IMPROVED GOLOMB CODE FOR JOINT GEOMETRIC DISTRIBUTION DATA AND ITS APPLICATIONS IN IMAGE PROCESSING

Jian-Jiun Ding (*丁建均*), *Soo-Chang Pei* (*貝蘇章*), *Wei-Yi Wei* (*魏維毅*)
Tzu-Heng Henry Lee (*李自恆*)

Department of Electrical Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Rd., 10617, Taipei, Taiwan, R.O.C, Tel 02-33669652, Fax 02-23671909,
Email: djj@cc.ee.ntu.edu.tw, pei@cc.ee.ntu.edu.tw, r97942024@ntu.edu.tw
henrylee915@gmail.com,

ABSTRACT

Golomb coding is a special case of Huffman coding that is optimal for the data with geometric progression distribution probability. In this paper, we will generalize the Golomb coding by joint probability. In image progression, there are many conditions that the probability of a data not only has geometric distribution probability but also depends on another data. (For example, the difference between the intensities of two pixels is highly dependent on their distance) Based on the requirement, we improve the Golomb codes by considering the joint probability and make Golomb coding more flexible and efficient. The proposed modified Golomb codes will be useful in digital signal and image processing. For example, after performing the DCT, we can use the high correlation between the average amplitudes of low frequency components and middle frequency components to improve the compression ratio by the proposed coding scheme.

1. HUFFMAN CODING

The Huffman algorithm was first proposed by D. A. Huffman [1], and it is a variable length coding algorithm. This data compression algorithm has been widely adopted in many image and video compression standards such as JPEG and MPEG. The Huffman algorithm is proved to provide the shortest average codeword length per symbol.

There are three important properties of the Huffman algorithm [1]-[7].

1. *Unique Prefix Property*: This property means that no any Huffman code for a particular symbol is a prefix of that for the other symbols. We take Fig. 1 for example, the code 1 assigned to the symbol a_2 is not a prefix of the other codes. The code 00 assigned to the symbol a_6 is not a prefix of the other codes.
2. *Instantaneous Decoding Property*: The decoder can efficiently decode the received code because of the unique prefix property. We take Fig. 1 for introduction again. If the decoder received code 1, then the decoder can immediately decode 1 to the symbol a_2 without waiting for the later received codes.
3. *Optimality*: It has been proved that Huffman code is optimal for a given data model with accurate probability distribution.

However, the Huffman coding has some disadvantages. First, it is only suitable for data with finite number of possible values. If the possible value of a data is from 0 to ∞ (such as the distance between two points), or from $-\infty$ to ∞ (such as the AC value of a data), the Huffman coding algorithm falls to work. Second, although Huffman coding can minimize the coding length, we require extra memory to record the "Huffman table", which shows the corresponding relations between codes and data values. This is not good for compression.

In [8], Golomb proposed a new type of codes, which is a lossless data compression algorithm designed to encode the positive integers with the one-sided geometric distribution, i.e.,

$$p(x=k)=p^{k-1}q, \quad q=1-p, \quad k=0,1,2,\ldots..,N,\ N\to\infty \tag{1}$$

Golomb found a regularity: When

$$p=(1/2)^{(1/n)}, \tag{2}$$

then after the $\lceil \log_2(n) \rceil^{\text{th}}$ layer, each layer should have n leaf nodes (see Section 2). Then, in [9], they proved the theory proposed by Golomb and modified the relation between p and n is modified as

$$p^n + p^{n+1} \le 1 \le p^n + p^{n-1}. \tag{3}$$

In [10], Rice proposed the adaptive prefix coding scheme using the subset of Golomb coding where n is a power of 2. In [11] and [12], the Golomb coding was extended to the two-sided case and the offset case.

Note that the geometric distribution sequence in (1) is an infinite sequence. Therefore, it is impossible to use the conventional Huffman coding algorithm to code this type of data. However, Golomb et. al [8]-[12] found the way to

encode it and proved that the results satisfy the requirement of Huffman coding, i.e., the node in the L^{th} layer always has the probability no less than that of the node in the $(L+1)^{th}$ layer. Even if the data do not have geometric distribution probability, we can use a geometric progression sequence to approximate its probability then encode it. Thus, the Golomb code provides a way to **encode the data with infinite possible values**.

Another important advantage of the Golomb code is that **no table is required** to retrieve the original data. Since the coding structure is dependent only on n, we only have to store the value of n. In the heading of JPEG and MPEG, we always need some bytes to store the table for Huffman coding. Now, with Golomb coding, these tables can be saved and the size of heading can be smaller.

In this paper, we will use the concept of the **joint probability** to further extend the Golomb coding algorithm. In nature, there are many conditions that a data not only has the probability near to geometric distribution but also depend on another data. For example, if x is the hamming distance between two points in a curve and y is the number of pixels between the two points along the curve, then the probability of $|y - 0.8x|$ not only has geometric distribution but also varies with x. In these cases, the probabilities can also be expressed as the form in (1), but p is not a constant and is dependent on other data.

We will modify the Golomb codes according to the joint probability. We will also modify the coding scheme and showed that, even with the joint probability, we only have to store one extra parameter (no table is required) to encode and retrieve the original data.

In Section 2, we review the Golomb coding algorithm for a geometric distribution sequence. In Section 3, we will propose the modified Golomb codes based on the joint probability. We also propose the coding scheme for the modified Golomb codes. In Section 4, we perform several simulations and show that the coding algorithm is indeed useful in image processing.

2. CODING FOR GEOMETRIC DISTRIBUTION SEQUENCE

Suppose that the data y has infinite number of possible values ($y = k$, k can be 0, 1, 2, ….., N, $N \to \infty$.}. We use s_k to denote the event of $y = k$ and use $P(s_k)$ to denote its probability. Suppose that

$$P(s_0) > P(s_1) > P(s_2) > P(s_3) > \cdots\cdots \quad (4)$$

If $P(s_k)$ is a **geometric progressing series**, i.e.,

$$P(s_k) = (1-p)p^k . \quad (5)$$

$$k = 0, 1, 2, \ldots..., N,\ N \to \infty$$

When

$$p^n + p^{n+1} \le 1 \le p^n + p^{n-1} , \quad (6)$$

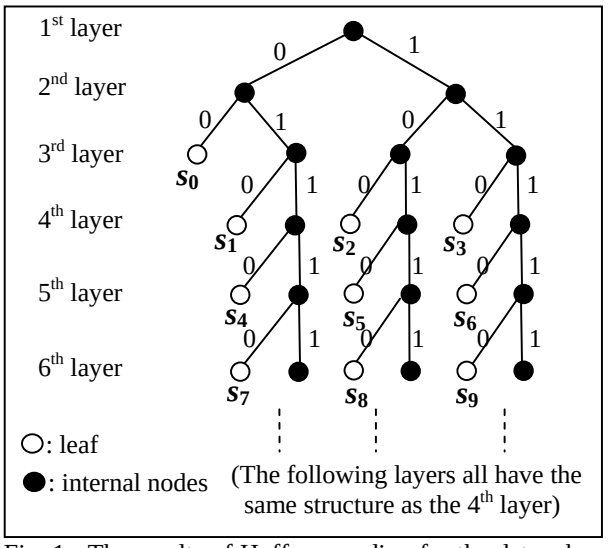

Fig. 1: The results of Huffman coding for the data whose probabilities form a geometric progression series $P(s_k) = 0.2(0.8)^k$, $k = 0, 1, 2, \ldots..., N$, $N \to \infty$. In addition to layers 1-3, other layers all have 3 leaf nodes and 3 internal nodes.

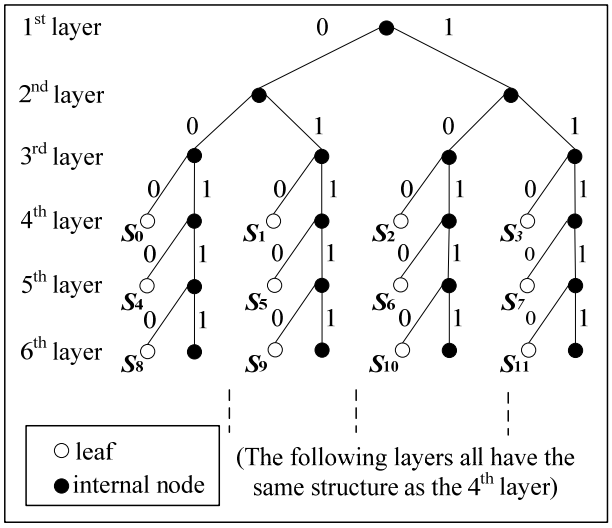

Fig. 2: The Huffman coding results for the data whose probabilities are $P(s_k) = 0.15(0.85)^k$, $k = 0, 1, 2, \ldots...$ In addition to layers 1-3, other layers all have 4 leaf nodes and 4 internal nodes.

in other words,

$$n - 1 \le -\frac{\log(p+1)}{\log(p)} \le n , \quad (7)$$

the Huffman coding of $\{s_k\}$ will obey the following rules:

(Rule 1) If

$$2^{H-1} < n \le 2^H \quad (8)$$

then the first H layers has no leafs. That is, when $1 \le L \le H$, the L^{th} layer contains exactly 2^{L-1} internal nodes.

(Rule 2) The $(H+1)^{th}$ layer contains

$$2^H - n \quad \text{leaf nodes}$$
$$\text{and} \quad n \quad \text{internal nodes,} \tag{9}$$

and the leaf nodes correspond to $s_0, s_1, \cdots s_{2^H-n-1}$.

(Rule 3) In the L^{th} layer where $L > H+1$, there are always

$$n \text{ leaf nodes} \quad \text{and} \quad n \text{ internal nodes,} \tag{10}$$

no matter what the value of L is. Moreover, the leaf nodes correspond to the data of

$$s_{(L-M-2)n+a}, \quad s_{(L-M-2)n+a+1}, \quad \cdots\cdots, \quad s_{(L-M-1)n+a-1}, \tag{11}$$

where $$a = 2^H - n. \tag{12}$$

We give an example as follows. If $p = 0.8$, then since from (5),

$$-\frac{\log(p+1)}{\log(p)} = 2.6341, \tag{13}$$

n should be equal to 3. Then, from (8), H should be equal to 2. Therefore, the first two layers have no leaf. The 3rd layer has $2^H - 3 = 1$ leaf node and 3 internal nodes. For the layer below the 3rd layer, there are always 3 leaf nodes and 3 internal nodes. Therefore, the result of Huffman coding for the data with the probabilities of $0.2(0.8)^k$ should have the form as in Fig. 1

Similarly, when $p = 0.85$, since

$$-\frac{\log(p+1)}{\log(p)} = 3.7853, \tag{14}$$

n should be 4 and H should be 2. Therefore, the first 2 layers have no leafs. From (9), the 3rd layer has 0 leaf nodes and 4 internal nodes. From (10)-(12), the L^{th} layer ($L > 3$) always have 4 leaf nodes and 4 internal nodes, as in Fig. 2.

(Rule 4) After determining how many leaf nodes in each layer, we determine the codes corresponding to s_k. In fact, there are many ways to encode s_k.. Two of the ways are describe as follows.

The first one is to calculate the quotient $q = \lfloor k/n \rfloor$ and the remainder r = mod(k, n). The quotient q is encoded by q '1's followed by one '0'. The remainder r is encoded by H–1 bits if $r < a$ where H and a are defined in (8) and (12). If $r \geq a$, we encode $r + a$ instead of r by H bits. The coding result for q is placed in the prefix and the coding result for r (or $r + a$) is placed in the suffix.

We give a simple example where the parameter n is 5. Therefore, we can obtain $H = \lceil \log_2(5) \rceil = 3$, and the threshold for the remainder is $a = 2^H - n = 2^3 - 5 = 3$. Based on these parameters, we can encode one positive integer 4 as q=0 and r=4. Because $r = 4 > a = 3$, r is encoded as $r + a = 7$. Finally, we can convert q and $r + a$ into binary form as 0 and 111. Thus, the Golomb code for integer 4 is 0111. Follow the similar procedure, we can obtain the final Golomb code as shown in Table 1.

Table 1: Golomb Codes for n = 5 (type 1)

k	q	r	Codeword	k	q	r	Codeword
0	0	0	000	8	1	3	10110
1	0	1	001	9	1	4	10111
2	0	2	010	10	2	0	11000
3	0	3	0110	11	2	1	11001
4	0	4	0111	12	2	2	11010
5	1	0	1000	13	2	3	110110
6	1	1	1001	14	2	4	110111
7	1	2	1010	15	3	0	111000

Table2: Golomb Codes for n = 5 (type 2)

k	q	r	Codeword	k	q	r	Codeword
0	–1	5	101	8	1	0	00010
1	–1	6	110	9	1	1	00110
2	–1	7	111	10	1	2	01010
3	0	0	0000	11	1	3	01110
4	0	1	0010	12	1	4	10010
5	0	2	0100	13	2	0	000110
6	0	3	0110	14	2	1	001110
7	0	4	1000	15	2	2	010110

Another coding method is to calculate q from $q = \lfloor (k-a)/n \rfloor$. If $q = -1$, then $r = 2^H - a + k$. If $q \geq 0$, then r = mod(k–a, n). Then, we code r by H bits in the prefix and when $q \geq 0$ we code q by q '1's followed by one '0' in the suffix. An example where n = 5 is shown in Table 2. Note that this coding method is used in Figs. 1 and 2.

3. IMPROVED THE GOLOMB CODE BY JOINT PROBABILITY

The Golomb code provides an efficient way to encode a data with geometric progression probability. However, sometimes although the probability of a data has geometric distribution, but the value of p in (5) depends on another data. In this case, we should modify the Golomb code and consider the joint probability to encode the data.

The examples that the probability of y is near to geometric distribution as in (5) but the value of p varies with another data x are listed as follows.

(1) x is the distance between two locations and y is the difference of temperatures of the two locations

(2) x is the number of days and y is the variation of the prices of commodities after x days.

(3) x is the height of a person and y is the difference between the standard weight and the weight of the person.

(4) x is the time duration and y is the moving distance of an animal.

(5) x is the distance between two pixels and y is the difference of intensities between the two pixels.

(6) x is the lower frequency AC values and y is the higher frequency AC values of the DCT for a 8×8 block (This will be illustrated in Section 4)

(7) x is the area of a pattern and y is the difference between the circumference of the pattern and $2\sqrt{\pi x}$.

(8) x is the hamming distance between two dots on a curve and y is the difference between round(0.8x) and the number of pixels between the two dots on the curve (This will be illustrated in Section 4) .

(9) In video compression, x can be the time difference between two pictures and y is the change of the direction of a moving object. Moreover, y can also be the amount of motion compensation.

In these cases, although y has geometric distribution probability, the probability is dependent on x. Therefore, to code the data, we should modify the Golomb coding. Suppose that the values of y are integers and the mean of $|y|$ is proportional to a function of x:

$$mean(|y|) \propto |f(x)|, \tag{15}$$

then we can use the following process to code $y[\tau]$.

(Step 1): Choose

$$X_1 = \max\{|f(x[\tau])|\}. \tag{16}$$

(Step 2): Then, scale the value of y according to

$$y_1[\tau] = \frac{y[\tau]X_1}{|f(x[\tau])|}. \tag{17}$$

(Step 3): Find the best value of p_1 such that the probability of $|y_1[\tau]|$ can be approximated by a geometric progression series:

$$P(|y_1[\tau]| = k) \approx (1-p_1)p_1^k. \tag{18}$$

(Step 4): Then, the probability of $|y[\tau]| = k$ can be approximated by:

$$P(|y[\tau]| = k) \approx (1-p[\tau])p^k[\tau], \tag{19}$$

where

$$p[\tau] = \frac{1}{\frac{X_1}{|f(x[\tau])|}\left(\frac{1}{p_1}-1\right)+1}. \tag{20}$$

$p[\tau]$ can also be expressed by the simpler form:

$$p[\tau] = p_1^{\frac{X_1}{|f(x[\tau])|}}. \tag{21}$$

(Step 5): Therefore, we can determine $n[\tau]$ (the number of leaf nodes in each layer) according to

$$n[\tau]-1 \le -\frac{\log(p[\tau]+1)}{\log(p[\tau])} \le n[\tau]. \tag{22}$$

(Step 6): After $n[\tau]$ is determined, we can use Rules 1-4 described in Section 2 to encode $|y[\tau]|$. Moreover, since $y[\tau]$ can be positive or negative, we use an extra bit to encode the sign if $y[\tau] \neq 0$.

There are some things to be noticed. First, the parameter $p[\tau]$ in (19) and (20) is not a constant and varies with $x[\tau]$. Second, since $n[\tau]$ is not a constant and varies with $x[\tau]$, thus, for different $x[\tau]$, we use different method to encode $y[\tau]$. In other words, we adjust the number of leaf nodes in each layer and **encode $y[\tau]$ adaptively** according to **the value of $x[\tau]$**.

Step 4 is illustrated as follows. First, for a data with geometric distribution probability as

$$P(z=k) = (1-p)p^k \tag{23}$$

k = 0, 1, 2, ….., N, $N \to \infty$,

then

$$\begin{aligned} mean(z) &= \sum_{k=0}^{\infty} kP(z=k) \\ &= P(z\ge 1)+P(z\ge 2)+P(z\ge 3)+\cdots\cdots \\ &= p+p^2+p^3+\cdots\cdots = \frac{p}{1-p}. \end{aligned} \tag{24}$$

In Step 3, we have estimated that when $f(x[\tau]) = X_1$, the probability of $y[\tau]$ is near to $(1-p_1)p_1^k$. Thus, from (24),

$$mean(|y[\tau]|) = \frac{p_1}{1-p_1} \quad \text{when } |x[\tau]| = X_1. \tag{25}$$

In the case where $f(x[\tau]) < X_1$, from (15), the mean of $|y[\tau]|$ is

$$mean(|y[\tau]|) = \frac{p_1}{1-p_1}\frac{|f(x[\tau])|}{X_1}. \tag{26}$$

If the probability of $|y[\tau]|$ can be expressed as the form in (19), then from (24), $p[\tau]$ should satisfy

$$\frac{p[\tau]}{1-p[\tau]} = \frac{p_1}{1-p_1}\frac{|f(x[\tau])|}{X_1}. \tag{27}$$

After some calculation, we obtain (20). Eq. (21) comes from the fact that if

$$p_1 = 1-\varepsilon \quad \text{where } \varepsilon \cong 0, \tag{28}$$

then from (20),

$$p[\tau] \cong \frac{1}{\frac{X_1}{|f(x[\tau])|}(1+\varepsilon-1)+1} \cong 1-\varepsilon\frac{X_1}{|f(x[\tau])|}. \quad (29)$$

Moreover,

$$p_1^{\frac{X_1}{|f(x[\tau])|}} = (1-\varepsilon)^{\frac{X_1}{|f(x[\tau])|}} \cong 1-\varepsilon\frac{X_1}{|f(x[\tau])|} \quad (30)$$

when $\varepsilon \cong 0$. Thus, from (29) and (30), (21) is proved.

Then, we discuss the problem about how to choose the value of p_1 in Step 3. There are two methods. The first one is to try several possible values and find which one has the best compression ratio. We suggest that the candidates of p_1 are

$$p_1 = 2^{-\frac{16}{cX_1}}, \quad (31)$$

where c = 1, 2, 3, …., 16, 18, 20, 22, ….32, 36, 40, 48, 56, 64, 80, 128, or 256. Note that c and hence p_1 has 32 possible values. Although in theorem p_1 can be chosen as any real number between 0 and 1, for the consideration of compression, we constrain p_1 to be one of the 32 possible values described above.

Then, we use (20) and (22) to determine the values of $p[\tau]$ and $n[\tau]$ for each p_1 and use the following formula to estimate the number of bits of the resulted codes:

$$\sum_{\tau} H[\tau]+J[\tau]+q, \quad (32)$$

where $$H[\tau] = \lceil \log_2 n[\tau] \rceil, \quad (33)$$

$$J[\tau] = \left\lceil \frac{|y[\tau]| - 2^{H[\tau]} + n[\tau]}{n[\tau]} \right\rceil \quad (34)$$

$\lceil\ \rceil$ means rounding toward infinite, and q is the number of $y[\tau]$ that is not equal to 0. Then, we vary the value of c in (31) and hence the value of p_1 iteratively to minimize (32). After the optimal p_1 is chosen, we calculate $p[\tau]$ and $n[\tau]$ and use Step 6 to encode $y[\tau]$.

Another faster way to choose p_1 is to calculate mean($|y_1[\tau]|$) at first. Then, from (25), the optimal p_1 and c can be estimated from:

$$p_1 \cong \frac{mean(|y_1[\tau]|)}{mean(|y_1[\tau]|)+1}, \quad (35)$$

$$c \cong \frac{-16}{X_1 \log_2 p_1}. \quad (36)$$

Then, we also use (32) to estimate the number of bits of the resulted codes. We can adjust the value of c around the values calculated above to minimize (32) and find the optimal values of p_1.

Note that, as the Golomb code, for the modified Golomb code described above, no table is required. We only have to use 5 bits to encode the value of c in (31) and hence the value of p_1. This is good for compression.

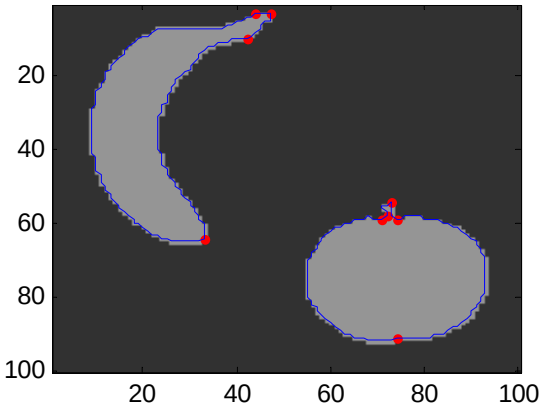

Fig. 3: An image with 9 dots on the edges of objects.

Table 3: Data of the 9 dots in Fig. 3. (A) The coordinates, (B) the values of x, i.e., the hamming distance between the dot and the next dot (C) the values of y (defined in (37).

Curve 1				
(A)	(3,44)	(3,47)	(10,42)	(64,33)
(B)	3	12	63	72
(C)	1	–2	10	32
Curve 2				
(A)	(59,71)	(58,72)	(54,73)	(59,74)	(91,74)
(B)	2	5	6	32	35
(C)	–1	1	0	27	22

Furthermore, since the value of $n[\tau]$ (the number of leaf nodes in each layer) is adjusted adaptively according to the value of $x[\tau]$, our method can achieve better compression ratio than the original Golomb code, especially for the case where the probability of the data $y[\tau]$ varies with $x[\tau]$.

4. SIMULATIONS

(Example 1)

Here, the data x is the hamming distances between the dots in Fig. 3. Note that these dots are on the edge of some object. The coordinates and the hamming distance between the 9 dots are listed in Table 3. The value of y is

the number of pixels on the curve between two dots
– round(0.8 × the hamming distance). (37)

The value of y is also shown in Table 3. From statistics, we can conclude that

$$mean(|y|) \propto \sqrt{\text{hamming distance}} = \sqrt{x}. \quad (38)$$

Therefore, in (15), $f(x) = \sqrt{x}$.

Table 4: The value of c (defined in (31)) and the number of bits of the resulted codes for the example in Table 3

c	16	18	20	22	24	26
# of bits	48	47	45	47	47	49

Table 5: The value of $n[\tau]$ (the number of nodes in each layer) for the example in Table 3 when $p_1 = 0.9367$. Note that, when $x[\tau]$ is larger, $n[\tau]$ is also larger

τ	1	2	3	4	5	6	7	8	9
$n[\tau]$	2	5	10	11	2	3	3	7	8
$x[\tau]$	3	12	63	72	2	5	6	32	35

Then, we use the modified Golomb coding algorithm in Section 3 to encode y. In (16),

$$X_1 = \max\{\sqrt{x}\} = 8.4853. \quad (39)$$

Then, we use (17) to calculate $y_1[\tau]$ and then use the method in (31)-(36) to find the optimal value of p_1. From the methods in (35) and (36), we obtain that

$$p_1 \cong 0.9415, \quad c \cong 21.6990. \quad (40)$$

Therefore, we first try $c = 22$ in (31).
Then, we use (32) and find that the number of bits of the codes is 47 when $c = 22$. Then, we adjust the value of c around 22 and use (32) to calculation the number of bits of the resulted codes. We find that the optimal condition is

$$c = 20, \quad \text{i.e., } p_1 = 0.9367. \quad (41)$$

In this case, the number of bits of the resulted codes is 45. Moreover, since we require 5 extra bits to encode c, in sum, we require **50 bits** when using the proposed algorithm to encode y.

In comparison, when using the original Golomb code for this example, the number of bits of the resulted codes is **56 bits**. Therefore, our algorithm can indeed achieve shorter coding length than the original Golomb code.

When we code y without any special technique, since for a 100×100 image, the possible value of y in (37) can be from –60 to 100^2, thus, we require 14 bits to encode each value of $y[\tau]$ and hence 14×9 = **126 bits** in sum. When we use the Huffman coding algorithm, although in theory we just require 29 bits to encode the data, however, we need a table to record which code should be decoded as which data value. Since there are 8 different values for $y[\tau]$ and the value of $y[\tau]$ ranges from –60 to 100^2, the size of table is 14×8 = 112 bits. Thus, in fact, we require 29 + 112 = **141 bytes** to encode the data when using the Huffman coding algorithm. Thus, using the proposed modified Golomb codes is much more efficient than using the two algorithms.

We then show the explicit coding result of the example. First, we should encode c. Since we choose $c = 20$, which is the 18th value among the 32 possible values listed in (31), we encode c as '10010'.

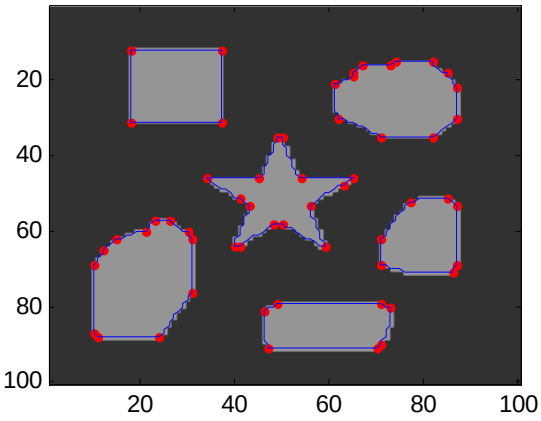

Fig. 4: An image with 58 dots on the edges of objects.

Then, from Steps 4 and 5, the value of $n[\tau]$ are calculated as in Table 5. Since $n[1] = 2$, we should substitute n by 2 and use Rules 1-4 in Section 2 to encode $|y[1]|$. From (8)-(12), $H = 1$ and $a = 0$. Then, from Rule 4, the code for $|y[1]| = 1$ is '10' (We use the method as in Table2). Since $y[1] \neq 0$, we should use an extra bit to code its sign. Thus, the complete code for $y[1]$ is '100'.

Then, we encode $y[2]$. Since $n[2] = 5$, from (8)-(12), $H = 3$ and $a = 3$. Then, from Rule 4, the code for $|y[2]| = 1$ is '111'. With the extra bit for sign, the complete code for $y[2]$ is '1111'. Similarly, the code for $y[3]$ is '010000'. The code for $y[4]$ is '01011100'. The full code of y is:

10010.100.1111.010000.01011100.101.0000.11.
10111100. 1101100, (42)

which has 50 bits and matches our previous prediction.

(Example 2)

Then, we perform another simulation. In Fig. 4, there are 58 dots on the edges. We also set x as the hamming distance and set y as the difference between the number of pixels on the curve between two dots and round(0.8x), as Example 1. When using the original Golomb coding algorithm, **195 bits** are required for encoding y. When using our proposed modified algorithm, only **182 bits** are required for encoding y. Therefore, the proposed coding algorithm is more efficient than the original Golomb coding algorithm.

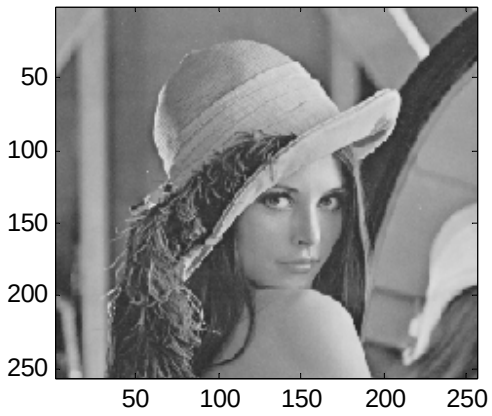

Fig. 5: Lena image

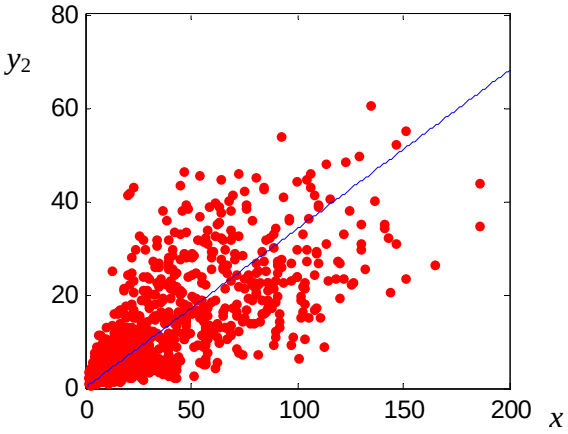

Fig. 6: The horizontal axis x is the average amplitude of the low frequency DCT coefficients of Lena image and the vertical axis y_2 is that of middle frequency DCT coefficients. The blue line is $y_2 = 0.34x$. This figure shows that the middle frequency DCT coefficients are highly correlated with the low frequency DCT coefficients.

(Example 3)

We then give another example. Fig. 5 is a 256×256 Lena image. Then we perform the 2-D DCT for each 8×8 block of Lena image. In Fig. 6, the horizontal axis is the average amplitude of the lower frequency DCT coefficients calculated from

$$x_{i,j} = \left\{ \left|C_{i,j}[0,1]\right| + \left|C_{i,j}[1,0]\right| + \left|C_{i,j}[2,0]\right| + \left|C_{i,j}[1,1]\right| + \left|C_{i,j}[0,2]\right| \right\} / 5 \quad , \tag{43}$$

where $C_{i,j}[m, n]$ means the DCT coefficients for the $(i, j)^{th}$ block. of Lena image. The vertical axis in Fig. 6 is the average amplitude of the middle frequency DCT coefficients:

$$\frac{1}{9}\left\{ \left|C[0,3]\right| + \left|C[1,2]\right| + \left|C[2,1]\right| + \left|C[3,0]\right| + \left|C[4,0]\right| + \left|C[3,1]\right| + \left|C[2,2]\right| + \left|C[1,3]\right| + \left|C[0,4]\right| \right\} \tag{44}$$

From Fig. 6, we can see that the amplitudes of the middle frequency DCT coefficients indeed has high correlation with the amplitude of the low frequency DCT coefficients. When the low frequency AC part has high energy, the middle frequency AC part usually has high energy, too.

Therefore, we can use the proposed modified Golomb coding algorithm together with the information of the low frequency DCT coefficients to code the middle frequency DCT coefficients of Lena image. Here, we set y as

$$y_{i,j}[m,n] = C_{i,j}[m,n] - round(0.34 X_{i,j}) . \tag{45}$$

$[m, n]$ = [0, 3], [1, 2], [2, 1], [3, 0], [4, 0], [3, 1], [2, 2], [1, 3], [0, 4],

i, j = 0, 1, 2, …., 31. (46)

The function $f(x)$ is $\sqrt{x}$, i.e.,

$$\left|y_{i,j}[m,n]\right| \propto x_{i,j} . \tag{47}$$

Then, we use our algorithm in Section 3 to encode y. The number of bits of the result is **49377 bits**. In comparison, when using the original Golomb code to encode y, the result has **53054 bits**. This again proves that our algorithm is more efficient than the original Golomb code and is useful in digital image processing.

5. CONCLUSIONS

In this paper, we use the concept of joint probability to improve the Golomb coding algorithm. As the Golomb code, our method also has the advantages of being suitable for the data with infinite possible values and no table is required for encoding. Furthermore, since our algorithm can use the information of another data to adjust the value of p and hence the coding tree, our method can achieve even shorter coding length than the original Golomb coding algorithm.

As the description at the beginning in Section 3, there are many conditions that a data has infinite number of possible values, the probability of the data is geometric-like, and the probability is dependent on another data. In these conditions, we can use the proposed algorithm together with the joint probability to improve the coding efficiency. Therefore, the proposed method will be useful in digital signal processing and digital image processing.

Furthermore, with the simulation in Example 3, it is possible to use the proposed coding scheme to further improve the efficiency of JPEG and MPEG using the high correlation between the low frequency DCT coefficients and the middle frequency DCT coefficients.

ACKNOWLEDGEMENT

The authors would like to thank the members of the NTU thesis defense committee for their constructive and valuable comments and suggestions. They also thank the support of the project 97-2221-E-002-112 by National Science Council of Taiwan.

REFERENCES

[1] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, 40(9): 1098-1101, 1952.

[2] D. E. Knuth, "Dynamic Huffman Coding," *Journal of Algorithms*, vol. 6, Issue 2, pp. 163-180, 1985.

[3] D J. S. Vitter, "Design and Analysis of Dynamic Huffman Codes," *Journal of ACM*, vol. 34, pp. 825-845, Oct. 1987.

[4] Steven Pigeon and Yoshua Bengio, "Memory Efficient and High-Speed Search Huffman Coding," *IEEE Trans. on Communications*, vol. 43, no. 10., pp. 2576-2581, 1995.

[5] M. Rabbani and P. W. Jones, *Digital Image Compression Techniques.* Bellingham, WA: SPIE Opt. Eng. Press, 1991.

[6] R. W. Hamming, *Coding and Information Theory* Englewood Cliffs, NJ: Prentice-Hall, 1980.

[7] S. Roman, *Coding and information Theory*. New York: Springer-Verlag, 1992.

[8] S. W. Golomb, "Run length encodings," *IEEE Trans. on Information Theory*, vol. IT12, pp. 399-401, 1966.

[9] R. G. Gallager and D. C. V. Voorhis, "Optimal source codes for geometrically distributed integer alphabets," *IEEE Trans. on Information Theory*, vol. IT21, pp. 228-230, 1975.

[10] R. F. Rice, *Some Practical Universal Noiseless Coding Techniques*, Jet Propulsion Laboratory, Pasadena, California, JPL Publication 79--22, Mar. 1979.

[11] G. Seroussi and M. J. Weinberger, "On adaptive strategies for an extended family of Golomb-type codes," *Proc. DCC'97*, pp. 131-140, 1997.

[12] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principle and standardization into JPEG-LS", *IEEE Trans. Image Proc.*, vol. 9, pp. 1309-1324, 2000.