cb” indicate the chroma (Chrominance or Chroma), which are used to describe the color and saturation of an image to specify the color of pixels. Among them, “Cr” reflects the difference between the red part of the RGB input signals and the brightness value of the RGB signals. “Cb” reflects the difference between the blue portion of the RGB input signals and the luminance value of the RGB signals.

The principle is that in YCrCb format, the luminance channel carries more signal energy, while the chrominance channels carry much less signal energy. After transformation, more effort can be spent on coding the luminance channel [20]. According to this principle, the converted image has a higher compression ratio and image quality than RGB image. Table 2.1 [21], [22] list some lossless transformation methods from RGB image to YCrCb image and vice versa, where nine (9) equations are related to RGB converted to Y, twelve (12) equations are related to RGB converted to CrCb.

Table 2.1 Transformation formulas for Y, Cr and Cb in color space

| *i* | Y |
|---|---|
| 1 | $G$ |
| 2 | $R$ |
| 3 | $B$ |
| 4 | $\lfloor (G+R)/2 \rfloor$ |
| 5 | $\lfloor (G+B)/2 \rfloor$ |
| 6 | $\lfloor (R+B)/2 \rfloor$ |
| 7 | $\lfloor (R+2G+B)/4 \rfloor$ |
| 8 | $\lfloor (2R+G+B)/4 \rfloor$ |
| 9 | $\lfloor (R+G+2B)/4 \rfloor$ |

| *j* | Cr | Cb |
|---|---|---|
| 1 | $R-G$ | $B-G$ |
| 2 | $G-R$ | $B-R$ |
| 3 | $R-B$ | $G-B$ |
| 4 | $R-G$ | $B-\lfloor (R+3G)/4 \rfloor$ |
| 5 | $G-R$ | $B-\lfloor (G+3R)/4 \rfloor$ |
| 6 | $R-B$ | $G-\lfloor (R+3B)/4 \rfloor$ |
| 7 | $B-G$ | $R-\lfloor (B+3G)/4 \rfloor$ |
| 8 | $G-B$ | $R-\lfloor (G+3B)/4 \rfloor$ |
| 9 | $B-R$ | $G-\lfloor (B+3R)/4 \rfloor$ |
| 10 | $R-G$ | $B-\lfloor (R+G)/2 \rfloor$ |
| 11 | $R-B$ | $G-\lfloor (R+B)/2 \rfloor$ |
| 12 | $B-G$ | $R-\lfloor (B+G)/2 \rfloor$ |

As an example for 512×512 “Lena” image, according to the maximum compression criterion, the genetic algorithm or particle swarm optimization algorithm selects the 8th transformation formula for Y space ($\lfloor (2R+G+B)/4 \rfloor$) and the 1st transformation formula for CrCb, that is,

$$Cr = R - G\,, \tag{2.1}$$

$$Cb = B - G\,. \tag{2.2}$$

We can recover the RGB image from the YCrCb image as:

$$R = Y + floor\{(2Cr - Cb)/4\}\,, \tag{2.3}$$

$$G = R - Cr\,, \tag{2.5}$$

$$B = Cb + G\,. \tag{2.6}$$

For 512×512 “Baboon” image, the transformation formulas for Y at the sixth, and CrCb at the eleventh row in Table 2.1 are adopted. The YCrCb image is converted to the RGB image as follows:

$$G = C\mathrm{b} - Y\,, \tag{2.7}$$

$$B = Y - floor\{Cr/2\}\,, \tag{2.8}$$

$$R = Y + Cr - floor\{Cr / 2\}\ . \qquad (2.9)$$

## 2.3 Prediction

Statistical analysis of natural scenes and character images shows that pixels with low brightness levels have a substantial probability. After analyzing of a large number of image difference signal statistics, the difference signal near zero value has the highest probability of occurrence. Therefore, the actual discrete amplitude of two pixels in the horizontal direction (or vertical direction) of the image can be subtracted to obtain their difference, and then the difference is encoded and transmitted to achieve the purpose of compressing the image data. The image compression coding of the prediction method is developed on this basis.

The predictive coded data compression technique is based on the correlation of signal data. It predicts the new sample by using the previous sample values according to a certain model, thereby reducing the temporal and spatial correlation of the data to achieve the purpose of compressing the data. In this thesis work, we adopt two kinds of linear predictors reported in [21], [22], and [23] for Y, Cr and Cb, respectively, although there are many predictors designed using traditional least-square design methods [24]. According to the certain correlation between the pixels of the image, the pixel is predicted by using the surrounding pixels, and then the difference between the actual value and the pre-value (prediction error) is encoded. The more effective the prediction, the smaller the residue error and residue entropy are. Therefore, the fewer bits are needed for encoding each residue value. Figure 2.2 shows a typical linear predictor. As shown in Figure 2.2, X is the predicted pixel; and C, B and A are the known neighbor pixels of X used for the predictor.

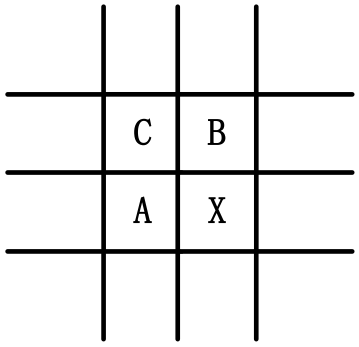


Figure 2.2 Neighbored pixels for the predictor in an image.

For a color image, we maintain the pixels in the first row and the first column, and implement the method showed in Figure 2.2. The pixel values of other rows and columns can be obtained according to the known pixel values of the first row and the first column. Then, residuals can be obtained by subtracting the predicted value from the original image. Table 2.2 lists two proposed predictors for linear predictive coding. Each of Y, Cr and Cb components is predicted using one of these two methods [11], [23], [25]. In the light of the maximum compression criterion, the best predictor can be adopted.

Table 2.2 Linear predictors for Y, Cr and Cb

| $i$ | $P(x)$ for Y | $P(x)$ for Cr | $P(x)$ for Cb |
|---|---|---|---|
| 1 | $(A+B)/2$ | $(A+B)/2$ | $(A+B)/2$ |
| 2 | $(3A+3B-2C)/4$ | $(3A+3B-2C)/4$ | $(3A+3B-2C)/4$ |

## 2.4 Residue Coding

After the prediction, the residues of predicted images can be compressed sequentially. It is assumed that the residue samples are uncorrelated and follow the Laplacian distribution approximately [26], [27]. We design four different methods to encode the residues, which are 2-D bi-level block coding, 1-D bi-level block coding, interval Huffman coding and standard Huffman coding.

### 2.4.1 2-D bi-level block coding and 1-D bi-level block coding

First, bi-level block coding is used, which is a simple and efficient method for encoding data sequences [28], [29], [30]. This method divides residues into two different blocks. One type of blocks contains smaller residue values, so it can be encoded with a smaller number of bits per residue. We call this type of bocks as level-1 block. Each sample in the block can be encoded using $N_1$ bits. The other type of blocks includes not only smaller residue values but also larger residue values. It requires a larger number of bits to encode each residue. This type of blocks is called as level-0 block. Each sample in the block must be encoded using $N_0$ bits ( $N_0 \rangle N_1$ ).

Let us derive for the general case. For an N-dimensional bi-level block coding method, assuming all data samples are statistically independent, the probability of a level-1 block is written as $P_1 = p^{x_1 x_2 \ldots x_N}$ , where $p = 1 - p_0$ is the probability of a data sample requiring less or equal to $N_1$ bits to encode and $p_0$ (close to zero) is the probability of a data sample requiring more than $N_1$ bits and less than or equal to $N_0$ bits to encode. The probability of a level-0 block is $P_0 = 1 - P_1 = 1 - p^{x_1 x_2 \cdots x_N}$ . For a sequence consisting of $m$ blocks in which there are $k$ level-1 blocks and ( $m - k$ ) level-0 blocks, the sequence coding length and its probability are, respectively, given below:

$$L(k) = m + N_0 x_1 x_2 \cdots x_N (m-k) + N_1 x_1 x_2 \cdots x_N k\,, \tag{2.10}$$

$$P(k) = \binom{m}{k} P_1^k \left(1 - P_1\right)^{m-k} = \binom{m}{k} p^{x_1 x_2 \cdots x_N k} \left(1 - p^{x_1 x_2 \cdots x_N}\right)^{m-k}, \tag{2.11}$$

We can obtain the average total length $L_{ave}$ as

$$\begin{aligned} L_{ave} &= \sum_{k=0}^{m} P(k) L(k) \\ &= (m + N_0 x_1 x_2 \cdots x_N m) \sum_{k=0}^{m} P(k) - (N_0 - N_1) x_1 x_2 \cdots x_N \sum_{k=0}^{m} k P(k)\,, \\ &= (m + N_0 x_1 x_2 \cdots x_N m) - (N_0 - N_1) x_1 x_2 \cdots x_N m p^{x_1 x_2 \cdots x_N} \end{aligned} \tag{2.12}$$

Assuming that $x_1 x_2 \cdots x_N p_0 \le 0.3$, we can approximate the probability $P_1$ as

$$\begin{aligned} P_1 &= p^{x_1 x_2 \cdots x_N} = (1 - p_0)^{x_1 x_2 \cdots x_N} \\ &= 1 - p_0 x_1 x_2 \cdots x_N + \ldots \approx 1 - p_0 x_1 x_2 \cdots x_N \end{aligned}, \tag{2.13}$$

by omitting the higher-order terms. Using $n = x \times y \times m$, we obtain

$$L_{ave} = \frac{n}{x_1 x_2 \cdots x_N} + n N_1 + (N_0 - N_1) n x_1 x_2 \cdots x_N p_0\,, \tag{2.13}$$

For a fixed $N_1$, taking derivative of (2.13) to $x$ and setting it to zero leads to

$$\frac{\partial L_{ave}}{\partial x_1} = -\frac{n}{x_1^2 x_2 \cdots x_N} + (N_0 - N_1) n x_2 \cdots x_N p_0 = 0\,, \tag{2.14}$$

$$\frac{\partial L_{ave}}{\partial x_2} = -\frac{n}{x_1 x_2^2 \cdots x_N} + (N_0 - N_1) n x_1 x_3 \cdots x_N p_0 = 0\,, \tag{2.15}$$

$$\cdots$$

$$\frac{\partial L_{ave}}{\partial x_N} = -\frac{n}{x_1 x_2 \cdots x_N^2} + (N_0 - N_1) n x_1 x_2 \cdots x_{N-1} p_0 = 0\,, \tag{2.16}$$

Then, we yield the optimal block size and the minimum average bits per sample as:

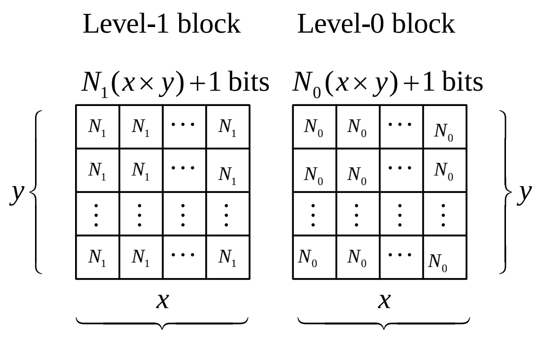
$$(x_1 x_2 \cdots x_N)^* = 1/\sqrt{(N_0 - N_1)\, p_0}\ , \tag{2.17}$$

$$\left(\frac{L_{ave}}{n}\right)_{\min} = 2\sqrt{(N_0 - N_1)\, p_0} + N_1\,. \tag{2.18}$$

2-dimensional bi-level block coding method is a special case in the N-dimensional bi-level block coding method, which is suitable for encoding color images. Table 2.3 shows the rules of the 2-D bi-level block coding algorithm.

Table 2.3 2-D bi-level block coding rules

1. Divide the data sequence with a length of $n = m \times x \times y$ into $m \le$ blocks in which each block consists of $x$ columns and $y$ rows, that is, $x \times y$ is the block size. There are two types of blocks, the level-0 block and the level-1 block.



2. For a level-1 block, any sample in the block requires only $N_1$ bits ( $N_1 < N_0$ [original sample size]) to encode. Encode each sample using $N_1$ bits and add the prefix "1" to designate the block as the level-1 block.

3. For a level-0 block, at least one of the samples in the block needs more than $N_1$ bits to encode. Encode each sample in the block using $N_0$ bits and add the prefix "0" to indicate the level-0 block.

Assuming all data samples are statistically independent the probability of a level-1 block is written as $P_1 = p^{xy}$, where $p = 1 - p_0$ is the probability of a data sample requiring less or equal to $N_1$ bits to encode and $p_0$ (close to zero) is the probability of a data sample requiring more than $N_1$ bits and less than or equal to $N_0$ bits to encode. The probability of a level-0 block is $P_0 = 1 - P_1 = 1 - p^{xy}$. For a sequence consisting of $m$ blocks in which there are $k$ level-1 blocks and ( $m-k$ ) level-0 blocks, the sequence coding length and its probability are, respectively, given below:

$$L(k) = m + N_0 xy(m-k) + N_1 xyk\,, \tag{2.19}$$

$$P(k) = \binom{m}{k} P_1^k \left(1 - P_1\right)^{m-k} = \binom{m}{k} p^{xyk} \left(1 - p^{xy}\right)^{m-k}, \tag{2.20}$$

We can obtain the average total length $L_{ave}$ as

$$\begin{aligned} L_{ave} &= \sum_{k=0}^{m} P(k)L(k) \\ &= (m + N_0 xym)\sum_{k=0}^{m} P(k) - (N_0 - N_1)xy\sum_{k=0}^{m} kP(k)\,, \\ &= (m + N_0 xym) - (N_0 - N_1)xymp^{xy} \end{aligned} \tag{2.21}$$

Assuming that $xyp_0 \leq 0.3$ , we can approximate the probability $P_1$ as $P_1 = p^{xy} = (1 - p_0)^{xy} = 1 - p_0 xy + ... \approx 1 - p_0 xy$ by omitting the higher-order terms. Using $n = x \times y \times m$, we obtain

$$L_{ave} = \frac{n}{xy} + nN_1 + (N_0 - N_1)nxyp_0\,, \tag{2.22}$$

$$xyp_0 < \gamma\,, \tag{2.23}$$

For a fixed $N_1$, taking derivative of (2.22) to $x$ and setting it to zero, we yield the optimal block size and the minimum average bits per sample

$$\frac{\partial L_{ave}}{\partial x} = -\frac{n}{x^2 y} + (N_0 - N_1)nyp_0 = 0\,, \tag{2.24}$$

$$\frac{\partial L_{ave}}{\partial y} = -\frac{n}{xy^2} + (N_0 - N_1)nxp_0 = 0\,, \tag{2.25}$$

Then, we yield the optimal block size and the minimum average bits per sample as:

$$(xy)^* = 1/\sqrt{(N_0 - N_1)p_0}\,, \tag{2.26}$$

$$\left(\frac{L_{ave}}{n}\right)_{\min} = 2\sqrt{(N_0 - N_1)p_0} + N_1\,. \tag{2.27}$$

The optimal coding parameters are the pair of $N_1$ and $(xy)^*$ corresponding to the smallest $(L_{ave}/n)_{\min}$ through the entire search for $1 \le N_1 < N_0$. We initially set $N_1 = N_0 - 2$, and $xy = 4$ in case $xyp_0 \le 0.3$ is not satisfied for all the searches. We summarize the 2-D bi-level block coding scheme in Table 2.4.

Table 2.4 2-D bi-level block coding algorithm

```
1. Find $N_0$ for a given data sequence.
Initially, set $N_1 = N_0 - 2$ and $(x \times y)^* = 4$.
2. For $N_1 = 1, 2, 3, N_0 - 1$
    Estimate $p_0$, the probability of the sample requiring more than $N_1$
    bits to encode; and calculate the optimal block size:
                $(x \times y)^* = 1/\sqrt{(N_0 - N_1)p_0}$
    Round up the block size to an integer value.
        If $(x \times y)^* \times p_0 \le 0.3$, calculate the average bits per sample:
                $(L_{ave}/n)_{\min} = 2\sqrt{(N_0 - N_1)p_0} + N_1$
        Record $N_1$ and $(xy)^*$ values for the next comparison
  End loop
After completing search loops, select $N_1$ and $(xy)^*$ corresponding to the
smallest value of $(L_{ave}/n)_{\min}$.
3. Perform bi-level block coding using the obtained optimal parameters $N_1$,
$x$ and $y$ which satisfy $(xy) = (xy)^*$, and rules listed in Table 2.3.
```

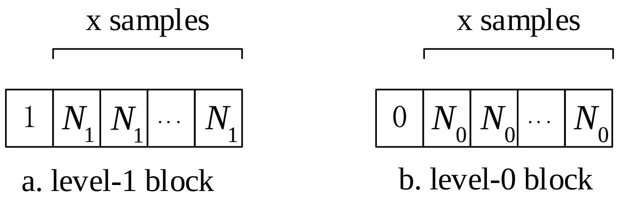
Unlike 2-dimensional bi-level block coding method, 1-dimensional bi-level block coding method divides every line of residues into two different blocks: level-1 block and level-0 block. The rule of 1-D bi-level block coding method is shown in Table 2.5.

Table 2.5 1-D bi-level block coding rules

1. Each line of residue data is divided into $m$ blocks with each block size of $x$, so $n = m \times x$ is the total number residues. There are two types of blocks: level-1 block and level-0 block.

2. For a level-1 block, each sample in the block can be encoded using $N_1$ bits. Add the prefix "1" to show the level-1 block.

3. For a level-0 block, at least one residue existing in the block requires more than $N_1$ bits to encode. This means that each sample in the block must be encoded using $N_0$ bits ($N_0 \rangle N_1$). Add the prefix "0" to indicate the level-0 block.



Assume probability of level-1 blocks is $P_1 = p^x$. The probability of level-0 blocks is $P_0 = 1 - P_1 = 1 - p^x$. For m blocks of residue data, there are $k$ level-1 blocks and ($m-k$) level-0 blocks. The sequence coding length and its probability are, respectively, given as follows [30]:

$$L(k) = m + N_0 x(m-k) + N_1 xk\ , \tag{2.28}$$

$$P(k) = \binom{m}{k} P_1^k (1-P_1)^{m-k} = \binom{m}{k} p^{xk}(1-p^x)^{m-k}\ , \tag{2.29}$$

We can obtain the total average length $L_{ave}$ as

$$L_{ave} = \sum_{k=0}^{m} P(k)L(k) = (m + N_0 xm) - (N_0 - N_1)xmp^x\ . \tag{2.30}$$

The optimal coding parameters are $N_1$ and $x^*$ corresponding to the smallest $(L_{ave}/n)_{\min}$, which is the averaged value in terms of bits per residue. Note that the total number of residues is $n = m \times x$.

The derivation of the formula can be found in [27], [30]. Table V lists the process of searching the optimal block size.

Table 2.6 1-D bi-level block coding algorithm

```
1. Find $N_0$ for a given data sequence.
Initially, set $N_1 = N_0 - 2$ and $x^* = 4$.
2. For $N_1 = 1, 2, 3, N_0 - 1$
    Estimate $p_0$, the probability of the sample requiring more than $N_1$
    bits to encode; and calculate the optimal block size:
                $x^* = 1/\sqrt{(N_0 - N_1) p_0}$
    Round up the block size to an integer value.
        If $x^* \times p_0 \le 0.3$, calculate the average bits per sample:
                $\left(L_{ave} / n\right)_{\min} = 2\sqrt{(N_0 - N_1) p_0} + N_1$
        Record $N_1$ and $x^*$ values for the next comparison
   End loop
After completing search loops, select $N_1$ and $x^*$ corresponding to the
smallest value of $\left(L_{ave} / n\right)_{\min}$.
3. Perform bi-level block coding using the obtained optimal parameters $N_1$,
$x$, and rules listed in Table 2.5.
```

In the process of transmission, it is inevitable that bit errors problem will be encountered. In order to prevent the image from being damaged by bit errors, the correction code is applied to the image transmission process to protect the crucial parameters of the image. Here, the error correction code we use is (7,4) Hamming code. As shown in Figure 2.3, (7,4) Hamming code protects color space transformation of Y and CrCb, predictors parameters, first row and first column of original image, bi-level block coding parameters and block types.

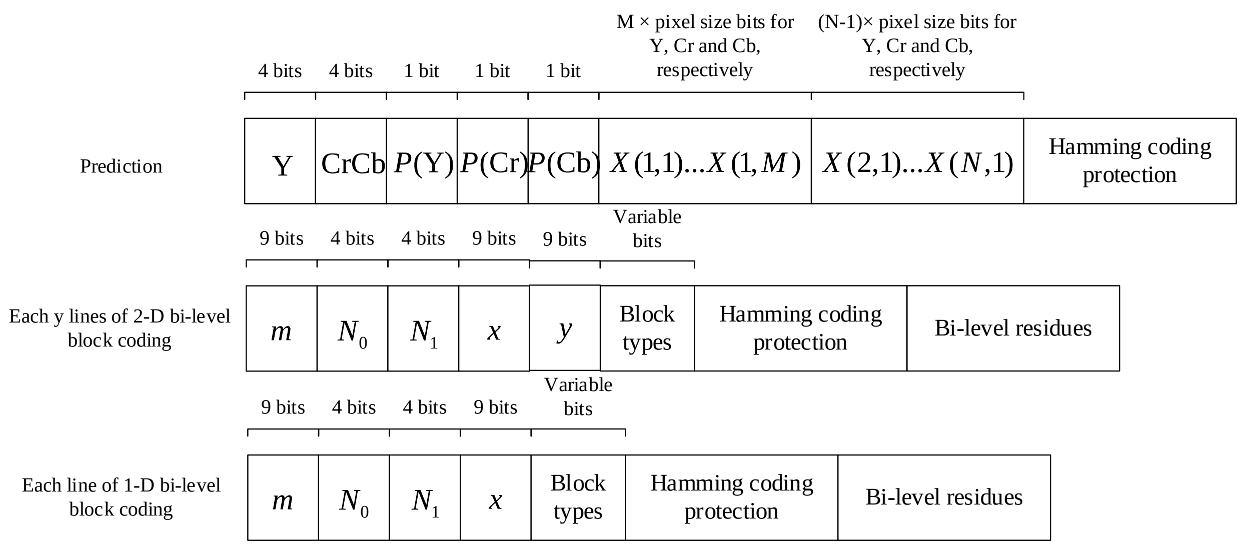


Figure 2.3 Coding protection for bi-level block coding method.

### 2.4.2 Interval Huffman coding and standard Huffman coding

In computer science and information theory, Huffman coding is a lossless statistical coding method that uses the characteristics of the probability distribution of information symbols to adapt the word length for coding. This method is developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper “A Method for the Construction of Minimum-Redundancy Codes” [31].

Huffman coding assigns codes of different lengths depending on the frequency at which each signal in the data appears. The basic idea is that in the encoding process, the shorter code length is allocated to the higher the frequency of occurrence, while the longer coded length is assigned to the lower the frequency.

For example, assuming that there are five characters: A, B, C, D, E, and the weights are 5, 4, 3, 2, 1, respectively. Then we take the first two weights as the left and right subtrees. Construct a new tree, that is, take 1, 2 to form a new tree, and its node is 1+2=3, as shown in Figure 2.4:

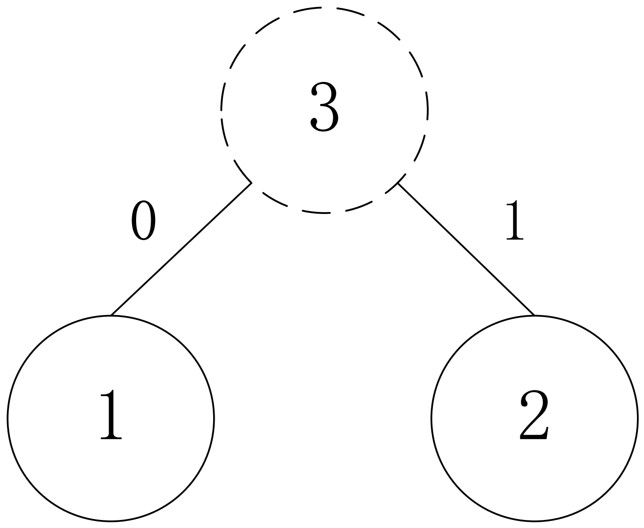


Figure 2.4 First step in example.

The dotted line is the newly generated node, and the second step puts the newly generated node with the weight 3 into the remaining set, so the set becomes {5, 4, 3, 3}, and then according to the second step, the minimum two weights are taken to form a new tree, as shown in Figure 2.5:

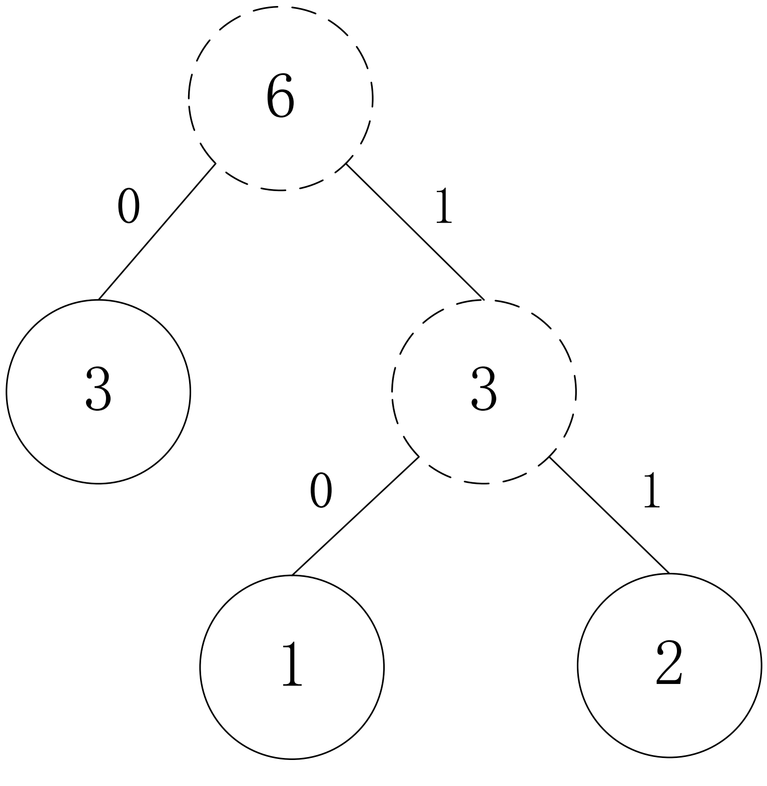


Figure 2.5 Second step in example.

Then, the Huffman tree is built in turn, as shown in Figure 2.6:

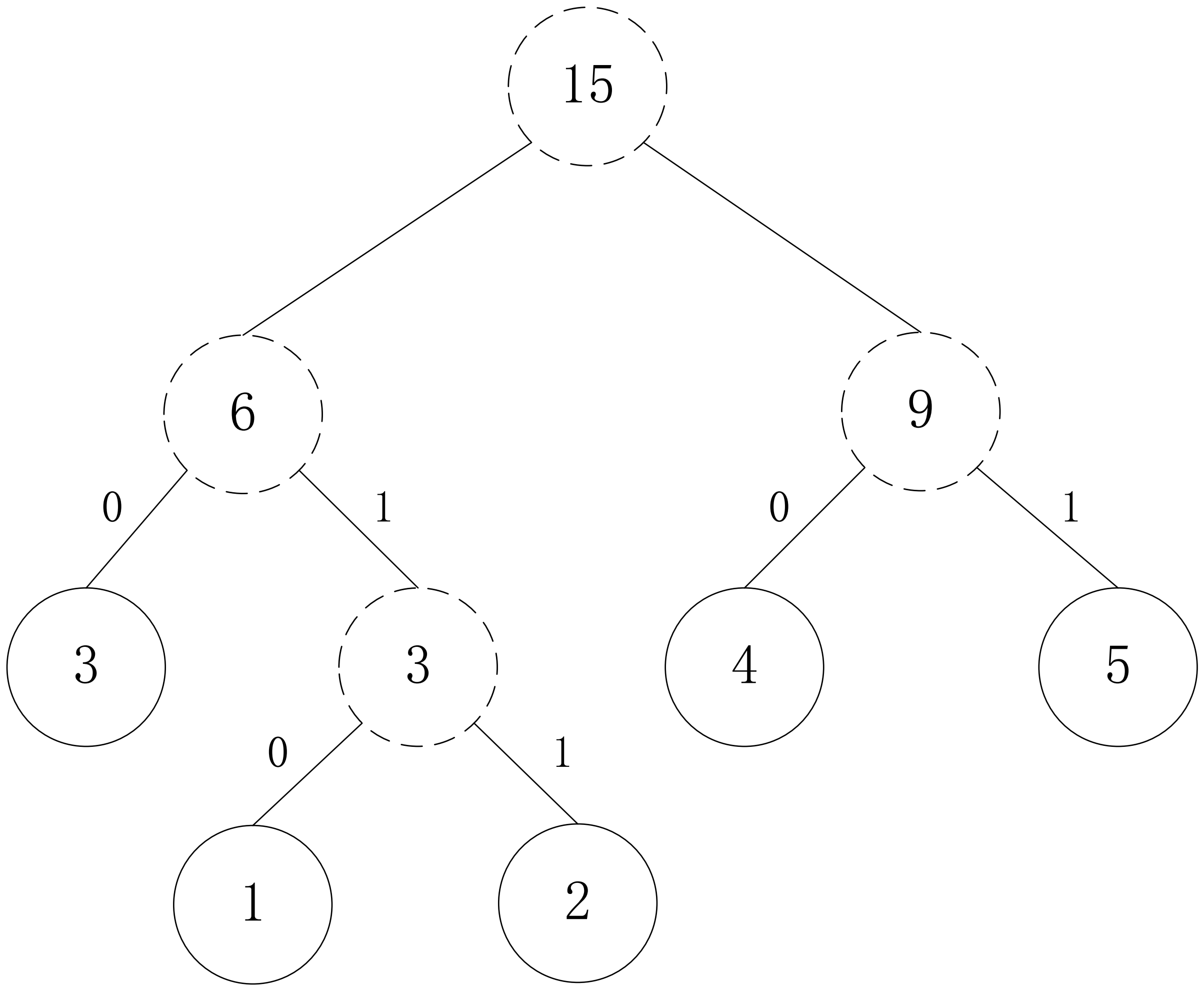


Figure 2.6 Final step in example.

The corresponding characters in each weight are replaced in Figure 2.7.

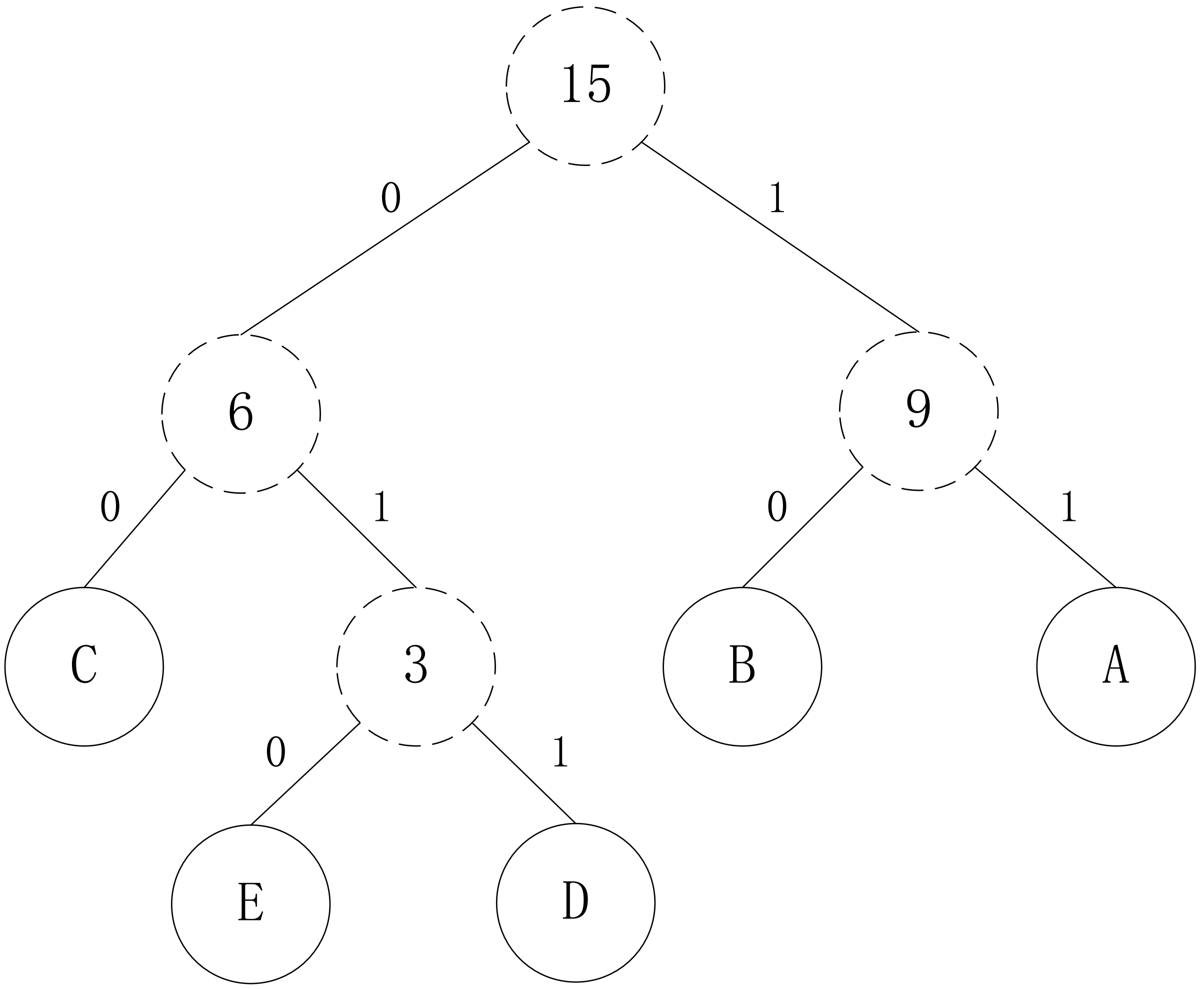


Figure 2.7 Huffman tree.

Therefore, the codes corresponding to each character are: A->11, B->10, C->00, D->011, E->010.

In this thesis work, beside 2-D and 1-D bi-level blocking mentioned above, we adopt two methods of Huffman coding. One is the interval Huffman coding method; and the other one is the standard Huffman coding method. Interval Huffman coding is an entropy coding, which can divide the residues from predictor into different interval and its offset. The formula is given as [30]:

$$q(n) = floor\{r(n)/2^{(N_0-N_1)}\}, \tag{2.31}$$

$$offset = r(n) - 2^{(N_0-N_1)} \times q(n), \tag{2.32}$$

where $q(n)$ is symbol of interval, which is quantized from a residue $r(n)$. It is entropy encoded and error protected. $N_0$ and $N_1$ are the symbol size. Function $floor(x)$ rounds $x$ down to the nearest integer towards negative infinity. We assume that our entropy coder achieves $N_0 - \beta$ bits per sample, where $\beta = 1 \sim 2$ bits. Assuming that $q(n)$ follows a perfect Laplacian distribution, choosing the smaller simple size $N_1$ for the interval entropy coder will gain approximately the same compression performance. Our method considers the positive and negative signs of the coded residual interval, so the case of $N_1 = 1$ is not considered. Besides, the average coded residue size is less than its original size $N_0$. Therefore, we get the following formula [30]:

$$(N_1 - \beta)\eta + (N_0 - N_1) \le N_0, \tag{2.33}$$

where $\eta$ is the error control coding rate. According to the error correction code of (7,4) Hamming code, $\eta$ is equal to 7/4. Hence, the lower and upper limits of $N_1$ as [30]:

$$2 \le N_1 \le \frac{\beta\eta}{\eta - 1} \tag{2.34}$$

Allowing $\beta = 2$ bits, it is obviously that $N_1 \le 4$. The interval sequence with four symbols and a finite length does not follow the Gaussian distribution function very well, so we exclude $N_1 = 2$. We finally choose $N_1 = 3$, and it reaches the best result from our experiments. The interval Huffman codes are listed as follow in Table 2.7 [30].

Table 2.7 Interval Huffman coding

| $q(n)$ | Interval codes | $q(n)$ | Interval codes |
|---|---|---|---|
| 0 | 1 | +2 | 01011 |
| -1 | 00 | -3 | 010100 |
| +1 | 011 | +3 | 0101010 |
| -2 | 0100 | -4 | 0101011 |

For example, if the largest number of residue $r(n)$ is 31, $N_0$ is equal to 6. Consider that $N_1$ is 3 and residue $r(n)$ equals 20. According to the formulas (2.31) and (2.32), $q(n)$ is equal to +2, and offset is equal to 4. $q(n)$ is encoded as shown in Table 2.7, and offset is coded in binary. Considering image quality and image compression ratio, (7,4) Hamming code is used to protect the value of $q(n)$. Offset does not need to be protected.

Standard Huffman coding method is also used in this paper for comparison. The scheme is shown as Table 2.8 [32]. A residue from prediction is encoded using a prefix which describes code size, cascaded by the binary amplitude bits. To encode -3, -2, +2 and +3, for example, the results are 01100, 01101, 01110, and 01111, respectively. The first three numbers are the prefix code. In this work, the prefix part is protected using (7,4) Hamming code.

Table 2.8 Standard Huffman coding

| Code size (No. bits) | Amplitude code | Code size (No. bits) | Amplitude code |
|---|---|---|---|
| 00(0) | 0 | 110(5) | -31,⋯,-16,+16,⋯,+31 |
| 010(1) | -1,+1 | 1110(6) | -63,⋯,-32,+32,⋯,+63 |
| 011(2) | -3,-2,+2,+3 | 11110(7) | -127,⋯,-64,+64,⋯,+127 |
| 100(3) | -7,⋯,-4,+4,⋯,+7 | 111110(8) | -255,⋯,-128,+128,⋯,+255 |
| 101(4) | -15,⋯,-8,+8,⋯,+15 | 111110(9) | -511,⋯,-256,+256,⋯,+511 |

It has been mentioned before that images are affected by bit errors during transmission, so it is necessary to implement error correction codes during the encoding process. As shown in Figure 2.8, (7,4) Hamming code protects some key information, such as transformation from RGB to YCrCb, predictors types, first row and first column of the original image, interval Huffman coding, and standard Huffman coding parameters.

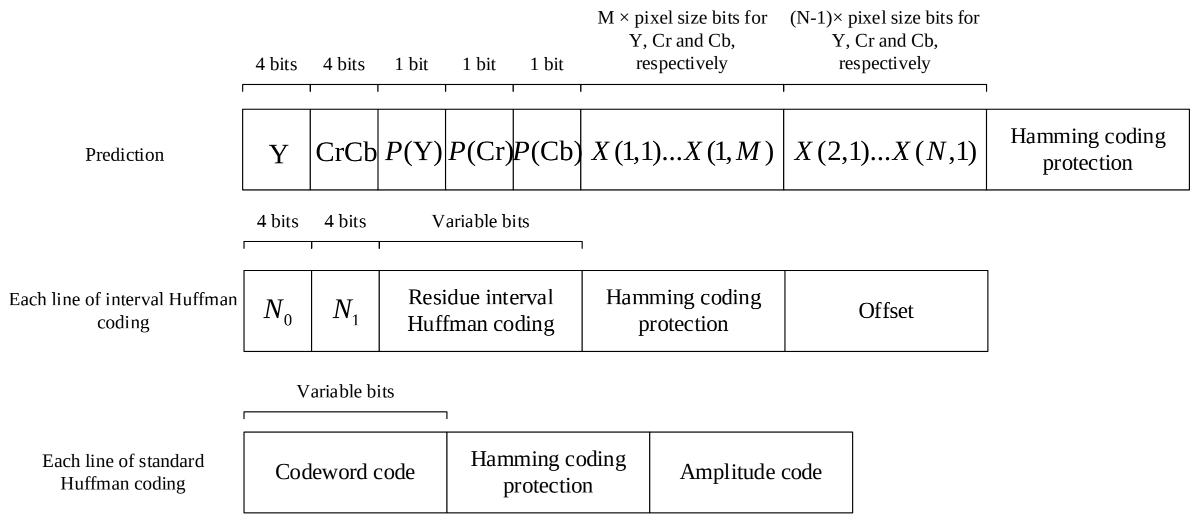

M × pixel size bits for Y, Cr and Cb, respectively
(N-1)× pixel size bits for Y, Cr and Cb, respectively
4 bits
4 bits
1 bit
1 bit
1 bit
Prediction
Y
CrCb
$P$(Y)
$P$(Cr)
$P$(Cb)
$X(1,1)...X(1,M)$
$X(2,1)...X(N,1)$
Hamming coding protection
4 bits
4 bits
Variable bits
Each line of interval Huffman coding
$N_0$
$N_1$
Residue interval Huffman coding
Hamming coding protection
Offset
Variable bits
Each line of standard Huffman coding
Codeword code
Hamming coding protection
Amplitude code

Figure 2.8 Coding protection for Huffman coding.

## 2.5 Error Control Coding

When a digital signal is transmitted on an actual channel, the received digital signal inevitably generates bit errors due to unsatisfactory channel transmission characteristics and additive noise. In order to achieve a certain bit error rate index under the known signal-to-noise ratio, the baseband signal should be designed reasonably. Meanwhile, the modulation and demodulation methods should be selected, and the frequency domain equalization and time domain equalization should be adopted to make the bit error rate as low as possible. However, if the bit error rate still fails to meet the requirements, channel coding, that is, error control coding, must be adopted to reduce the bit error rate to match the index requirements. With the improvement of error control coding theory and the development of digital circuit technology, channel coding has been successfully applied in various communication systems, and it has also been widely used in computers, magnetic recording and storage.

The primary method of error control coding is to add some supervised symbols to the information sequence transmitted by the transmitting end, and these redundant symbols and information symbols are associated (constrained) with specific certain rules. The receiving end checks the relationship between the information symbols and the supervised symbols according to the established rules. Once an error occurs during the transmission, the relationship between the information symbol and the supervised symbol is destroyed, so that the error can be found and corrected.

There are three commonly used error control methods: Automatic repeat request (ARQ), forward error correction (FEC) and hybrid error correction (HEC). In the ARQ mode, the transmitting end sends a code that can detect the error after being encoded, and the receiving end receives the code. If it finds that there is an error during the transmission, the judgment result is fed back to the transmitting end through the reverse channel. Then, the sender retransmits the previously sent information once until the receiver believes that the message has been received correctly.

In the FEC system, the transmitting end is coded to issue a code that can correct the error, and after receiving the code group, the receiving end can automatically find and correct the error during the transmission through decoding. The FEC mode does not require a feedback channel. It is particularly suitable for applications where only a unidirectional channel can be provided. Since it can automatically correct errors and does not check for retransmissions, the time delay is small and the real-time performance is good. In order to obtain a low bit error rate after error correction, the error correction code should have strong error correction capability. However, the stronger the error correction capability, the more complicated the decoding device is. The main disadvantage of the FEC system is that the equipment is complicated.

The HEC method is a combination of the FEC method and the ARQ method. In this system, the sender not only can correct errors but also can detect errors that exceed the error correction capability. In the latter case, the sender is required to resend the errors through the feedback channel. The HEC method is a compromise between FEC and ARQ methods in terms of real-time performance and decoding complexity.

In this paper, we adopt the FEC system to implement bit error correction. The problem with the FEC system mentioned is that the device is more complicated. Hamming code uses the parity block mechanism to reduce the cost of this system. Hamming code (7.4) is used to detect and correct the single bit error.  General Hamming code can detect and correct more than one bit error in the code word, in which more parity bits are included. Table 2.9 shows the relationship between code length and number of parity bits. According to Table 2.9, we decide to protect the key information by using a (7, 4) Hamming code that encodes the four bits of the data into seven bits by adding three parity bits.

Table 2.9 Relationship between code length and number of parity bits

| n | k |
|---|---|
| 1 | 2 |
| 2～4 | 3 |
| 5～11 | 4 |
| 12～26 | 5 |

# CHAPTER 3. INTELLIGENT OPTIMIZATION ALGORITHM

Today science and technology are in an era of multidisciplinary cross-infiltration. In this era, the development of computer technology has greatly improved the living standards of the people. At the same time, with the development of human knowledge, people put forward higher requirements for science and technology. Therefore, efficient optimization techniques and intelligent computing are valued by more and more people.

Optimization technology is an application technology based on mathematics to solve the optimal solution of various engineering problems. As an important branch of science and technology, it is widely used in artificial intelligence, pattern recognition, computer engineering and other fields.

There are many optimization algorithms, including classical optimization algorithms, improved local search algorithms, guided search methods and system evolution methods. Classical algorithms include linear programming, dynamic programming, etc.; improved local search algorithms include hill climbing, steepest descent, and so on. Simulated annealing, genetic algorithms, and tabu search are called instructional search methods. Neural network and chaotic search belong to the dynamic evolution method of the system.

Gradient-based traditional optimization algorithms have the advantages of high computational efficiency, strong reliability, and mature technology. They are the most important and widely used optimization algorithms. However, traditional optimization methods have significant limitations when applied to complex and difficult optimization problems. An optimization problem is complex and usually refers to one of the following characteristics:

(1) The objective function does not have a clear analytical expression.

(2) The objective function is clearly expressed, but it is impossible to accurately evaluate it.

(3) The objective function is a multimodal function.

(4) There are multiple objective functions, that is, multi-objective optimization.

An optimization problem is difficult. It usually means that the objective function or constraint is discontinuous, non-differentiable, highly nonlinear, or the problem itself is a difficult combination problem. Traditional optimization methods often require that the objective function be convex, continuously differentiable, and the feasible domain is a convex set. In addition, these methods have a poor ability to process non-deterministic information. These weaknesses make traditional optimization methods limited when solving many practical problems.

Intelligent optimization algorithms are generally random search algorithms based on bio-intelligence or physical phenomena. At present, they are far less theoretically perfect than traditional optimization algorithms, and often cannot guarantee the optimality of solutions. Therefore, they are often regarded as “Meta-heuristic”. In computer science and mathematical optimization, a metaheuristic is designed to find, generate, or select a heuristic (partial search algorithm) that can provide a sufficiently good solution to an optimization problem [33]. However, from the point of view of practical application, such new algorithms generally do not require the continuity and convexity of the objective function and constraints, and sometimes they do not need analytical expressions. Besides, these new algorithms also have a strong adaptation to the uncertainty of the data in the calculation. Therefore, intelligent optimization algorithms are applied to solve some complex practical problems.

The intelligent optimization algorithm is generally to solve the optimization problem. The optimization problem can be divided into two problems:

(1) A function optimization problem for solving the value of an independent variable with the smallest value of a function.
(2) A combinatorial optimization problem that finds the optimal solution in a solution space and minimizes the value of the objective function.

In this thesis work, we adopt two intelligent optimization methods, which are genetic algorithm and particle swarm optimization algorithm to select the best combination of color space conversions and predictors in the experiment. There are 864 formula combinations of color space and predictors. Searching with intelligent algorithms takes one third less time than searching with normal exhaustion.

## 3.1 Genetic Algorithm

Genetic Algorithm (GA), which originated from computer simulation research on biological systems, is a random search method that draws on the natural selection and natural genetic mechanism of the biological world [34]. In the early 1960s, American professor Holland proposed to use the basic principles of genetics to simulate the natural evolution of organisms when designing artificial adaptive systems. In 1975, Holland published the first monograph on the basic theory and method of genetic algorithm. The monograph proposed the most important schema theory in the research and development of genetic algorithm. Therefore, it is generally believed that 1975 is the birth year of genetic algorithm. From the perspective of the entire development process, genetic algorithm emerged in the 1970s, developed in the 1980s, and entered the climax in the 1990s.

Genetic algorithm is a random global search and optimization method that mimics the evolutionary mechanism of biological evolution in nature. It draws on the great evolution of Darwin and Mendel’s genetic theory. Using the principle of “survival of the fittest”, an approximately optimal solution is generated successively in the potential solution population. When solving problems, genetic algorithm has the following advantages:

(1) The genetic algorithm can simultaneously process multiple individuals in a group, that is, simultaneously evaluate multiple solutions in the search space. It has excellent global search performance and is easy to parallelize.

(2) No auxiliary information is required. Genetic individuals are evaluated only by fitness functions, and genetic manipulations are performed on this basis.

(3) Instead of using deterministic rules, genetic algorithms use probabilistic transition rules to guide the search direction.

(4) Even in the case where the defined fitness function is discontinuous or irregular, the genetic algorithm is not easy to fall into local optimum during the search process.

(5) Natural evolutionary mechanisms are used to express complex phenomena. It is able to solve very difficult problems quickly and accurately.

(6) It has inherent parallelism and parallel computing capacity.

(7) It is easy to mix with other technologies.

Of course, the genetic algorithm also has some drawbacks comparing with other optimization technology:

(1) The programming implementation of genetic algorithm is more complicated.

(2) A single genetic algorithm encoding cannot fully represent the constraints of the optimization problem.

(3) Efficiency is low.

(4) It is prone to premature convergence.

(5) There is no effective quantitative analysis method for the accuracy, credibility and computational complexity of the algorithm.

In general, genetic algorithm is a good way to solve the optimal solution problem in this experiment. Figure 3.1 shows the flow chart of genetic algorithm.

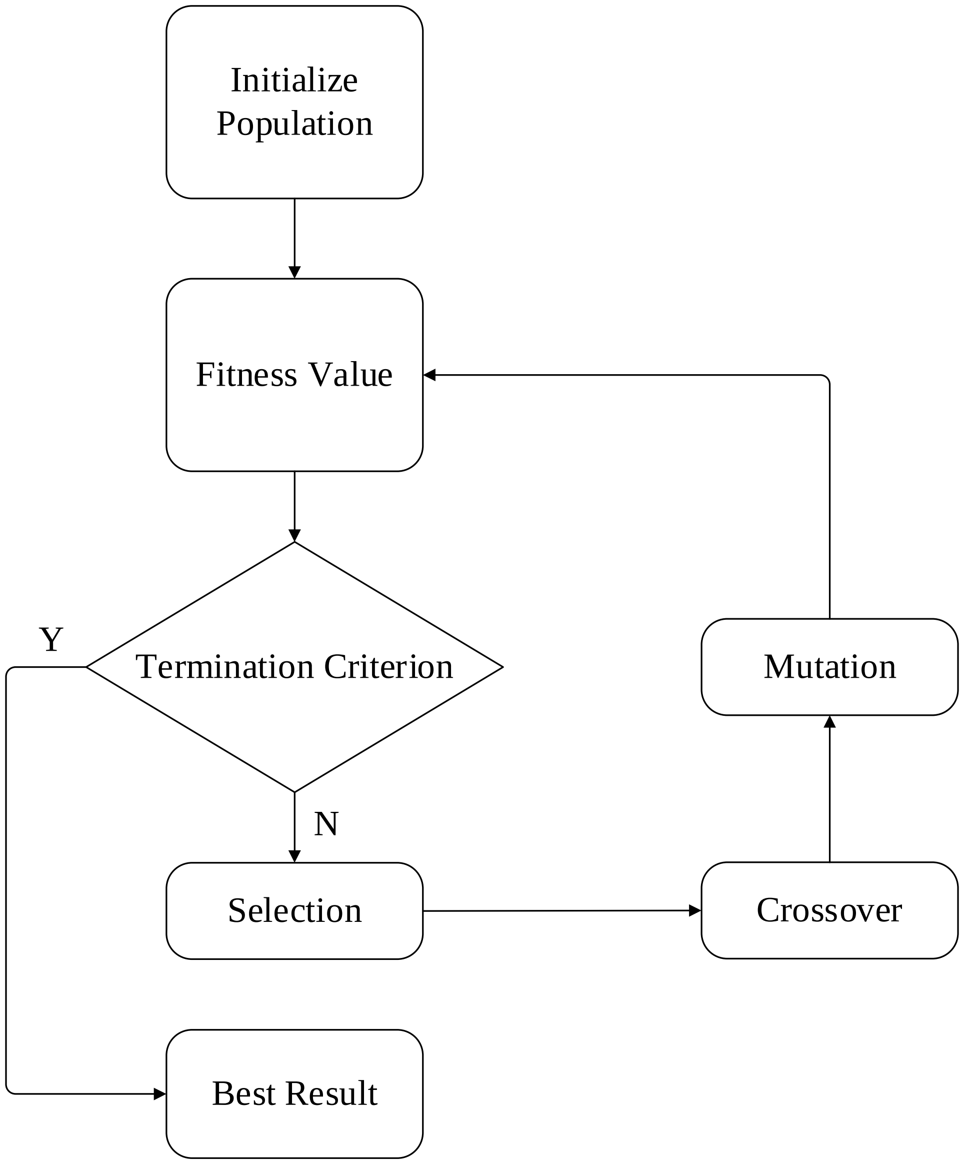


Figure 3.1 Flow chart of genetic algorithm.

As shown in Figure 3.1, when we want to use the genetic algorithm to find the optimal solution of a problem, an initial population group is given firstly. Next, the fitness value of each individual in the population is calculated, and the optimal fitness value is worked out. Then, we determine whether the optimal solution satisfies the judgment index. Since the genetic algorithm is a random approximation algorithm, we must take measures to not converge to the local optimal solution but the global optimal solution, and try to improve the probability of reaching the optimal solution. Therefore, in addition to designing the fitness function, the genetic algorithm has three important parts: selection, crossover, and mutation. New generation of population is selected through these three steps. Then, we repeat the previous experimental steps. Finally, the best result can be found. Four significant parts of the genetic algorithm are fitness function, selection, crossover and mutation. We will focus on these four parts in the next couple sections.

### 3.1.1 Fitness function

Fitness function, also called evaluation function, is a criterion for distinguishing the quality of individuals in a group based on the objective function. High fitness values, that is, excellent individuals have a higher chance of participating in reproduction, inheriting their genes.

Fitness consists of five variables in Figure 3.2. The first two variables are the transformation information for converting RGB image to YCrCb image. They have 9 and 12 cases respectively, so 4 bits are used for encoding each variable. Each of the last three variables has 2 cases for prediction of Y, Cr or Cb, which is encoded by 1 bit.

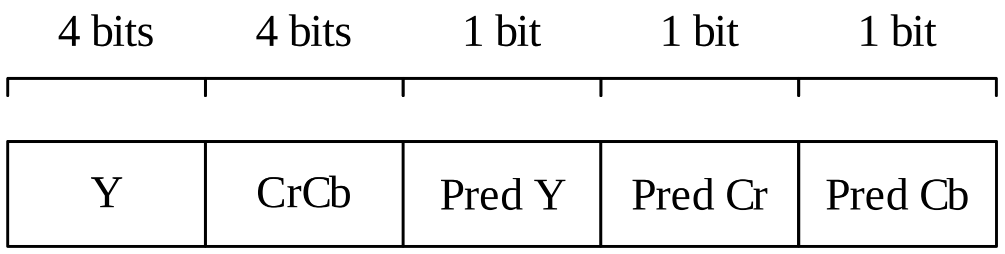


Figure 3.2 Five variables of the fitness function.

Fitness value is defined as the information entropy. The formula for calculating information entropy is expressed as

$$\begin{aligned} entropy = &-\sum p_Y(i)\log_2 p_Y(i) \\ &-\sum p_{Cr}(i)\log_2 p_{Cr}(i), \\ &-\sum p_{Cb}(i)\log_2 p_{Cb}(i) \end{aligned} \tag{3.1}$$

where $p_Y(i)$ is the residue probability from the Y component predictor. $p_{Cr}(i)$ is the residue probability from the Cr component predictor. $p_{Cb}(i)$ is the residue probability from the Cb component predictor.

### 3.1.2 Selection

The selection operation in the genetic algorithm is to select some individuals from the parent population to inherit into the next generation group. There are some common selection operators.

1. Roulette Wheel Selection: It is a playback random sampling method. The probability that each individual enters the next generation is equal to the ratio of its fitness value to the sum of individual fitness values in the population. The selection error is large.
2. Excepted Value Selection: It is a random selection operation based on the survival expectation of each individual in the next generation group.
3. Uniform Sort Selection: All individuals in the group are ranked according to their fitness values, and the probability that each individual is selected is assigned based on this ranking rule.
4. Tournament Selection: K random individuals are selected in the population, where K is called the size of the tournament. Then, the best individual enters the next generation. Figure 3.3 shows an example of tournament selection.

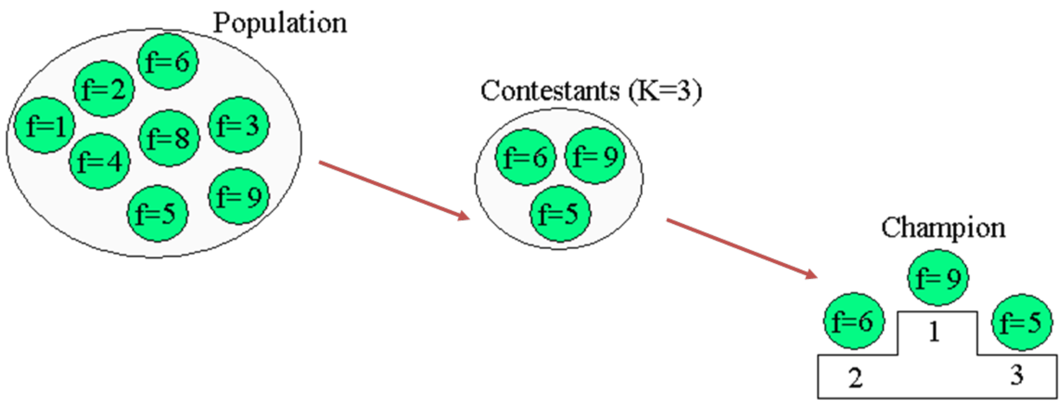


Figure 3.3 An example of tournament selection.

### 3.1.3 Crossover

Crossover refers to the operation of replacing the partial structure of two parent individuals to generate new individuals. The following lists some common crossover methods showed in Figure 3.4:

1. Single Point Crossover: It means that only one intersection point is randomly set in the individual code string, and then two parts of the two paired individuals are exchanged at the point.
2. Two Point Crossover: Two intersections are randomly set in the individual code string, and then some genes are exchanged.
3. Uniform Crossover: The genes on both individuals are exchanged with the same crossover probability to form two new individuals.
4. Arithmetic Crossover: Two new individuals are produced by a linear combination of two individuals. The operand is typically an individual represented by the floating-point number.

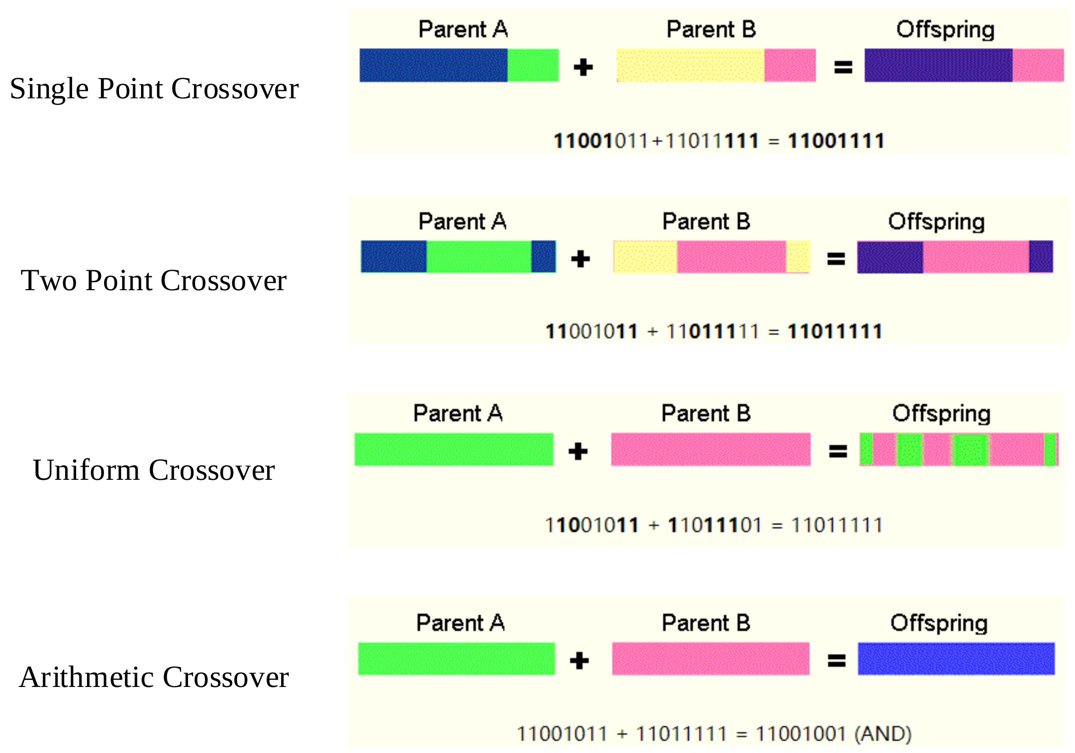


Figure 3.4 Four crossover methods.

### 3.1.4 Mutation

The mutation operation in the genetic algorithm refers to replacing some values in the individual code strings with other values to form a new individual. Figure 3.5 lists four mutation methods, and these methods are shown below:

1. Bit Flip Mutation: It is to change the value of a bit randomly specified by the mutation probability in the individual binary code string.
2. Swap Mutation: This method randomly changes the value of two genes in an individual coding string.
3. Scramble Mutation: Scramble mutation randomly selects a region of the individual to disrupt the genes in it.
4. Inversion Mutation: It randomly selects a segment of the individual to reverse the values in the region.

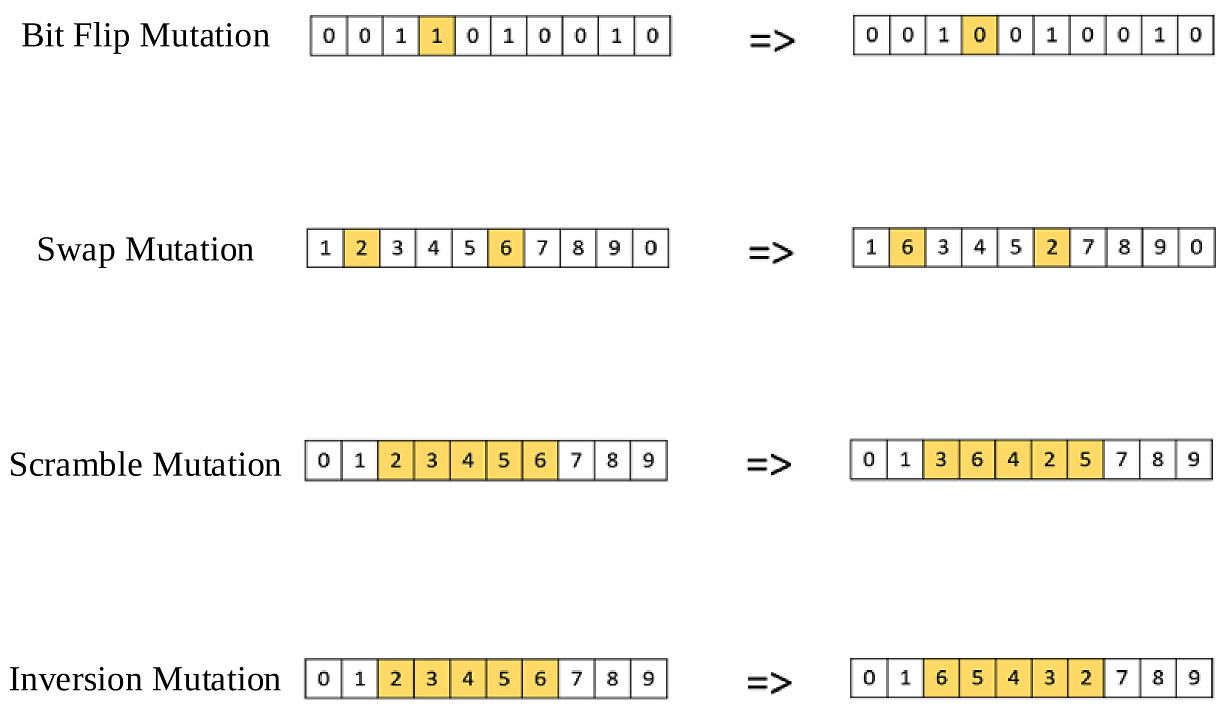


Figure 3.5 Four mutation methods.

In the experiment, tournament selection, single point crossover, and bit flip mutation are used to solve the optimal solution.

## 3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) was first proposed by Eberhart and Kennedy in 1995. It is a random search algorithm based on group collaboration developed by simulating bird foraging behavior [35], [36]. It is generally considered to be a type of Swarm intelligence (SI), and it can be incorporated into the Multiagent optimization system (MAOS).

In this algorithm, each particle in the particle swarm represents a possible solution to a problem. Through the simple behavior of the individual particles, the information interaction within the group realizes the intelligence of the problem-solving. Due to its simple operation and fast convergence, PSO has been widely used in many fields, such as function optimization, image processing, and geodetic survey. With the expansion of the application scope, the PSO algorithm also has some shortcomings. This algorithm does not handle discrete optimization problems well, and it is easy to fall into local optimum. According to its characteristics, there are several development directions as follows:

(1) The global detection and local search capabilities of the algorithm are balanced by adjusting the parameters of the PSO. For example, Shi and Eberhart introduce inertia weights for the velocity term of the PSO algorithm, and they make linear (or nonlinear) dynamic adjustments to the inertia weights according to the iterative process and particle flight conditions, so that balancing the globality and convergence speed of the search.
(2) Different types of topologies are designed, and particle learning patterns are changed to increase population diversity
(3) PSO algorithm is combined with other optimization algorithms (or strategies) to form a hybrid PSO algorithm.
(4) Niche technology is adopted. Niche is a bionic technique that simulates ecological balance and is suitable for the optimization of multimodal functions and multi-objective functions.

Different development directions represent different application areas. Some need to perform global detection continuously; some need to improve the precision of optimization; some require the balance between global search and local search; others need to solve high-dimensional problems. We should choose the most appropriate algorithm for solving different problems in different fields.

The basic concept of the PSO algorithm mentioned before stems from the study of the foraging behavior of birds. In the PSO algorithm, an optimization problem is regarded as a group of birds that feed in the air. Food is regarded as the optimal solution to the optimization problem, and each foraging bird flying in the air is a particle that the PSO algorithm searches in the solution space. Each particle can be regarded as a search individual in the N-dimensional search space. The current position of the particle is a candidate solution to the optimization problem, and the flight process of the particle is the individual's search process. The flight speed of particles can be dynamically adjusted according to the optimal position of the particle history and the optimal position of the population history.

### 3.2.1 Position and velocity

Particles have only two properties: velocity and position, where velocity represents the speed of movement, and position represents the direction of movement. Besides, the optimal solution for each particle to search separately is called the individual extremum, and the optimal individual extremum in the particle swarm is taken as the current global optimal solution. By updating the velocity and position of the particles, an optimal solution that satisfies the termination condition is finally obtained.

As shown in Figure 3.6, when we use the PSO algorithm to search for the optimal solution, the first step is to initialize the particles with random position and velocity vectors. In the second step, the fitness value of each particle should be calculated. The third step updates the best solution for each particle ($pbest$) and the optimal solution for all particles in the entire particle group ($gbest$). Next, the position and velocity of each particle are updated. Then, we determine whether the optimal solution satisfies the judgment index. If the system satisfies the decision condition, it outputs the optimal solution. If not, the algorithm returns to the previous second step.

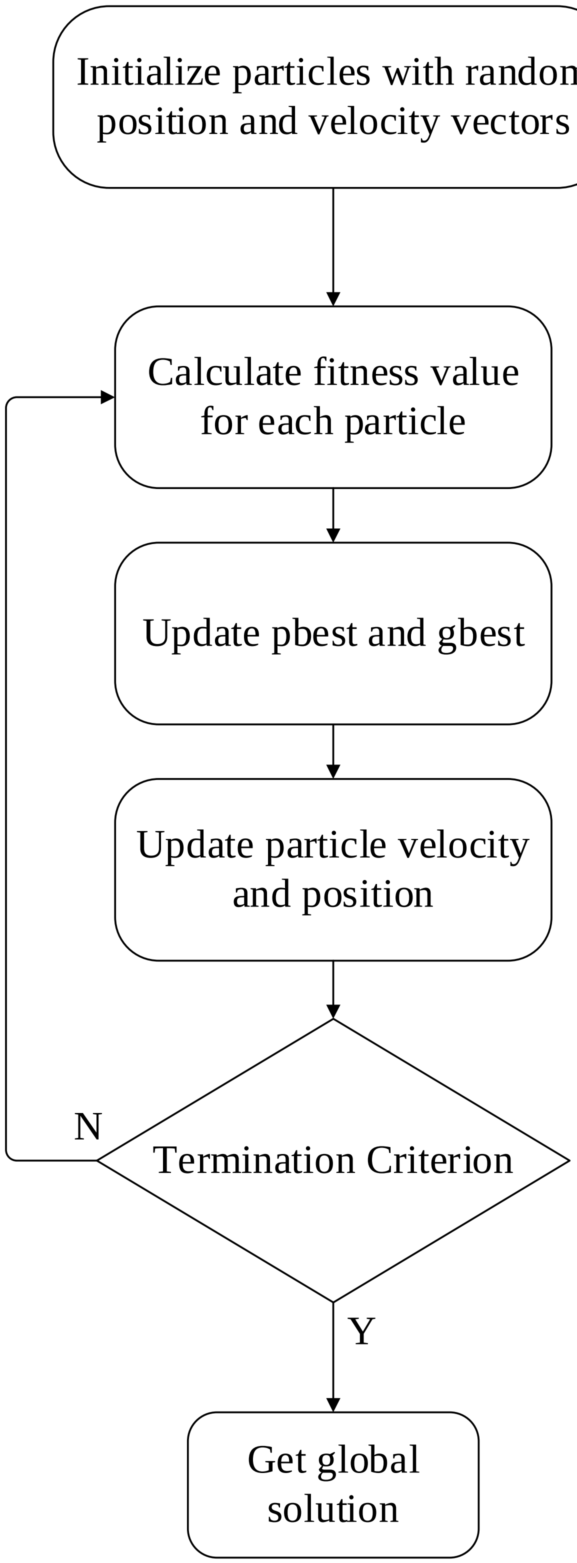

Initialize particles with random position and velocity vectors
Calculate fitness value for each particle
Update pbest and gbest
Update particle velocity and position
Termination Criterion
N
Y
Get global solution

Figure 3.6 Flow chart of particle swarm optimization.

### 3.2.2 Fitness value

The fitness value in the PSO algorithm is the same as the one in the GA algorithm. As shown in Figure 3.2, the fitness contains five variables. The first two variables are conversion information for converting an RGB image into a YCrCb image. They have 9 and 12 cases respectively, so each variable is encoded using 4 bits. Each of the last three variables has two cases for prediction of Y, Cr or Cb, which is encoded by 1 bit. In addition, Equation (3.1) calculates the fitness value.

### 3.2.3 Update rule

PSO algorithm first initializes a group of random particles (random solutions). Then, we can find the optimal solution by iteration. In each iteration, the particle updates itself by tracking two "extreme values" ( $pbest$ , $gbest$ ). After finding the two optimal values, the particles update their velocity and position by the following equations:

$$v(i) = \omega \times v(i) + c_1 \times r_1 \times (pbest(i) - x(i)) + c_2 \times r_2 \times (gbest(i) - x(i)), \tag{3.2}$$

$$x(i) = x(i) + v(i), \tag{3.3}$$

where $\omega$ is the inertia factor, which is non-negative. When the factor's value is large, the global optimization ability is strong, and the local optimization ability is weak; when the value is small, the algorithm has poor global optimization ability and high local optimization ability. $c_1$ and $c_2$, are accelerating constants, where $c_1$ is the individual learning factor of each particle, and $c_2$ is the social learning factor of each particle. We usually set $c_1 = c_2 = 2$, but $c_1$ and $c_2$ do not have to be equal to 2. $r_1$ and $r_2$ are random numbers from 0 to 1. $pbest$ is the fitness value of the best position every individual has experienced. $gbest$ is the global optimal fitness value.

## 3.3 Similarities and Differences

Since both the PSO algorithm and the GA algorithm belong to the intelligent optimization algorithm, they have a lot of commonalities:

(1) Both of them belong to the bionic algorithm, the random search algorithm, and the global optimization algorithm.

(2) They imply parallelism.

(3) These two algorithms search based on individual adaptation information, so they are not limited by functional constraints such as continuity, derivative, and so on.

(4) For high-dimensional complex problems, these two algorithms often encounter the disadvantages of premature convergence and poor convergence performance, so they can't guarantee convergence to the best solution.

At the same time, the PSO algorithm and GA algorithm also have some differences:

(1) The PSO algorithm applies the solution to the formation of a new population in each iteration, while the GA algorithm does not save the solution to the next population.

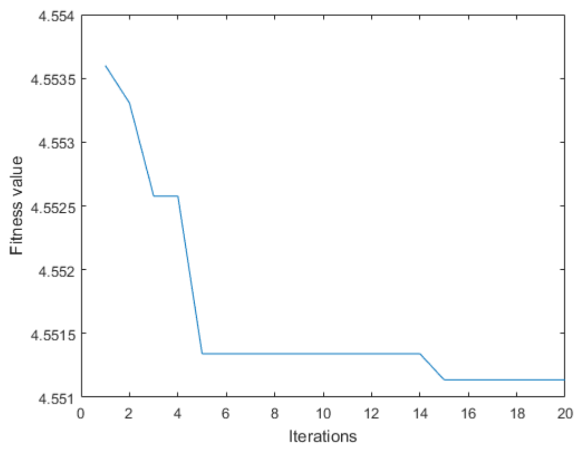
(2) In the GA algorithm, chromosomes share information with each other, so the entire population moves more evenly toward the optimal region. PSO is a single item information sharing mechanism, and the whole search update process is the process of following the current optimal solution. In most cases, all particles can converge to the optimal solution faster than the evolutionary individuals in the genetic algorithm.

(3) GA's coding technology and genetic operation are relatively simple, while PSO does not require crossover and mutation operation relative to GA. Particles are only updated by internal velocity. Therefore, the principle of PSO is simpler, and it is easier to implement.

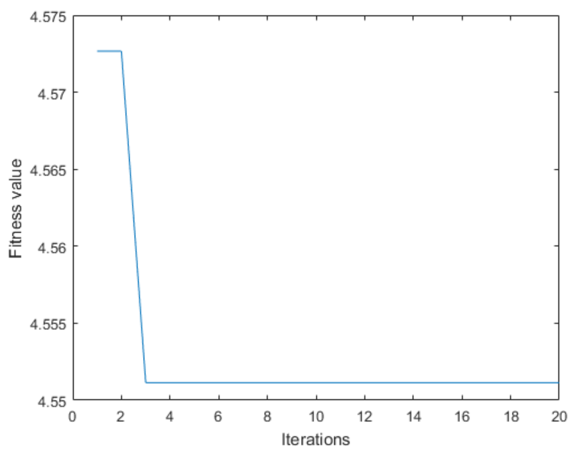
## 3.4 Performance Evaluations

In order to compare the performance of the GA algorithm and the PSO algorithm, we use these two methods to process four different 512×512 color images, which are named as "Lena", "Baboon", "Pepper", and "Airplane", and a 256×256 "House" image. These images are standard images for image processing.

Figure 3.7 shows the plots of fitness value versus the number of iterations using the genetic algorithm and particle swarm optimization for "Lena" image. It takes about 6.5 minutes and 6 minutes to process these two images on Lenovo Y900 (i7-6700k). The best fitness value is 4.5511. According to the five variables of the fitness value, the eighth formula of Y space and the first formula of CrCb space in Table 2.1 are selected, and the second kind of linear predictor for Y and the first kind of linear predictors for Cr and Cb are chosen.



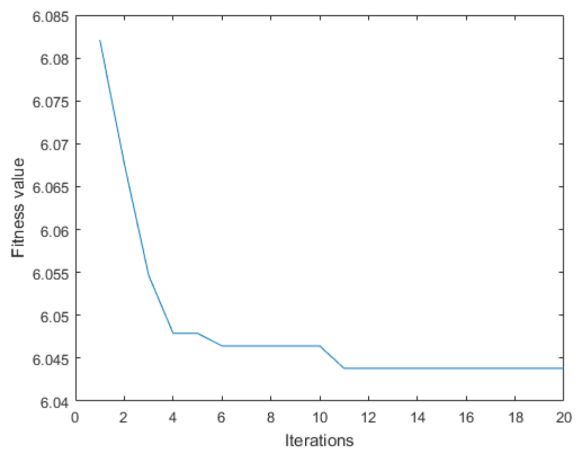
(a) Genetic Algorithm



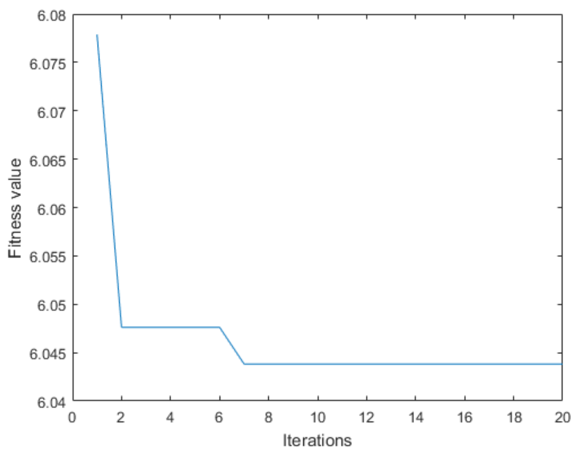
(b) Particle Swarm Optimization

Figure 3.7 Iterative curves for "Lena" image.

As shown in Figure 3.8, we obtained the plots of fitness value versus the number of iterations using both genetic algorithm and particle swarm optimization for "Baboon" image. It takes about 10 minutes to process these two images, respectively. The best fitness value is 6.0438. The sixth formula of Y space and the eleventh formula of CrCb space in Table 2.1 are selected, and the first linear predictors for Y, Cr and Cb are chosen.



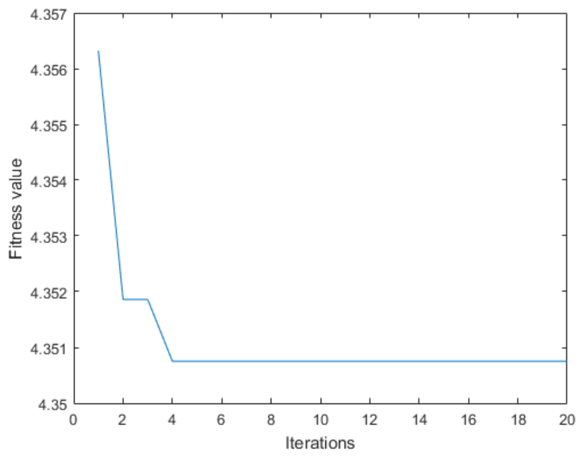
(a) Genetic Algorithm



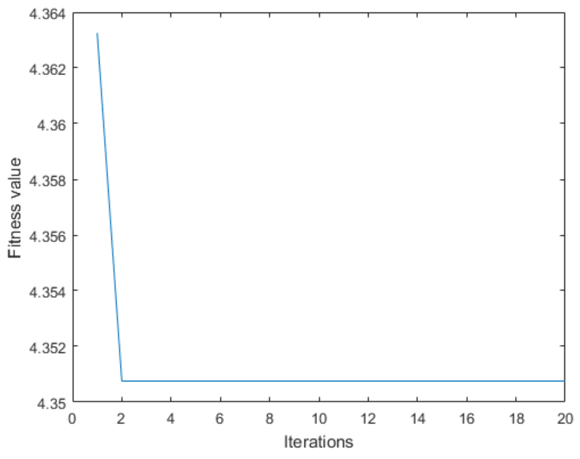
(b) Particle Swarm Optimization

Figure 3.8 Iterative curves for "Baboon" image.

It takes about 1.5 minutes to process these two images in Figure 3.9, respectively. The best fitness value is 4.3508. The fourth formula of Y space and the seventh formula of CrCb space in Table 2.1 are selected, and the second linear predictors for Y and Cr and the first predictor for Cb are adopted.



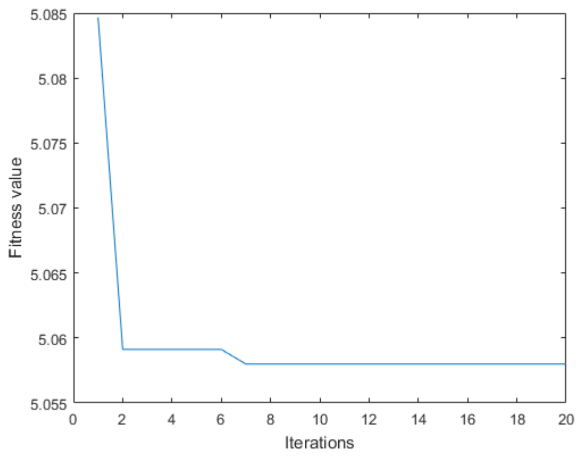
(a) Genetic Algorithm



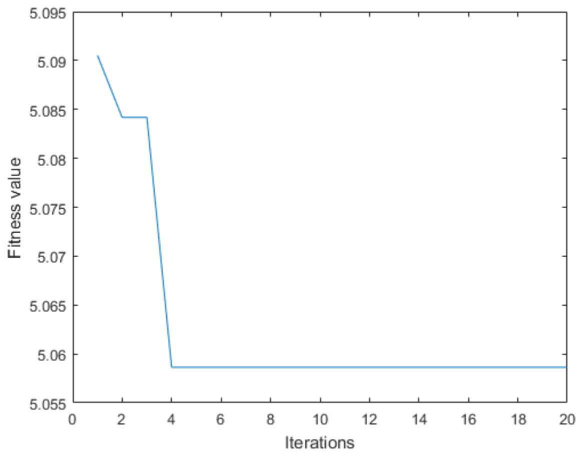
(b) Particle Swarm Optimization

Figure 3.9 Iterative curves for "House" image.

As shown in Figure 3.10, it takes about 9 minutes to process these two images on Lenovo Y900 (i7-6700k), respectively. The best fitness value is 5.05799. We choose the ninth formula of Y space and the twelfth formula of CrCb space in Table 2.1, and the first kind of linear predictors for Y, Cr and Cb.



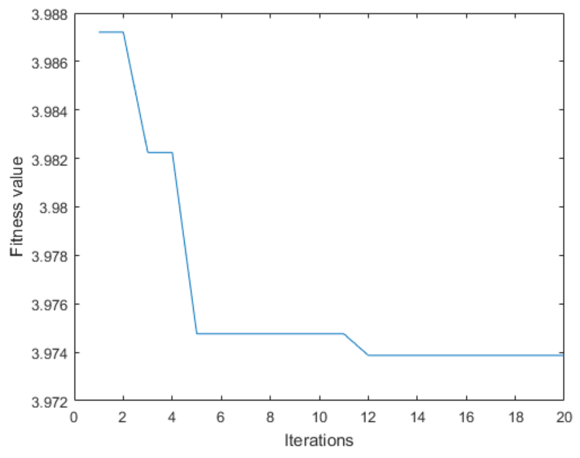
(a) Genetic Algorithm



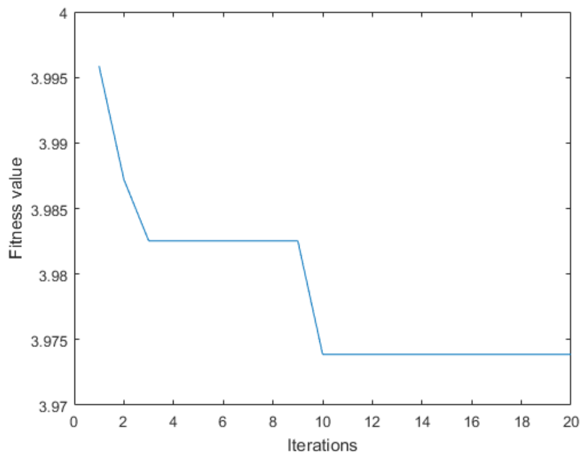
(b) Particle Swarm Optimization

Figure 3.10 Iterative curves for “Pepper” image.

Figure 3.11 shows the plots of fitness value versus the number of iterations using the genetic algorithm and particle swarm optimization for “Airplane” image. It takes about 8 minutes to process these two images, respectively. The best fitness value is 3.97387. The sixth formula of Y space and the seventh formula of CrCb space in Table 2.1 are chosen, and the second linear predictor for Y and the first linear predictors for Cr and Cb are chosen.



(a) Genetic Algorithm



(b) Particle Swarm Optimization

Figure 3.11 Iterative curves for “Airplane” image.

Combining Figure 3.7 to Figure 3.11, both PSO and GA algorithms can achieve the optimal solution that we expect. In addition, the PSO algorithm converges to the smallest fitness value and faster than the GA algorithm. As summary from our experiments, the PSO algorithm is slightly better than the GA algorithm.

# CHAPTER 4. EXPERIMENTS AND RESULTS

This chapter presents all color image compression results using four different residue coding methods. In our experiments, we adopt four different 512×512 color images, which are named as “Lena”, “Baboon”, “Pepper”, and “Airplane”, and a 256×256 “House” image. These images are standard images for image processing. The peak signal to noise ratio (PSNR) and compression ratio (CR) are used for encoding performance evaluations.

The peak signal to noise ratio is an engineering term that represents the ratio of the maximum possible power of a signal to the destructive noise power that affects its representation accuracy. Since many signals have a vast dynamic range, PSNR is often expressed in logarithmic decibel units. In image processing, to objectively evaluate an image, it is often necessary to calculate the PSNR. PSNR is an objective measure of image distortion or noise level. The larger the PSNR value between the two images, the more similar the images are. The universal benchmark for PSNR is 30dB, and the image degradation is more obvious below 30dB. The formulas for calculating the PSNR of an M×N size RGB image are as follows:

$$PSNR(dB) = 20 \times \log_{10}\left(\frac{255}{RMSE}\right) \tag{4.1}$$

$$\begin{aligned} RMSE^2 = \frac{1}{3M \times N}\Bigg( & \sum_{i=1}^{N}\sum_{j=1}^{M}[R(i,j) - \hat{R}(i,j)]^2 \\ & + \sum_{i=1}^{N}\sum_{j=1}^{M}[G(i,j) - \hat{G}(i,j)]^2 + \sum_{i=1}^{N}\sum_{j=1}^{M}[B(i,j) - \hat{B}(i,j)]^2 \Bigg) \end{aligned} \tag{4.2}$$

where $R(i, j)$ and $\hat{R}(i, j)$, $G(i, j)$ and $\hat{G}(i, j)$, and $B(i, j)$ and $\hat{B}(i, j)$ are the original pixels and the recovered pixels, for RGB components, respectively.

Compression ratio is the ratio of the pixel size of the original image and the bits of per pixel of the compressed image. Therefore, the larger the compression ratio, the better the image compression performance is.

## 4.1 Performance Evaluation for RGB image and Color-converted Image

For an RGB color image, it is necessary to convert RGB image to YCrCb image which is generally used for digital image and video. If the image is directly compressed without color space conversion, its compression ratio and image quality will decrease. Table 4.1 to Table 4.5 list the

PSNRs and CRs of RGB images without color space processing and RGB images processed by color space using different residue coding algorithms when the bit-error rates (BERs) are 0.001 and 0.005. Here, we use the five different color images mentioned before. Each PSNR is obtained by averaging the values from 10 independent runs.

Table 4.1 Performance evaluation for RGB image and color-converted image in "Lena" image

| Algorithms | RGB image without color space processing | Color-converted image |
|---|---|---|
| 2-D bi-level block coding | PSNR: 34.3162 dB (BER=0.001)<br>PSNR: 25.8931 dB (BER=0.005)<br>CR: 1.3760 | PSNR: 37.5844 dB (BER=0.001)<br>PSNR: 28.9834 dB (BER=0.005)<br>CR: 1.4456 |
| 1-D bi-level block coding | PSNR: 35.6342 dB (BER=0.001)<br>PSNR: 26.5518 dB (BER=0.005)<br>CR: 1.3490 | PSNR: 37.9850 dB (BER=0.001)<br>PSNR: 28.0733 dB (BER=0.005)<br>CR: 1.4276 |
| Interval Huffman coding | PSNR: 35.4030 dB (BER=0.001)<br>PSNR: 22.1875 dB (BER=0.005)<br>CR: 1.1639 | PSNR: 37.1824 dB (BER=0.001)<br>PSNR: 24.0405 dB (BER=0.005)<br>CR: 1.2345 |
| Standard Huffman coding | PSNR: 33.7797 dB (BER=0.001)<br>PSNR: 20.0381 dB (BER=0.005)<br>CR: 1.0532 | PSNR: 36.1313 dB (BER=0.001)<br>PSNR: 22.5706 dB (BER=0.005)<br>CR: 1.0735 |

Table 4.2 Performance evaluation for RGB image and color-converted image in "Baboon" image

| Algorithms | RGB image without color space processing | Color-converted image |
|---|---|---|
| 2-D bi-level block coding | PSNR: 28.7418 dB (BER=0.001)<br>PSNR: 18.6025 dB (BER=0.005)<br>CR: 1.1117 | PSNR: 32.5854 dB (BER=0.001)<br>PSNR: 21.8879 dB (BER=0.005)<br>CR: 1.1647 |
| 1-D bi-level block coding | PSNR: 28.6679 dB (BER=0.001)<br>PSNR: 18.6078 dB (BER=0.005)<br>CR: 1.002 | PSNR: 32.8938 dB (BER=0.001)<br>PSNR: 21.6702 dB (BER=0.005)<br>CR: 1.1540 |
| Interval Huffman coding | PSNR: 31.8899 dB (BER=0.001)<br>PSNR: 18.1437 dB (BER=0.005)<br>CR: 0.9785 | PSNR: 35.6433 dB (BER=0.001)<br>PSNR: 22.4370 dB (BER=0.005)<br>CR: 1.0265 |
| Standard Huffman coding | PSNR: 27.7489 dB (BER=0.001)<br>PSNR: 14.7084 dB (BER=0.005)<br>CR: 0.8616 | PSNR: 31.6817 dB (BER=0.001)<br>PSNR: 19.0349 dB (BER=0.005)<br>CR: 0.8927 |

Table 4.3 Performance evaluation for RGB image and color-converted image in “House” image

| Algorithms | RGB image without color space processing | Color-converted image |
|---|---|---|
| 2-D bi-level block coding | PSNR: 38.5419 dB (BER=0.001)<br>PSNR: 29.7221 dB (BER=0.005)<br>CR: 1.4470 | PSNR: 40.3158 dB (BER=0.001)<br>PSNR: 32.8741 dB (BER=0.005)<br>CR: 1.4880 |
| 1-D bi-level block coding | PSNR: 40.0837 dB (BER=0.001)<br>PSNR: 31.8607 dB (BER=0.005)<br>CR: 1.4141 | PSNR: 41.5466 dB (BER=0.001)<br>PSNR: 34.0512 dB (BER=0.005)<br>CR: 1.4602 |
| Interval Huffman coding | PSNR: 41.6379 dB (BER=0.001)<br>PSNR: 28.7982 dB (BER=0.005)<br>CR: 1.2293 | PSNR: 46.0751 dB (BER=0.001)<br>PSNR: 34.6393 dB (BER=0.005)<br>CR: 1.2742 |
| Standard Huffman coding | PSNR: 40.8307 dB (BER=0.001)<br>PSNR: 28.6386 dB (BER=0.005)<br>CR: 1.0932 | PSNR: 44.1098 dB (BER=0.001)<br>PSNR: 31.9526 dB (BER=0.005)<br>CR: 1.0977 |

Table 4.4 Performance evaluation for RGB image and color-converted image in “Pepper” image

| Algorithms | RGB image without color space processing | Color-converted image |
|---|---|---|
| 2-D bi-level block coding | PSNR: 34.4890 dB (BER=0.001)<br>PSNR: 26.3337 dB (BER=0.005)<br>CR: 1.3115 | PSNR: 35.8982 dB (BER=0.001)<br>PSNR: 28.2509 dB (BER=0.005)<br>CR: 1.3227 |
| 1-D bi-level block coding | PSNR: 35.6128 dB (BER=0.001)<br>PSNR: 26.6219 dB (BER=0.005)<br>CR: 1.2923 | PSNR: 38.3388 dB (BER=0.001)<br>PSNR: 28.5178 dB (BER=0.005)<br>CR: 1.3104 |
| Interval Huffman coding | PSNR: 34.2800 dB (BER=0.001)<br>PSNR: 20.2287 dB (BER=0.005)<br>CR: 1.0980 | PSNR: 38.1848 dB (BER=0.001)<br>PSNR: 25.2053 dB (BER=0.005)<br>CR: 1.1294 |
| Standard Huffman coding | PSNR: 36.5374 dB (BER=0.001)<br>PSNR: 22.9097 dB (BER=0.005)<br>CR: 1.0141 | PSNR: 39.4418 dB (BER=0.001)<br>PSNR: 25.9164 dB (BER=0.005)<br>CR: 1.0098 |

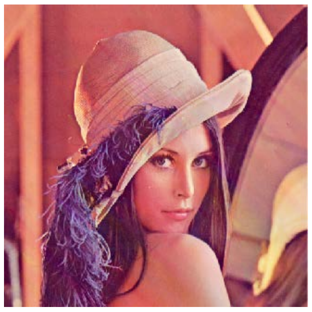
Table 4.5 Performance evaluation for RGB image and color-converted image in “Airplane” image

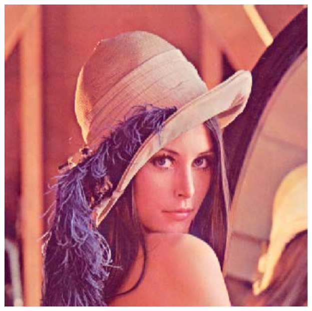
| Algorithms | RGB image without color space processing | Color-converted image |
|---|---|---|
| 2-D bi-level block coding | PSNR: 35.1219 dB (BER=0.001)<br>PSNR: 26.9715 dB (BER=0.005)<br>CR: 1.4559 | PSNR: 36.9359 dB (BER=0.001)<br>PSNR: 29.2148 dB (BER=0.005)<br>CR: 1.5104 |
| 1-D bi-level block coding | PSNR: 36.2719 dB (BER=0.001)<br>PSNR: 26.8778 dB (BER=0.005)<br>CR: 1.4349 | PSNR: 38.6429 dB (BER=0.001)<br>PSNR: 29.3021 dB (BER=0.005)<br>CR: 1.4929 |
| Interval Huffman coding | PSNR: 33.2473 dB (BER=0.001)<br>PSNR: 19.4933 dB (BER=0.005)<br>CR: 1.208 | PSNR: 37.7154 dB (BER=0.001)<br>PSNR: 23.2509 dB (BER=0.005)<br>CR: 1.2079 |
| Standard Huffman coding | PSNR: 35.3079 dB (BER=0.001)<br>PSNR: 22.9221 dB (BER=0.005)<br>CR: 1.1359 | PSNR: 38.6629 dB (BER=0.001)<br>PSNR: 25.8670 dB (BER=0.005)<br>CR: 1.1667 |

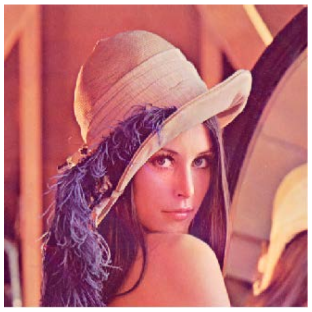
As shown in Table 4.1 to Table 4.5, the CRs and PSNRs of images that have not been processed by color space are smaller than those of images from converting RGB to YCrCb. In addition, for these residue coding methods, the PSNRs of these four methods are almost the same when the BER is 0.001. Bi-level block coding algorithm has a higher PSNR when the BER is 0.005. Even though the PSNR of the 1-D bi-level block coding method is almost the same with the PSNR of the 2-D bi-level block coding method, the 2-D bi-level block coding method has the highest CR.

As can be seen in Figure 4.1 and Figure 4.3, the recovered image is almost the same with the original image for RGB image without color space processing and color-converted image when the BER is 0.001. The images from using 2-D and 1-D bi-level block coding algorithms are identical, we display one of them in Figure 4.1, Figure 4.3, Figure 4.5, Figure 4.7 and Figure 4.9.

Original RGB image “Lena”

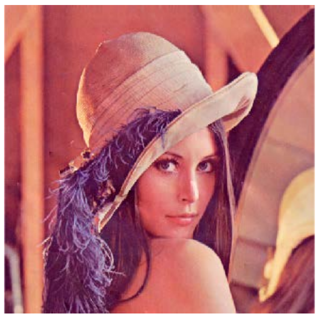

Bi-level block coding

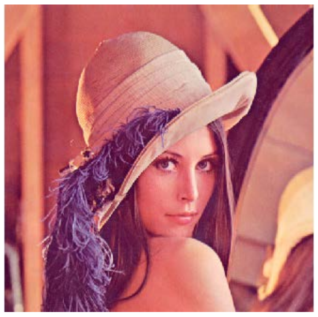

Interval Huffman coding

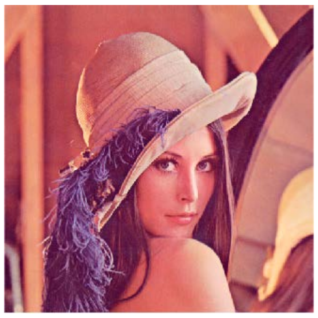

Standard Huffman coding

(a) RGB image without color space processing,

Original color-converted image
“Lena”

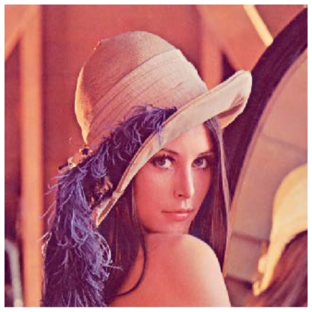

Bi-level block coding

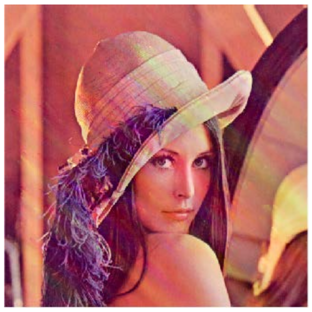

Interval Huffman coding

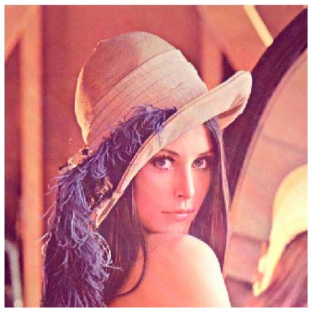

Standard Huffman coding

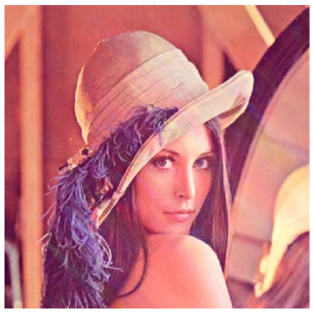
(b) Color-converted image.

Figure 4.1 Comparison results using predictive bi-level block coding, predictive interval Huffman coding and predictive Huffman coding at BER = 0.001 for “Lena” image.

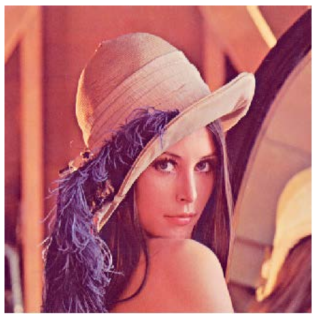

Original RGB image “Lena”

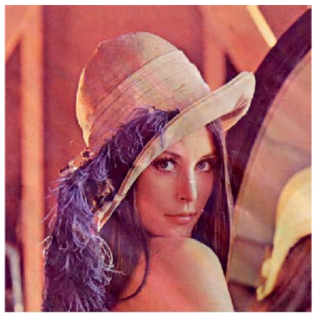

Bi-level block coding

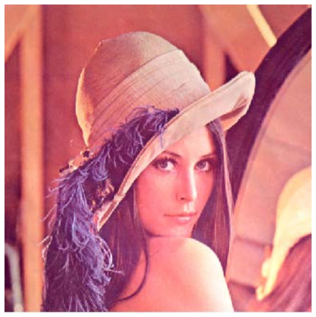

Interval Huffman coding

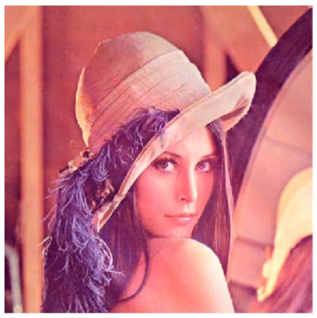

Standard Huffman coding

(a) RGB image without color space processing,


Original color-converted image
“Lena”

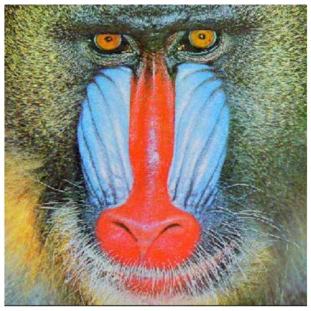

Bi-level block coding


Interval Huffman coding

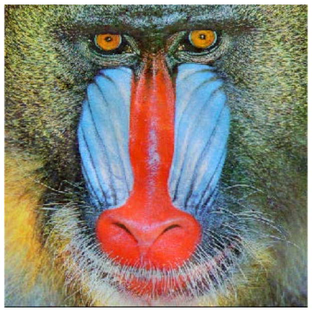

Standard Huffman coding

(b) Color-converted image.

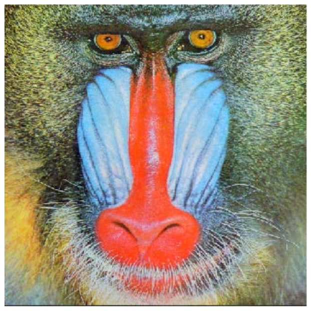
Figure 4.2 Comparison results using predictive bi-level block coding, predictive interval Huffman coding and predictive Huffman coding at BER = 0.005 for “Lena” image.



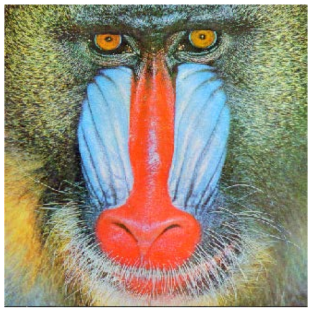
Original RGB image “Baboon”



Bi-level block coding



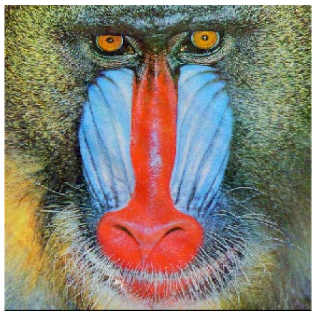
Interval Huffman coding



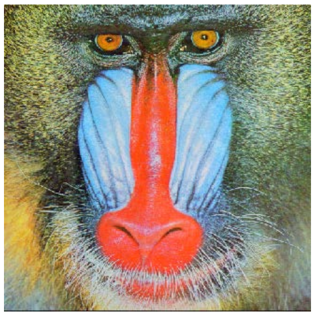
Standard Huffman coding

(a) RGB image without color space processing,

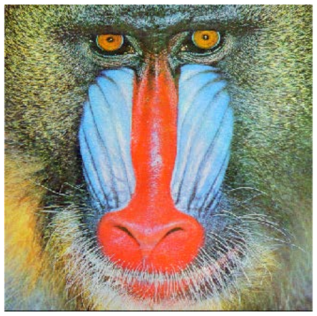
Original color-converted image
“Baboon”



Bi-level block coding



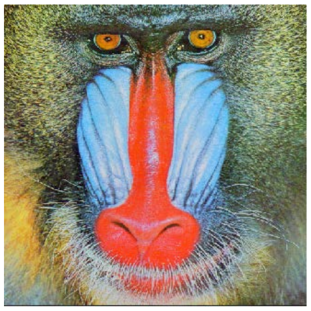
Interval Huffman coding

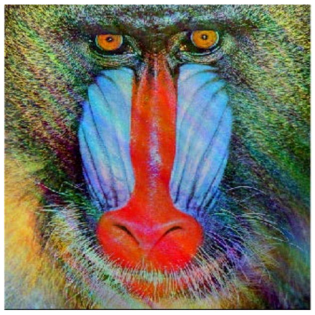


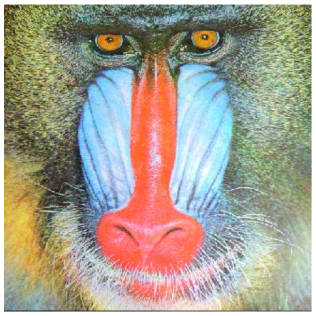
Standard Huffman coding

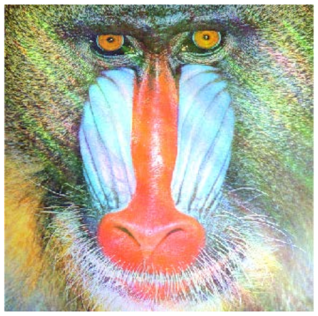
(b) Color-converted image.

Figure 4.3 Comparison results using predictive bi-level block coding, predictive interval Huffman coding and predictive Huffman coding at BER = 0.001 for “Baboon” image.

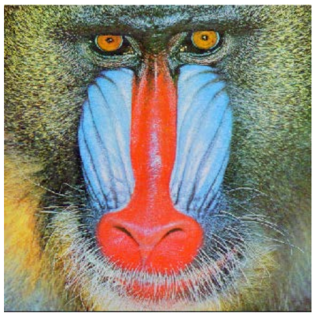

Original RGB image “Baboon”

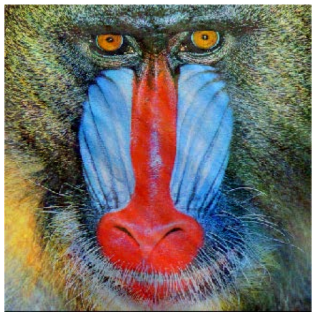

Bi-level block coding

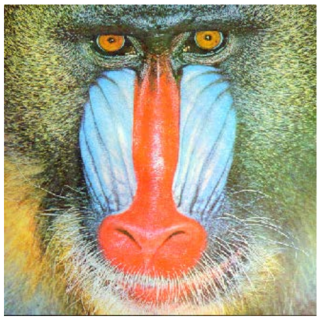

Interval Huffman coding

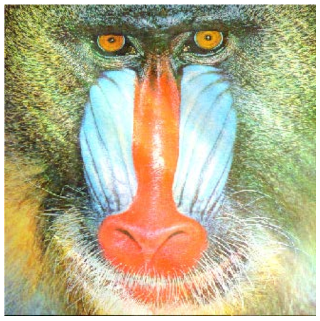

Standard Huffman coding

(a) RGB image without color space processing,


Original color-converted image “Baboon”


Bi-level block coding

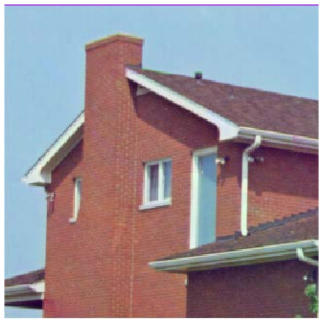

Interval Huffman coding

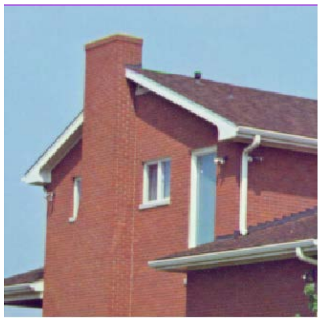

Standard Huffman coding

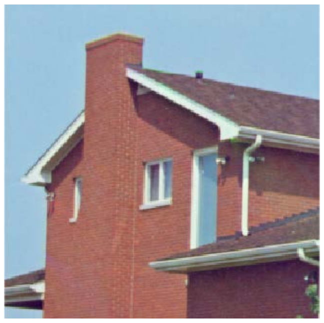
(b) Color-converted image.

Figure 4.4 Comparison results using predictive bi-level block coding, predictive interval Huffman coding and predictive Huffman coding at BER = 0.005 for “Baboon” image.

Original RGB image “House”

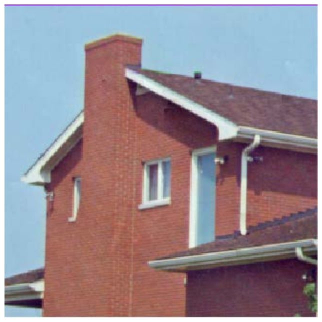

Bi-level block coding

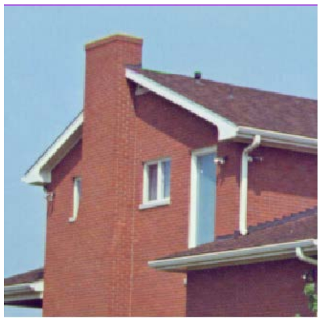

Interval Huffman coding

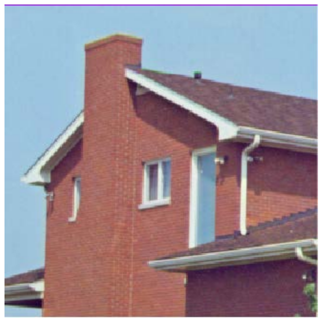

Standard Huffman coding

(a) RGB image without color space processing,

Original color-converted image
“House”

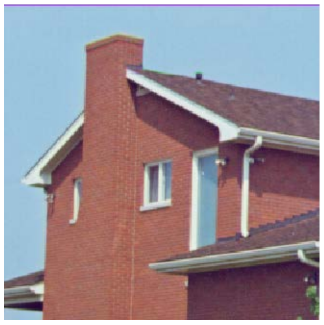

Bi-level block coding

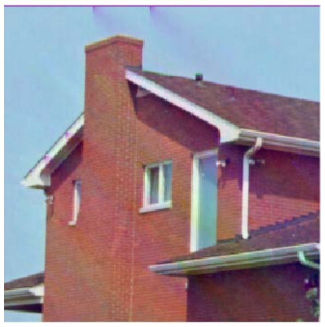

Interval Huffman coding

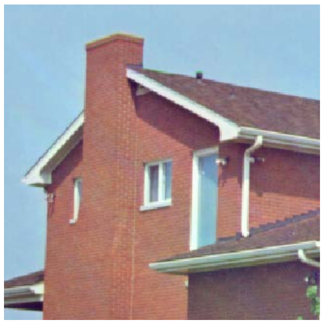

Standard Huffman coding

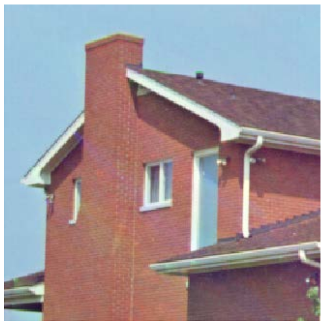
(b) Color-converted image.

Figure 4.5 Comparison results using predictive bi-level block coding, predictive interval Huffman coding and predictive Huffman coding at BER = 0.001 for “House” image.

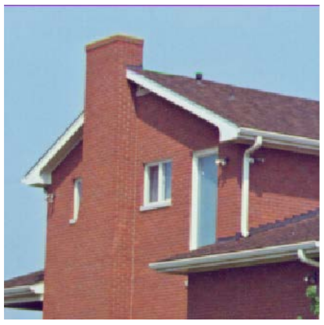

Original RGB image "House"

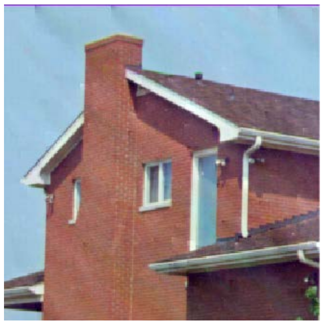

Bi-level block coding

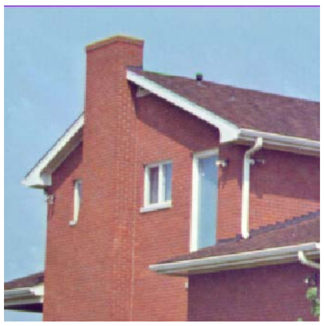

Interval Huffman coding

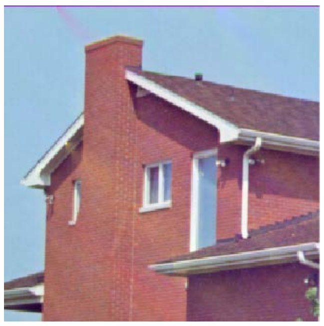

Standard Huffman coding

(a) RGB image without color space processing,


Original color-converted image
"House"


Bi-level block coding

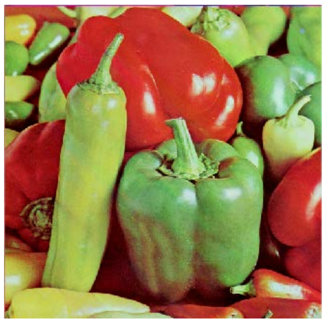

Interval Huffman coding

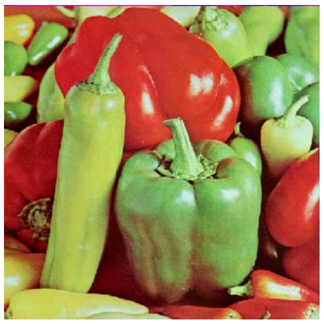

Standard Huffman coding

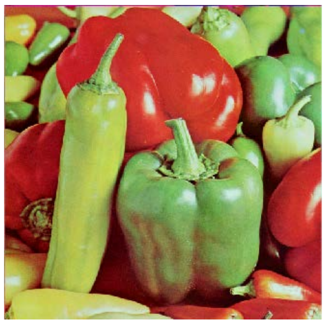
(b) Color-converted image.

Figure 4.6 Comparison results using predictive bi-level block coding, predictive interval Huffman coding and predictive Huffman coding at BER = 0.005 for "House" image.

Original RGB image “Pepper”

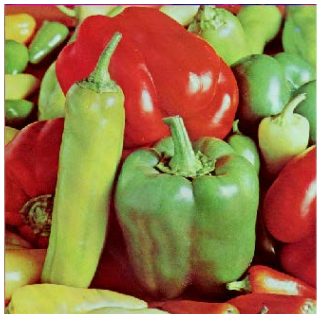

Bi-level block coding

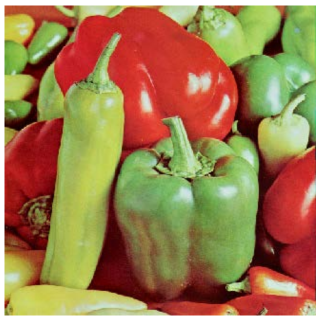

Interval Huffman coding

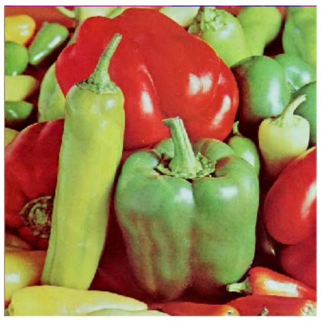

Standard Huffman coding

(a) RGB image without color space processing,

Original color-converted image “Pepper”

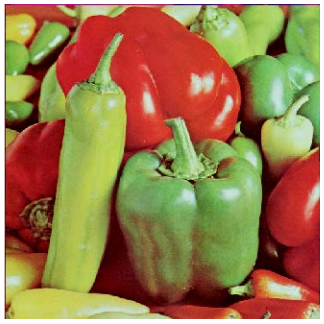

Bi-level block coding

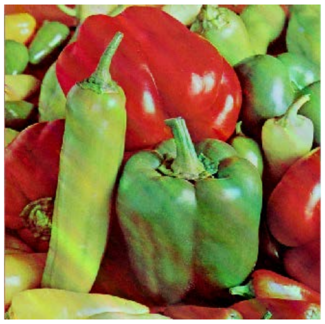

Interval Huffman coding

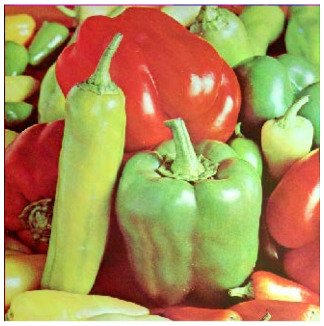

Standard Huffman coding

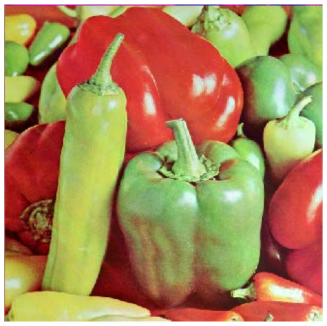
(b) Color-converted image.

Figure 4.7 Comparison results using predictive bi-level block coding, predictive interval Huffman coding and predictive Huffman coding at BER = 0.001 for “Pepper” image.

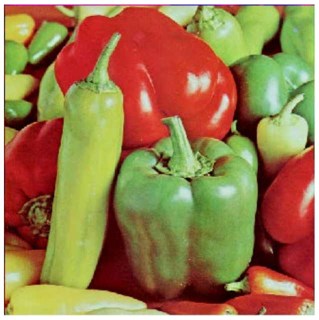

Original RGB image “Pepper”

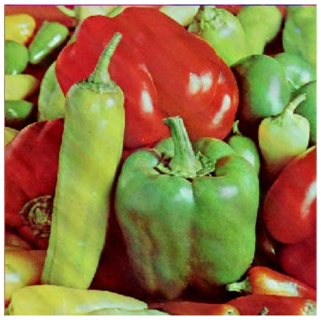

Bi-level block coding

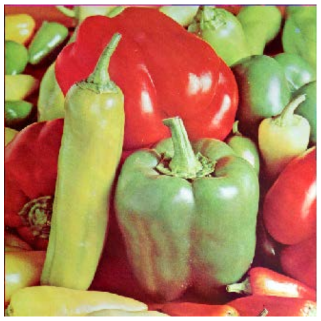

Interval Huffman coding

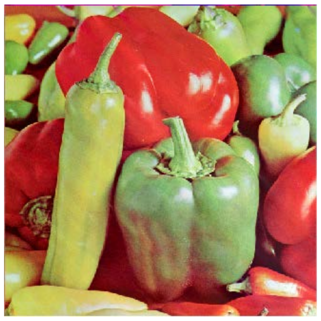

Standard Huffman coding

(a) RGB image without color space processing,


Original color-converted image
“Pepper”

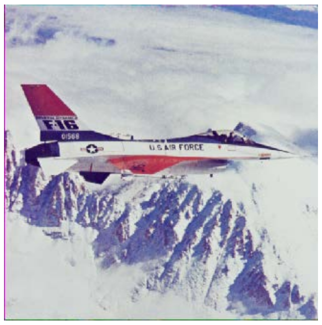

Bi-level block coding

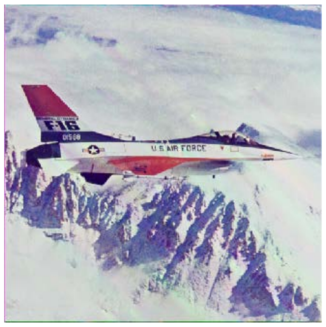

Interval Huffman coding

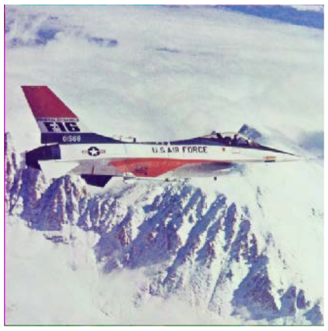

Standard Huffman coding

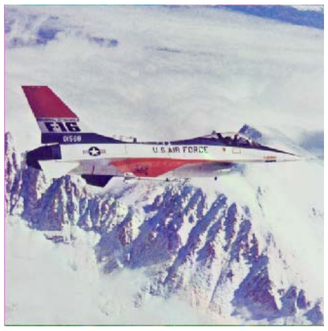
(b) Color-converted image.

Figure 4.8 Comparison results using predictive bi-level block coding, predictive interval Huffman coding and predictive Huffman coding at BER = 0.005 for “Pepper” image.

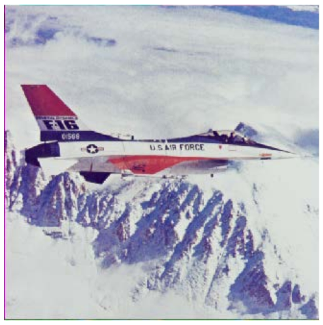

Original RGB image “Airplane”

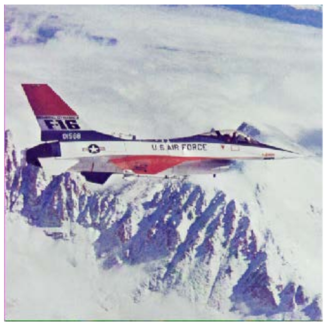

Bi-level block coding

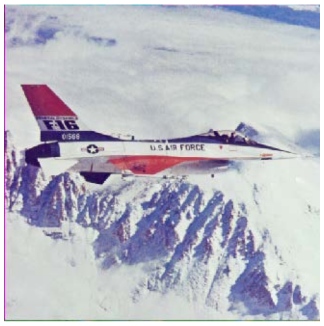

Interval Huffman coding

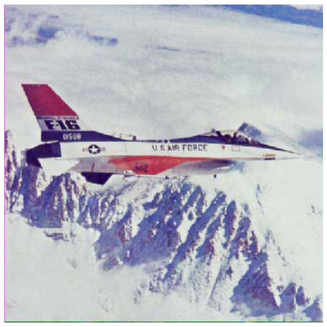

Standard Huffman coding

(a) RGB image without color space processing,


Original color-converted image “Airplane”

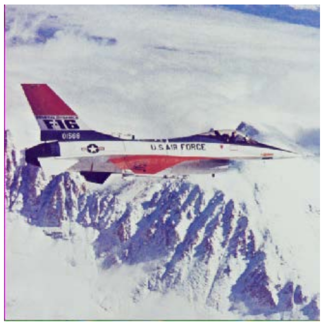

Bi-level block coding

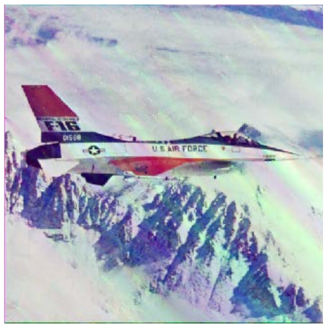

Interval Huffman coding

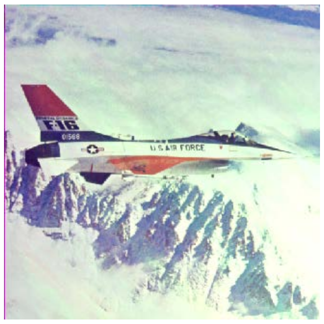

Standard Huffman coding

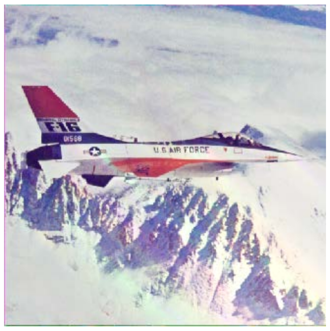
(b) Color-converted image.

Figure 4.9 Comparison results using predictive bi-level block coding, predictive interval Huffman coding and predictive Huffman coding at BER = 0.001 for “Airplane” image.

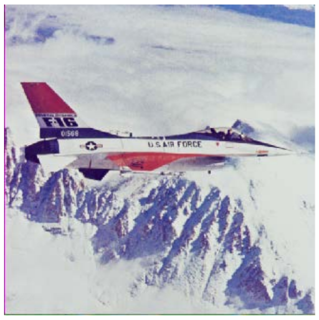

Original RGB image “Airplane”

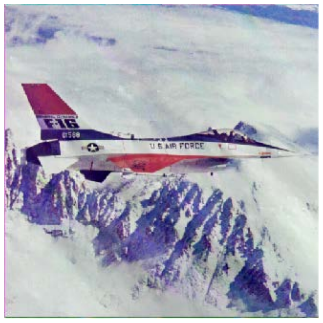

Bi-level block coding

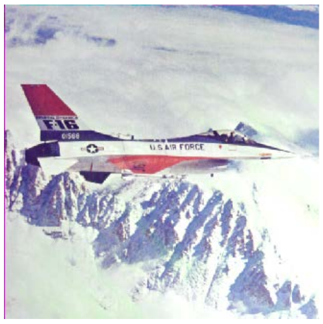

Interval Huffman coding

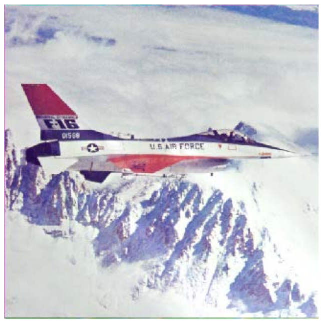

Standard Huffman coding

(a) RGB image without color space processing,


Original color-converted image “Airplane”


Bi-level block coding


Interval Huffman coding


Standard Huffman coding

(b) Color-converted image.

Figure 4.10 Comparison results using predictive bi-level block coding, predictive interval Huffman coding and predictive Huffman coding at BER = 0.005 for “Airplane” image.

With the increase of BER, it has a more significant impact on the image, resulting in image distortion. As shown in Figure 4.2, Figure 4.4, Figure 4.6, Figure 4.8 and Figure 4.10, images have varying degrees of distortion using the residue coding methods when the BER is 0.005.

## 4.2 PSNR Performances for Different Bit-error Rates

We have already known that the compression ratio and image quality of the color-converted image are higher than the RGB image without color space processing, so our next experiment will use the color-converted image for analysis. In this section, we mainly study the change of image quality with bit-error rate. Figures 4.11 to 4.15 show the PSNR performances with the change of the bit-error rate for color-converted images. Each PSNR is obtained by averaging the values from 10 independent runs.

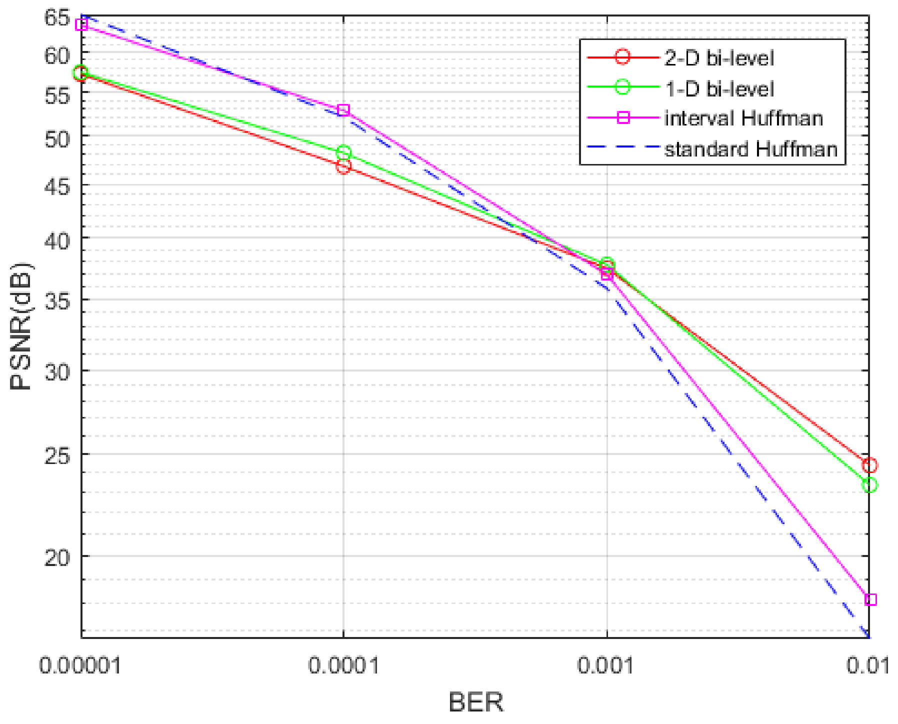


Figure 4.11 PSNR performances versus the bit-error rate for “Lena” image.

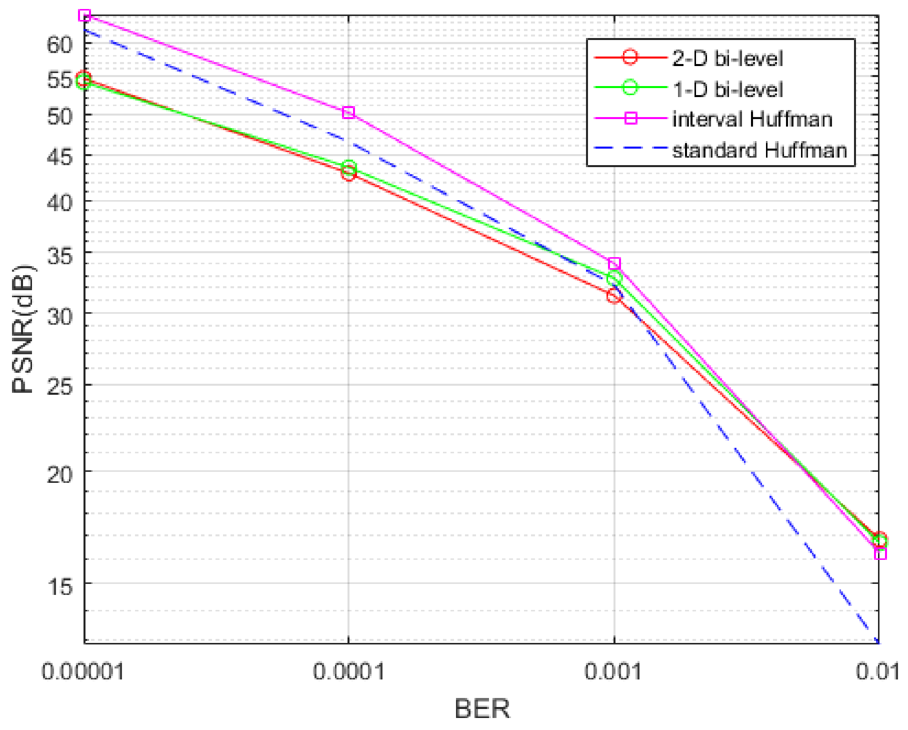


Figure 4.12 PSNR performances versus the bit-error rate for “Baboon” image.

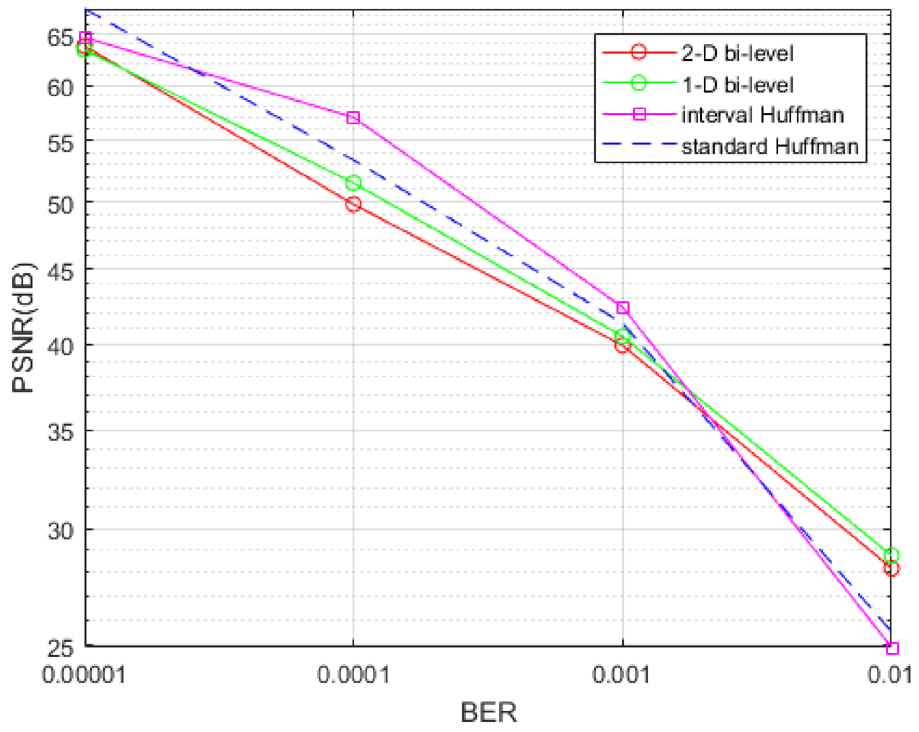


Figure 4.13 PSNR performances versus the bit-error rate for “House” image.

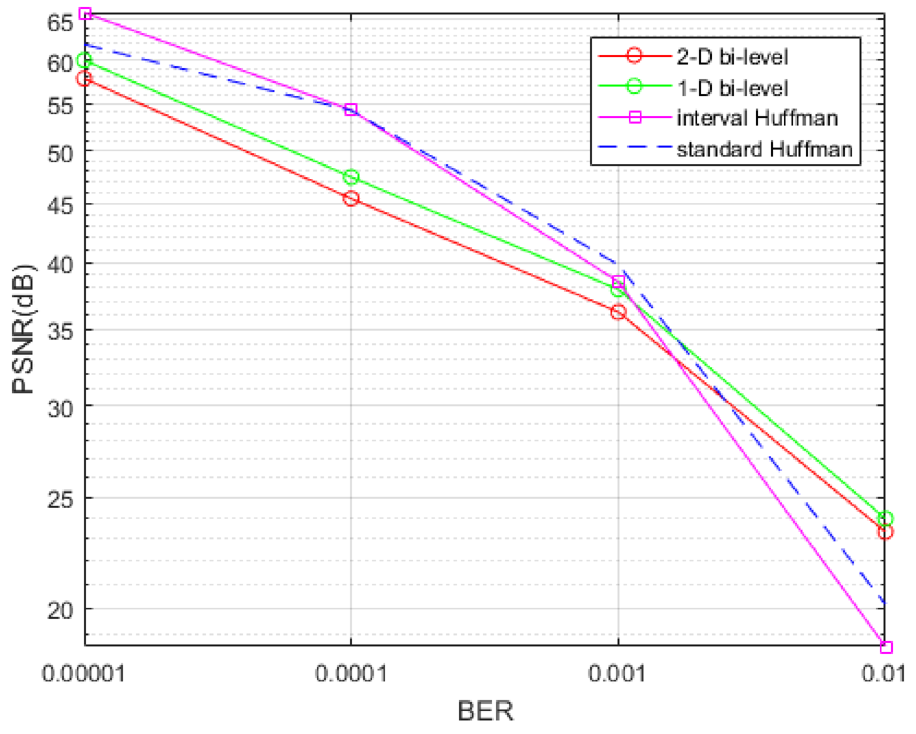


Figure 4.14 PSNR performances versus the bit-error rate for “Pepper” image.

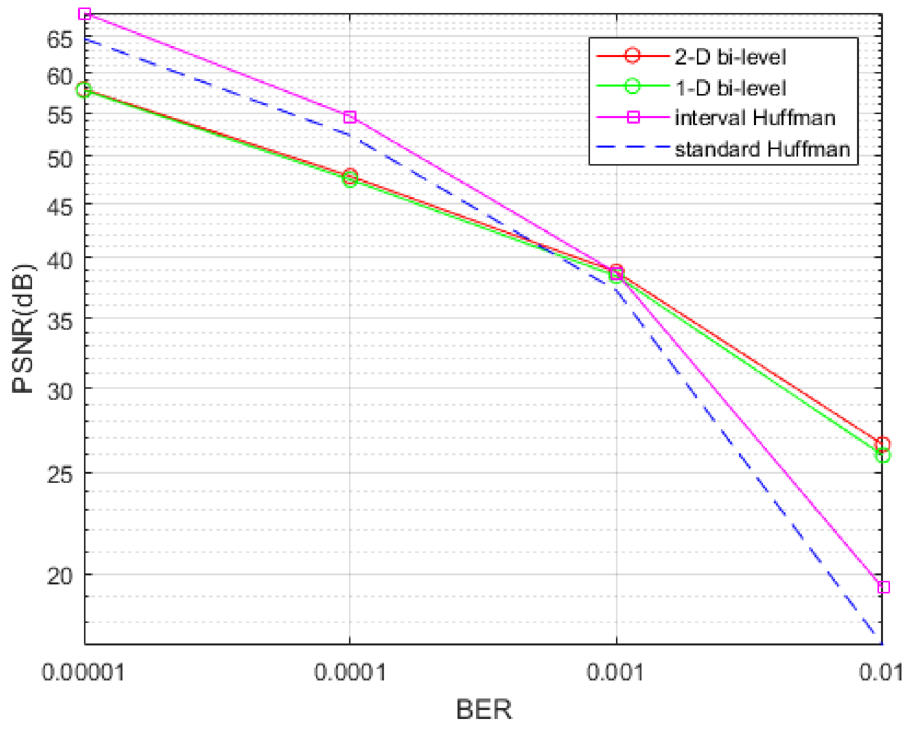


Figure 4.15 PSNR performances versus the bit-error rate for “Airplane” image.

It is easy to see that the PSNRs of 2-D and 1-D bi-level block coding algorithms are almost the same. When the BER is more than 0.001, bi-level block coding offers a higher value of PSNR than the other two methods. On the other hand, the image using bi-level block coding has the lowest value of PSNR when the BER is less than 0.001. However, bi-level block coding algorithm still has an excellent image quality.

## 4.3 CR Performances for Different Images

In order to further compare CR performances using different residue coding methods, we choose five images to do this experiment. The results are shown in Figure 4.16.

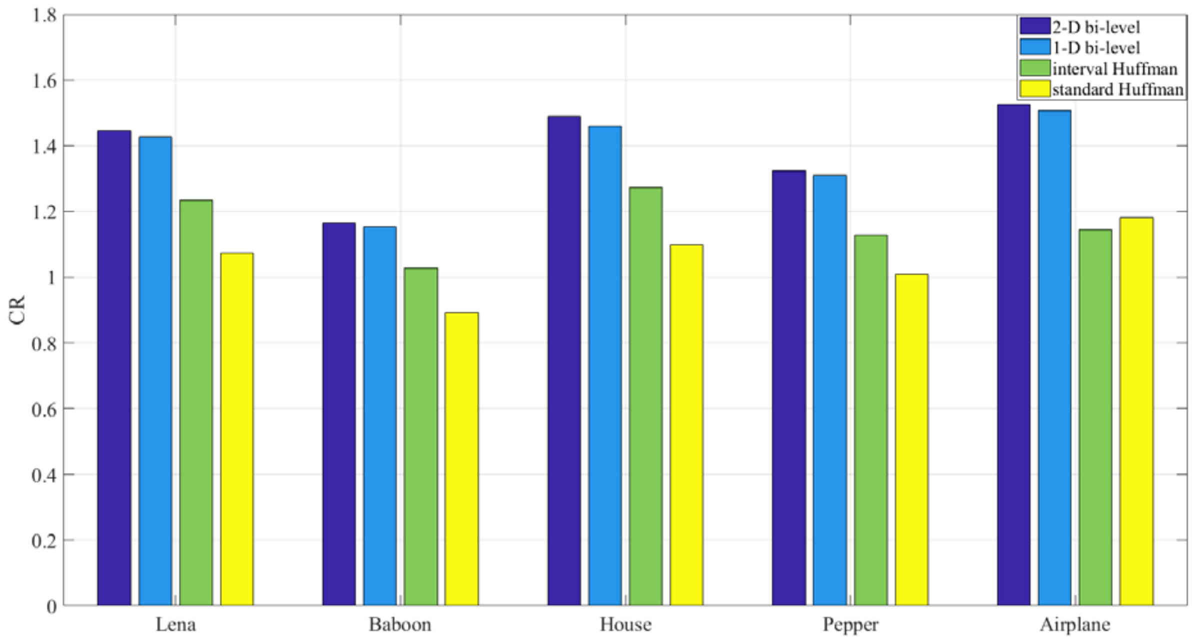


Figure 4.16 CR performances for five different color images.

Image using bi-level block coding algorithm has a higher compression ratio, and the performance of 2-D bi-level block coding is slightly better than the performance of 1-D bi-level block coding. Considering both image quality and compression ratio, the 2-D bi-level block coding algorithm is the best algorithm to compress the residue sequences.

# CHAPTER 5. CONCLUSION AND FUTURE WORK

In this paper, we have proposed four residue coding algorithms to deal with residue sequences. For the whole process of color image compression with bit-error awareness, there are three significant stages: converting RGB image to YCrCb image; applying linear predictors for the converted image. Note that the best combination of color space conversion and predictors are selected by using the particle swarm optimization or the genetic algorithm; encoding residue sequences using different methods, which are 2-D and 1-D bi-level block coding, interval Huffman coding and standard Huffman coding. Key information parameters of these three stages are protected using (7,4) Hamming code. The algorithm using color space conversion obtains the higher compression ratio and image quality than the ones without using color space conversion.

Our experimental results validate the coding performances in terms of the compression ratio (CR) and peak signal to noise ratio (PSNR). When the bit-error rate (BER) is more than 0.001, bi-level block coding offers the highest PSNR, which means the highest image quality. Even if the BER is less than 0.001, this method still maintains a good value of PSNR. For compression ratio, 2-D bi-level block coding achieves the highest CR. In summary, the 2-D bi-level block coding algorithm is the best residue coding method among all the algorithms.

Through this thesis work, we make the following contributions:

1. We propose the methods of converting RGB image to YCrCb image and predictors on YCrCb image to improve the image quality and compression ratio.
2. GA algorithm and PSO algorithm are adopted to search the best combination of color space conversions and predictors.
3. We extend 1-D bi-level block coding to N-D bi-level block coding. As a special case, 2-D bi-level block coding is adopted.
4. After validation, new 2-D bi-level blocking coding shows the improved results over 1-D bi-level block coding, interval Huffman coding, and standard Huffman coding.

Our future work may include:

1. Applying convolutional neural network (CNN) to search the best combination of color space conversion and predictors.
2. Applying the bit-error aware lossless compression algorithms to video sequences.

# REFERENCES

[1] X. Wu, N. Memon, “Context-based, adaptive, lossless image coding,” *IEEE Trans. on Communications*, Vol. 45, No. 4, April, 1997.

[2] T. Lin, Pengwei Hao, “Compound image compression for real-time computer screen image transmission,” *IEEE Trans. on Image Processing*, vol. 14, no. 8, pp. 993-1005, Aug. 2005.

[3] S. Singh, M. Mishra, P. Gupta, “Image compression on biomedical images using predictive coding with the help of ROI,” *2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN)*, Noida, 2015, pp. 120-125.

[4] C. D. Rawat, S. Rao, “Evaluation of Burrows Wheeler Transform based image compression algorithm for multimedia applications,” *2014 International Conference on Advances in Communication and Computing Technologies (ICACACT 2014)*, Mumbai, 2014, pp. 1-2.

[5] J. Mielikainen, B. Huang, “Lossless compression of hyperspectral images using clustered linear prediction with Adaptive prediction length,” *IEEE Geoscience and Remote Sensing Letters*, vol. 9, no. 6, pp. 1118-1121, Nov. 2012.

[6] S. Miaou, F. Ke, S. Chen, “A lossless compression method for medical image sequences using JPEG-LS and interframe coding,” *IEEE Trans. on Information Technology in Biomedicine*, vol. 13, no. 5, pp. 818-821, Sept. 2009.

[7] V. Sanchez, R. Abugharbieh, P. Nasiopoulos, “Symmetry-based scalable lossless compression of 3D medical image data, ” *IEEE Trans. on Medical Imaging*, vol. 28, no. 7, pp. 1062-1072, July 2009.

[8] H. Wu, X. Sun, J. Yang, W. Zeng, F. Wu, “Lossless compression of JPEG coded photo collections,” *IEEE Trans. on Image Processing*, vol. 25, no. 6, pp. 2684-2696, June 2016.

[9] S. Kim, N. I. Cho, “Hierarchical prediction and context adaptive coding for lossless color image compression,” *IEEE Trans. on Image Processing*, vol. 23, no. 1, pp. 445-449, Jan. 2014.

[10] R. Kannan, C. Eswaran, “Lossless compression schemes for ECG signals using neural network predictors,” *EURASIP J. Adv. Signal Process., (Special Issue on Advances Electrocardiogram Signal Processing and Analysis)*, vol. 2007, pp. 1–20.

[11] R. Starrosolski, “Simple fast and adaptive lossless image compression algorithm,” *Softw. Pract. xper.*, Vol. 37, pp. 65-91, 2007.

[12] T. Leung, M. W. Marcellin, A. Bilgin, “Visually lossless compression of windowed images,” *2013 Data Compression Conference,* Snowbird, UT, 2013, pp. 504-504.

[13] A. Koski, “Lossless ECG Coding,” *Computer methods and programs in biomedicine*, Vol. 52, No. 1 pp. 23–33, 1997.

[14] V. Sanchez, P. Nasiopoulos, R. Abugharbieh, “Efficient lossless compression of 4-D medical images based on the advanced video coding scheme,” *IEEE Trans. on Information Technology in Biomedicine*, vol. 12, no. 4, pp. 442-446, July 2008.

[15] N. Sriraam, C. Eswaran, “Context based error modeling for lossless compression of EEG signals using neural networks,” *Journal of Medical Systems,* Vol. 30, No.6, pp.439–448, December, 2006.

[16] M. Weinbergner, G. Seroussi, G. Sapiro, “The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS,” *IEEE Trans. on Image Processing*, Vol. 9, No. 8, August 2000.

[17] S. D. Stearns, L. Tan, N. Magotra, “Lossless compression of waveform data for efficient storage and transmission,” *IEEE Trans. Geosci. Remote Sensing*, vol. 31, no. 3, pp. 645–654, May 1993.

[18] S. Lin, D. Jr. Costello, *Error Control Coding: Fundamentals and Applications.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1983.

[19] S. Rhee, C. Kim, J. Kim, Y. Jee, “Concatenated Reed-Solomon code with hamming code for DRAM controller,” *2010 Second International Conference on Computer Engineering and Applications*, Bali Island, 2010, pp. 291-295.

[20] L. Tan, J. Jiang, *Digital Signal Processing: Fundamentals and Applications*. Third Edition, Elsevier/Academics, 2018.

[21] T. Strutz, A. Leipnitz, “Reversible color spaces without increased bit depth and their adaptive selection,” *IEEE Signal Processing Letters*, vol. 22, no. 9, pp. 1269-1273, Sept. 2015.

[22] X. Peng, J. Hou, L. Tan, J. Chen, J. Jiang, X. Guo, “Bit-error aware lossless color image compression,” *2019 IEEE International Conference on Electro/Information Technology*, pp. 126-131, Brookings, South Dakota, May 2019.

[23] L. Tan, L. Wang, “Bit-error aware lossless image compression,” *International Journal of Modern Engineering*, vol. 11, no. 2, pp. 54-59, Spring/Summer 2011.

[24] S. D. Stearns, *Digital Signal Processing with Examples in MATLAB*. CRC Press, 2002.

[25] Z. Li, M. S. Drew, *Fundamentals of Multimedia*. Prentice Hall, Upper Saddle River, NJ 07458, 2004.

[26] S. D. Stearns, L. Tan, N. Magotra, “A bi-level coding technique for compressing broadband residue sequences,” *Digital Signal Processing*, Vol. 2, No. 3, pp. 146-156, July 1992.

[27] S. D. Stearns, “Arithmetic coding in lossless waveform compression,” *IEEE Trans. Signal Process.*, vol. 43, pp. 1874–1879, 1995.

[28] G. Zeng, N. Ahmed, “A block coding technique for encoding sparse binary patterns,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 5, pp. 778–780, May 1989.

[29] L. Tan, J. Jiang, “A bi-level block coding technique for encoding data sequences with sparse distributions,” *Technol. Interface J.*, vol. 9, no. 1, 2008.

[30] L. Tan, J. Jiang, Y. Zhang, “Bit-error aware lossless compression of waveform data,” *IEEE Signal Processing Letters*, vol. 17, no. 6, pp. 547-550, June 2010.

[31] Wikipedia. Available: https://en.wikipedia.org/wiki/Huffman_coding

[32] F. Marcelloni, M. Vecchio, “A simple algorithm for data compression in wireless sensor networks,” *IEEE Commun. Lett.*, vol. 12, pp. 411–413, Jun. 2008.

[33] Wikipedia. Available: https://en.wikipedia.org/wiki/Metaheuristic

[34] P. Guo, X. Wang, Y. Han, “The enhanced genetic algorithms for the optimization design,” *2010 3rd International Conference on Biomedical Engineering and Informatics*, Yantai, 2010, pp. 2990-2994.

[35] Mojtaba Ahmadieh Khanesar, Mohammad Teshnehlab, Mahdi Aliyari Shoorehdeli, “A novel binary particle swarm optimization,” *2007 Mediterranean Conference on Control & Automation*, Athens, pp. 1-6, July 2007.

[36] X. Wu, “A density adjustment based particle swarm optimization learning algorithm for neural network design,” *2011 International Conference on Electrical and Control Engineering*, Yichang, 2011, pp. 2829-2832.

# PUBLICATIONS

**Conference Papers:**

J. Dai, V. Vijayarajan, X. Peng, L. Tan, J. Jiang, “Speech recognition using sparse discrete wavelet decomposition feature extraction” 2018 *IEEE International Conference on Electro/Information Technology*, pp. 812-816, Oakland University, Rochester, Michigan, May 2018.

X. Peng, J. Hou, L. Tan, J. Chen, J. Jiang, X. Guo, “Bit-error aware lossless color image compression,” *2019 IEEE International Conference on Electro/Information Technology*, pp. 126-131, Brookings, South Dakota, May 2019.