KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS VOL. 10, NO. 11, Nov. 2016

# Improving JPEG-LS Performance Using Location Information

**Jae Hyeon Woo[1], Hyoung Joong Kim[1]**
[1]Graduate School of Information Security
Korea University Seoul 136-701, Korea
[1][e-mail: {bull0330, khj-}@korea.ac.kr]
*Corresponding author: Hyoung Joong Kim

*Received July 11, 2016; revised September 2, 2016; accepted October 2, 2016;*
*published November 30, 2016*

---

## Abstract

JPEG-LS is an international standard for lossless or near-lossless image-compression algorithms. In this paper, a simple method is proposed to improve the performance of the lossless JPEG-LS algorithm. With respect to JPEG-LS and its supplementary explanation, Golomb-Rice (GR) coding is mainly used for entropy coding, but it is not used for long codewords. The proposed method replaces a set of long codewords with a set of shorter location map information. This paper shows how efficiently the location map guarantees reversibility and enhances the compression rate in terms of performance. Experiments have also been conducted to verify the efficiency of the proposed method.

---

***Keywords:*** JPEG-LS, Lossless compression, Golomb code, Golomb-Rice, Location map

**http://dx.doi.org/10.3837/tiis.2016.11.019** **ISSN : 1976-7277**

## 1. Introduction

The JPEG-LS algorithm [1] is a lossless or near-lossless compression standard for still images. Most of the recent compression algorithms accepted the *Modeling and Coding* [2] concept to achieve an efficient compression. JPEG-LS uses the two-sided geometric distribution (TSGD) model [3] of the prediction error that is based on the median edge detector (MED) [4], and encodes this model by Golomb-Rice (GR) coding [5]–[6]. As a lossless compression algorithm, it performs quite effectively and is superior to most of the other algorithms including JPEG2000 [7]–[10]. A group of authors tried to improve the performance of JPEG-LS by introducing novel prediction methods. Baligar et al. [7] proposed a linear prediction method that minimizes the squared error and exploits quadtree coding. Bedi et al. [8] proposed a prediction method that detects the diagonal edges in addition to the vertical and horizontal edges. While the compression performances of both studies [7]-[8] exceed that of JPEG-LS, the computational complexities of their algorithms are far greater [9]. Kademi et al. [10] paradoxically showed the superiority of JPEG-LS. Kim et al. [11] proposed hierarchical average and copy prediction (HACP) scheme and showed the upper bound of significant bit truncatuon (SBT) coding. Combinations of multiple predictions [12]–[13] are also used to improve the compression rates. Masmoudi et al. [14] proposed a block-based lossless image compression for which finite-mixture models and adaptive arithmetic coding are used. Zhao et al. [15] and Starosolski [16] showed that the context-based adaptive lossless image codec (CALIC) [17] provides high compression rates within reasonable time frames, while JPEG-LS is effective enough and very fast. As a result, the JPEG-LS algorithm is still considered excellent in terms of the compression rate and the compression time.

Mobasseri et al. [18] proposed a method for the embedding of data into the JPEG bitstream, whereby Huffman-code mapping is used, and the data embedding is performed according to a reversible mapping of the unused codewords; notably, only a fraction of the JPEG codewords are actually used. Later, Qian and Zhang [19] improved their method, whereby the most important contribution of their methods from [18] and [19] is a reversible data hiding capability for the improvement of the compression rate; this reversibility is used in this paper for the replacement of long codewords. Ding et al. [20] proposed a modified Golomb coding for an asymmetric TSGD.

Even though JPEG-LS is almost perfect as a lossless algorithm, its performance can still be improved. The goal of this paper is the improvement of the JPEG-LS performance through a slight modification of the long codewords. JPEG-LS accepted the GR code as a main encoding alogrithm, but this GR code can sometimes produce excessively long codewords. A number of techniques can be used to remedy the long codeword artifact of the GR code; that is, the JPEG-LS algorithm modifies these long GR codewords into a fixed-length code (the authors call this modification JPEG-LS GR), while an additional improvement is proposed in this paper. The GR artifact can also be resolved by utilizing the location map. The location map is a tool for the book-keeping of the side information that assures the reversibility of data hiding schemes [21]–[23]. It is mostly used to avoid underflow, overflow or decoding errors at the decoding stage.

This paper shows that the location map can be used for lossless data compression; moreover, it shows that long codewords can be replaced with shorter codewords. The contribution of this paper consists of three methods that reduce the size of the symbol length in bits, as follows; replacing the fixed-length prefixes with location information, reducing the location map itself using a variable-length information, and reducing the suffix.

This paper is organized as follows. Section 2 briefly summarizes the concepts of GR coding and JPEG-LS GR coding. Section 3 revisits the concept of the location map, and includes an explanation of the application of location information for JPEG-LS GR coding; a simple method for the reduction of the location map size is also presented; along with a demonstration of improvements of the JPEG-LS GR method for specific cases. Section 4 presents an analysis of the experiment results, whereby the performances of the JPEG-LS GR method and the proposed method are compared. Section 5 concludes the paper.

## 2. Golomb-Rice coding vs. JPEG-LS coding

When an integer $N$ is divided by another integer $m$, a quotient $q$ and a remainder $r$ are derived so that the number $N$ can be represented as $N = q \cdot m + r$. As long as $m$ is given, the GR code in the JPEG-LS is represented by [unary quotient $q$] + [one bit for a separator "1"] + [binary remainder $r$]. JPEG-LS does not encode the pixel value itself, but encodes the *prediction error* through the GR coding. The integer notation is defined here as follows: $N_D$ is the digit expression (either in 8 bits or 16 bits), $N_B$ is the binary expression of $N_D$, and $N_U$ is the unary expression. For example, when $N_D$ is $3_{(10)}$ for an 8-bit depth image, $N_B$ is $00000011_{(2)}$ and $N_U$ is $000_{(1)}$, and they are the same representations. Note that $N_U$ has three zeros since $N_D$ is 3. Similarly, when a given prediction error in a decimal number is $N_D = 73_{(10)}$, its binary representation in 8-bit format is expressed as $N_B = 01001001_{(2)}$. Let a divisor be $m = 2^k$, for example, where $k = 2$, $q$ is the first $8 - k$ bits of $N_B$, which is $q = 010010_{(2)} = 18_{(10)} = 000000000000000000_{(1)}$; and $r$ is the last $k$ bits of $N_B$, which is $r = 01_{(2)}$. In this case, the corresponding GR code is represented by the following format:

$$N_{GR} = 000000000000000000|1|01_{(2)}$$

The output $N_{GR}$ is of a pseudo-binary form. Namely, it looks like a binary sequence, but it is actually a combination of binary symbols. The GR code starts with 18 leading zeros that are equivalent to the unary representation of the quotient. The first "1" that separates the quotient and the remainder with two red bars is a separator. The last part of "01" is just the remainder $r$ in binary format.

Another given number $73_{(10)}$ can be represented in a different form depending on the $k$ value. Let $k$ be 1 so that $q = 0100100_{(2)} = 36_{(10)} = 000000000000000000000000000000000000_{(1)}$ and $r = 1_{(2)}$. As a result, its GR code is obtained according to the following format:

$$N_{GR} = 000000000000000000000000000000000000|1|1_{(2)}$$

In this case, the total length of the GR code is 38 bits, which seems excessively long; therefore, this lengthy GR code $N_{GR}$ is not used whenever $q \geq 23_{(10)}$, but a shorter code $N_{LS}$ is used in a different format as follows:

$$N_{LS} = 00000000000000000000000|1|01001000_{(2)}$$

where the eight LSB bits are represented as $N_D - 1 = 72_{(10)} = 01001000_{(2)}$. Now, 32 bits (i.e., [23 bits of leading zeros] + [1-bit separator "1"] + [8-bits of $N_D - 1$]) are needed for $N_{LS}$ rather than 38 bits for $N_{GR}$. This representation mostly shortens the length of the GR code.

To achieve consistency with the GR code, 23 zeros are leading both the separator and the eight bits of $N_D -1$ itself for the pixels quantized with 8-bit depth; then, maximally 32 bits are needed in any case. A gain by the JPEG-LS GR code is obtained when the quotient is large enough, and the gain of the JPEG-LS GR code is six bits, which is six bits less than the GR code in this example.

Such a gain, however, does not always occur with the use of JPEG-LS GR coding. For example, a given number $47_{(10)}$ can be represented as follows:

$$N_B = 00101111_{(2)}$$

For $k = 1$, its GR code is represented with 23 leading zeros, as follows:

$$N_{GR} = 00000000000000000000000|1|1_{(2)}$$

However, the JPEG-LS GR code should be used whenever $q \geqq 23$. The JPEG-LS GR code is, therefore, represented as follows:

$$N_{LS} = 00000000000000000000000|1|00101110_{(2)}$$

It is obvious that the JPEG-LS GR code needs 32 bits while the GR code needs only 25 bits. Even though the JPEG-LS GR code is not always superior to the GR code, the former code is used in the JPEG-LS algorithm. The length of the JPEG-LS GR code that includes a prefix, a separator, and a suffix is always 32 bits. A combination of the [*prefix* and *separator*] of the JPEG-LS GR code is called the *replaceable JPEG-LS GR prefix* or *replaceable prefix* for short in this paper. This replaceable prefix consists of 24 bits that comprise 23 leading zeros plus the separator “1”. The 8-bit suffix is called *intact suffix* in this paper. If the prefixes are replaceable, their associated suffixes are intact suffixes.

Notably, most images are quantized with eight bits per pixel and eight bits per color (such as red, green, and blue); however, some accurate images are quantized with 16 bits per pixel and 16 bits per color. In this case, the JPEG-LS GR code consists of a 47-bit prefix, a 1-bit separator, and a 16-bit suffix; therefore, the replaceable JPEG-LS GR prefix is 48 bits for the 16-bit-depth images.

## 3. Proposed Method

Both the *run* and *regular* modes are used for JPEG-LS. The *run* mode encodes the run length, which is the count of the same continuous values. Either GR or JPEG-LS GR code is used for each codeword when the *regular* mode is utilized, and this *regular* mode is the sole focus of this paper.

In this section, three contributions are introduced. First, the replacement of the replaceable JPEG-LS GR prefix with location information can reduce the codeword size. Second, the size of the location information itself can be reduced. Third, the intact suffix code size can also be reduced.

### 3.1 Use of Location Information & Occurrence Gain

Location information is a kind of side information to guarantee reversibility. The encoder and decoder can perform the same job if they share the same location information. If the encoder modifies the pixel values, the modification can cause an overflow or underflow error. Obviously, in that case, the encoder skips the pixel to avoide the causing of an error, and the pixel should be marked in the location map. The decoder also skips the decoding when it encounters the marked position. A set of the marked position information is called the *location map*.

The first contribution of this paper is the replacement of the replaceable prefixes. This paper identifies the possibility that the JPEG-LS GR codeword can still be shortened using a simple method; that is, the location information is used instead of the replaceable JPEG-LS GR prefix. Even though the replaceable prefix is removed, the intact suffix is not altered. The position information of the replaceable prefix will be used here instead of the lengthy JPEG-LS GR code.

The location map consists of the simple (*row*, *col*) coordinates of such JPEG-LS GR code in this paper. The map of each *row* and *column* are of an 8-bit length in the 256 × 256 images, while a 9-bit length applies for the 512×512 images. For the 256×256 images, the coding gains for each replaceable prefix are therefore 8 ($\text{i. e.}, 24 - 2 \times 8$) bits for the 8-bit-pixel depth, and 32 ($\text{i. e.}, 48 - 2 \times 8$) bits for the 16-bit depth. For the 512×512 images, the coding gains for each prefix are 6 ($\text{i. e.}, 24 - 2 \times 9$) bits for the 8-bit depth and 30 bits for the 16-bit depth. This location information is subsequent to the total number information of these locations at the beginning of the bitstream.

**Table 1.** Example of the JPEG-LS codewords for a 512 × 512 image

| Position | $N_D$ | $k$ | Prefix | Separator | Suffix |
|---|---|---|---|---|---|
| (0, 0) | 33 | 2 | 00000000 | 1 | 01 |
| … | … | … | … | … | … |
| (197, 256) | 84 | 1 | 00000000000000000000000 | 1 | 01010011 |
| … | … | … | … | … | … |
| (339, 211) | 111 | 2 | 00000000000000000000000 | 1 | 01101110 |
| … | … | … | … | … | … |
| (480, 290) | 239 | 3 | 00000000000000000000000 | 1 | 11101110 |
| … | … | … | … | … | … |
| (480, 394) | 139 | 0 | 00000000000000000000000 | 1 | 10001010 |
| … | … | … | … | … | … |
| (511, 511) | … | … | … | … | … |

At a position (*row*, *col*) = (339, 211) of the 512×512 image in **Table 1**, for example, the JPEG-LS GR code is $00000000000000000000000|1|01101110_{(2)}$, and a replaceable prefix $000000000000000000000001_{(2)}$ is encountered. This prefix is removed during the proposed encoding and only the intact suffix of $01101110_{(2)}$ (i.e., $N_D - 1$) remains. After removing the replaceable prefix, its position information of $101010011011010011_{(2)}$, which is equivalent to (339, 211), is recorded at the header of the bitstream and a coding gain of six bits is obtained. The replaceable prefixes are underlined in **Table 1**.

Let $H$ and $W$ be the height and width of the image, respectively. As long as the value $\lceil \log_2 H \rceil + \lceil \log_2 W \rceil$ is smaller than 24 for the 8-bit depth, or less than 48 for the 16-bit depth, the replacement with the location map results in a coding gain; such a gain is called *occurrence* *gain* in this paper. Of course, the number of replaceable prefixes should be delivered to the decoder as a part of the side information.

### 3.2 Map Gain

The second contribution is a possible reduction of the location information itself. The position information of the first replaceable prefix is marked as it is because the reference information regarding the position is not available; however, for the rest of the replaceable prefixes, the information regarding the previous position is available. The available information can be exploited for the attainment of further gain. In the example in **Table 1**, four replaceable prefixes appear where the positions are (197, 256), (339, 211), (480, 290) and (480, 394); here, the row positions {197, 339, 480, and 480} are in an ascending order. For the first position (197, 256) for a 512 × 512 image, 18 (i.e., 9 plus 9) bits are needed to record $011000101100000000_{(2)}$. The next position (339, 211) is $101010011011010011_{(2)}$; here, because of the fact that $339 \geq 256$, it is obvious that 339 is located in the second half (bottom part) of the image, and the subsequent row positions that are larger than 256 indicate that they are located in the second half. As a result, these rows are all marked as $1xxxxxxxx_{(2)}$. Because both the encoder and the decoder recognize this fact, it is not necessary to encode the first bit of "1", and only the 8-bit $xxxxxxxx_{(2)}$ is needed without the "1" in this case; therefore, only the 17 (i.e., 8 plus 9) bits of $\underline{11100000}100100010_{(2)}$ are recorded as the location information of ($\underline{480}$, 290).

If a replaceable prefix is in the second half of the image, one bit of gain can be attained in the recording of the next piece of location information; similarly, if an *i-th* replaceable prefix is placed in the fourth quarter, two bits of gain can be attained in the recording of the position of the (*i*+1)-*th* prefix.

The column information in the same row can also be reduced by using a similar method because the columns of the same row are of an ascending order; therefore, the location information of (480, $\underline{394}$) in **Table 1** is $11100000\underline{10001010}_{(2)}$ in 16 (i.e., 8 plus 8) bits according to a referencing of the position information of (480, $\underline{290}$).

This type of gain is called *map gain* in this paper. Of course, the replaceable prefix positions occur randomly; therefore, such a gain is not always achieved.

### 3.3 K3 (or K10) Gain

The third contribution is a size reduction of the intact suffix. When the *replaceable prefix* is encountered for a pixel that is quantized by eight bits, the manifestation of the leading 23 zeros means that this sample is encoded by the JPEG-LS GR method, whereby the value of the quotient $q \geq 23$ and the suffix value is $N_D - 1$; that is, the replaceable prefix does not occur whenever $q < 23$. By utilizing this fact, the divisor information $k$ can be exploited to reduce the size of the intact suffix. With respect to $k = 3$ regarding the 8-bit pixel values, for example, the value range of the first five bits of any $q$ is from 0 to 31; as a result, $N_B$ ranges from $00000xxx_{(2)}$ to $11111xxx_{(2)}$, where $xxx_{(2)}$ stands for three bits of "don't care" binary numbers (i.e., remainders).

Regarding the quotient part, numbers from 0 to 22 do not appear when $q \geq 23$, and, in this case, numbers from 23 to 31 appear when $k$ is 3; that is, only nine numbers are actually used. Thus, quotient values from 23 to 31 can therefore be reassigned to other values from 0 to 8, respectively. **Table 2** shows the reassigned number representation.

A modified quotient of 0 is equivalent to the unmodified quotient value of 23, a modified quotient of 1 is equivalent to the unmodified quotient value 24, and so on. Only four bits are consequently needed to represent the nine numbers from 0 to 8 ($0000_{(2)}$ to $1000_{(2)}$), rather than the use of five bits to represent numbers from 23 to 31 ($10111_{(2)}$ to $11111_{(2)}$); moreover, one more bit can be saved here. Three bits are assigned to represent the numbers from 0 to 6, and four bits are assigned to represent 7 and 8.

**Table 2.** *q* representation method with *K3 gain* in 8-bit image

| Unmodified *q* | Bit expression | Modified *q* | Recorded bits |
|---|---|---|---|
| 23 | 10111 | 0 | 000 |
| 24 | 11000 | 1 | 001 |
| 25 | 11001 | 2 | 010 |
| 26 | 11010 | 3 | 011 |
| 27 | 11011 | 4 | 100 |
| 28 | 11100 | 5 | 101 |
| 29 | 11101 | 6 | 110 |
| 30 | 11110 | 7 | 1110 |
| 31 | 11111 | 8 | 1111 |

$N_B$ is used rather than $N_B$ —1 to record the suffix in this case. The intact suffix of position (480, 290), for example, is $11101110_{(2)}$ in **Table 1**. The original $N_B = 11101110_{(2)} + 1 = 11101111_{(2)}$ can be split into two parts. If $k$ is 3, then the unmodified-quotient part is $11101_{(2)}$ and the remainder part is $111_{(2)}$. The unmodified-quotient part of $11101_{(2)}$ (i.e., $29_{(10)}$) is modified to $110_{(2)}$ (i.e., $6_{(10)}$) by the subtraction of 23. Through this modification, the intact suffix is changed to $110111_{(2)}$, which requires six bits rather than eight bits, and saves two bits. This type of gain is called *K3 gain* in this paper.

The same number of bits can be saved for the values that are quantized by a 16-bit depth with $k = 10$. Then, the $N_B$ ranges from $000000xxxxxxxxxx_{(2)}$ to $111111xxxxxxxxxx_{(2)}$ in normal case. Similarly, only 17 numbers from $101111xxxxxxxxxx_{(2)}$ to $111111xxxxxxxxxx_{(2)}$ are actually replaceable since the replaceable prefix does not occur whenever $q < 47$. As a result, the assignment of six bits is not required, but five or four bits are enough. Two bits can be saved by assigning short binary numbers from $0000_{(2)}$ to $1110_{(2)}$ (for the unmodified $47_{(10)}$ to $61_{(10)}$, respectively) or one bit can be saved by assinging $11110_{(2)}$ and $11111_{(2)}$ (for the unmodified $62_{(10)}$ and $63_{(10)\ )}$, respectively) (See **Table 3**). This gain is called *K10 gain* in this paper.

**Table 3.** *q* representation method with *K10 gain* in 16-bit image

| Unmodified *q* | Bit expression | Modified *q* | Recorded bits |
|---|---|---|---|
| 47 | 101111 | 0 | 0000 |
| 48 | 110000 | 1 | 0001 |
| 49 | 110001 | 2 | 0010 |
| … | … | … | … |
| 59 | 111011 | 12 | 1100 |
| 60 | 111100 | 13 | 1101 |
| 61 | 111101 | 14 | 1110 |
| 62 | 111110 | 15 | 11110 |
| 63 | 111111 | 16 | 11111 |

## 4. Experiment Results

The performance of the proposed method was compared with that of the JPEG-LS algorithm (see **Tables 4 to 8**). A variety of uncompressed images ([24]-[25], **Figs. 1 and 2**) are used for the comparison; here, a few images do not have a replaceable prefix, while most of the images have a sufficient number of replaceable prefixes.

In the experiments, 12 gray-scale images of a 256×256 size and 12 gray-scale 512×512 images, all quantized by eight bits, are used; in addition, 8 color images of varying sizes that have been quantized by eight bits are used (see **Fig. 1**). The images quantized by 16 bits are shown in **Fig. 2**, and include 15 gray-scale images and 14 color images.

Before the compressed bitstream is recorded, the overhead representing the number of location information is first recorded.

For the *lena* (256×256) image with an 8-bit pixel depth, the replaceable prefixes occur 20 times, and 160 bits (i.e., 20×8) can be reduced purely by the *occurrence gain*. For the recording of the 20 occurrences, *five bits* are needed to represent the number $20_{(10)}$ or $10100_{(2)}$, and this number of bits, *five*, is expressed as $0101_{(2)}$. The entirety of the overhead information is therefore expressed with nine bits as $0101|10100_{(2)}$ and this overhead header is preserved at the beginning of the binary stream for this image.

Among the nine bits in the overhead header, the first four bits are used to represent the following number of bits representing the number of replaceable prefixes; therefore, the leading four bits can represent numbers from 0 ($0000_{(2)}$) to 15 ($1111_{(2)}$). The maximum number of overhead bits is 19 (four bits of leading "1111" plus 15 bits to represent the number of replaceable prefixes).

The exact output bitstream size of JPEG-LS depends on two kinds of zero padding. The first type pads a number of zeros at the end to make the bitstream size a multiple of eight; for example, when the actual output bitstream is $00000000110_{(2)}$ (11 bits), five zeros are padded to make 16 bits in total and the output is then $00000000110\mathbf{00000}_{(2)}$. The second type of zero padding occurs whenever the consecutive eight bits are all "1". For the bit stream $11111111110_{(2)}$, where the first eight bits are all "1", one zero is inserted as a delimiter and four zeros are padded to make $11111111\mathbf{0}110\mathbf{0000}_{(2)}$. Such a zero-padding rule is briefly stated to calculate the exact number of the output bits of the JPEG-LS algorithm.

For the *lena* image (256×256), the original GR and JPEG-LS GR output stream size without any padding is 299,901 bits, and 123 bits are added through the padding process to make 300,024 bits; notably, since 300,024 is a multiple of eight, the first type of zero padding is not necessary. In the pure output bit string of 299,901 bits, 123 cases of $11111111_{(2)}$ occurs, and the same number of zero bits are inserted as a second type of zero padding.

In the proposed method of this paper, nine bits of overhead header (i.e., "$0101|10100_{(2)}$") are added to the pure JPEG-LS output of 299,901 bits; alternatively, 160 bits are reduced by the occurrence gain due to the 20 replaceable prefixes. In addition, six bits are reduced by the map gain; moreover, two bits are reduced by the K3 gain. The modified JPEG-LS output size is therefore 299,742 bits (i.e., $299{,}901 + 9 - 160 - 6 - 2$). Regarding the 299,742 bits, since there are 122 consecutive eight "1"s, 122 zeros are inserted. As a result, the final output size of the modified JPEG-LS is 299,864 ($\mathrm{i.e.,}\ 299{,}742 + 122$) bits, which is the final result of the *lena* image because it is a multiple of eight, and this is slightly more favorable than that of the original JPEG-LS. In **Table 4, 5 and 7**, the numbers in the upper rows of the *occurrence gain* and the *K3 gain* (or *K10 gain*) are shown in bits, while the numbers in parenthesis in the lower rows indicate the number of occurrences of such gain.

1. bird (256*256) 2. bridge (256*256) 3. cameraman (256*256) 4. circles (256*256)
5. crosses (256*256) 6. goldhill (256*256) 7. horiz (256*256) 8. lena (256*256)
9. montage (256*256) 10. slope (256*256) 11. squares (256*256) 12. Text (256*256)

1. barb (512*512) 2. boat (512*512) 3. France (496*672) 4. frog (498*621)
5. goldhill (512*512) 6. lena (512*512) 7. library (352*464) 8. mandrill (512*512)
9. mountain (480*640) 10. peppers (512*512) 11. washsat (512*512) 12. zelda (512*512)

1. clegg (880*814) 2. frymire (1105*1118) 3. lena (512*512) 4. monarch (512*768)
5. peppers (512*512) 6. sail (512*768) 7. serrano (794*629) 8. tulips (512*768)

**Fig. 1.** 8-bit small size images. (http://links.uwaterloo.ca/Repository.html)

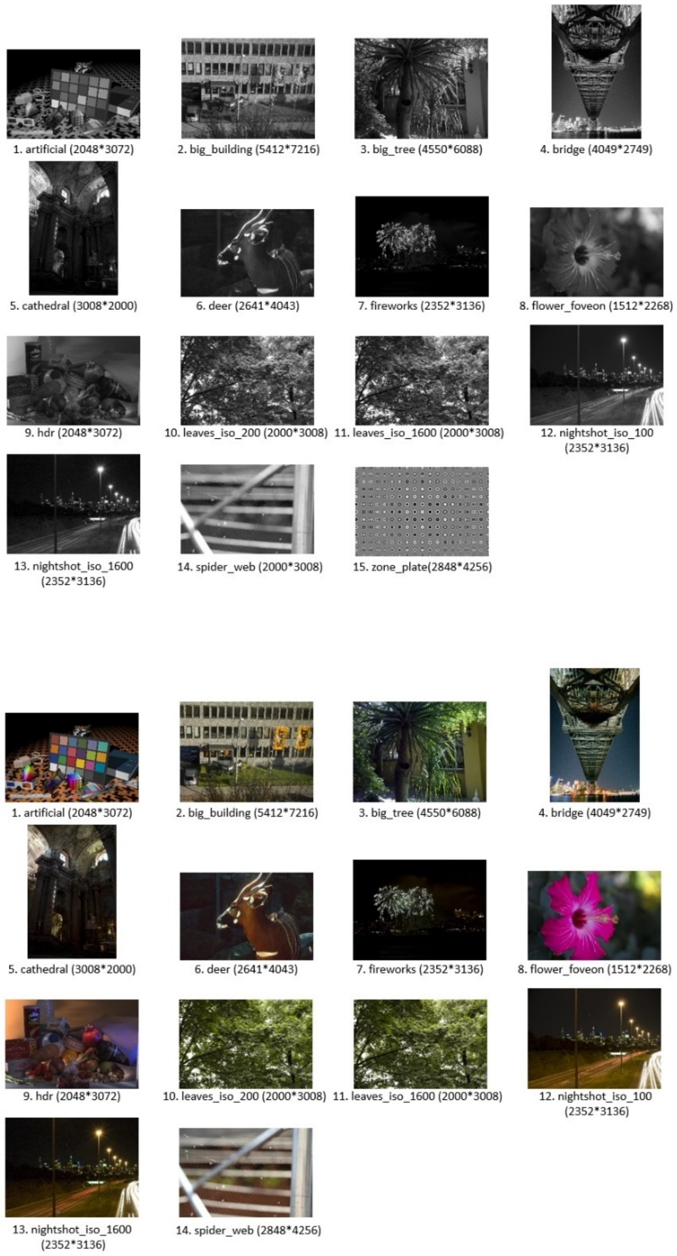

**Fig. 2.** 16-bit large size images. (http://imagecompression.info/test_images)

For a few images, replaceable prefixes are not encountered; however, some images have a large number of replaceable prefixes. The *artificial* image (2,048×3,072) that has been quantized by 16 bits (see **Table 7**) has 452,775 bits of occurrence gain meaning that the artificial image has 18,111 replaceable prefixes. Since the image size is 2,048×3,072, 23 bits are needed to mark the positions; therefore, the apparent occurrence gain is 25 bits (i.e., $48 - 23$, where the replaceable prefix takes 48 bits for the 16-bit depth images). In this case, 19 bits are required as an overhead header to represent the 18,111 replaceable prefixes (i.e., the mandatory four bits to represent 15, and the next 15 bits to represent 18,111); therefore, the overhead header is represented as $1111|100011010111111_{(2)}$, which needs 19 bits in total. Notably, the *artificial* image is a synthetic image and not a natural one.

For the *cameraman* (256×256) image, 107 replaceable prefixes are encountered, and this means that 856 bits can be reduced; moreover, 32 bits are reduced due to the map gain, and 12 bits are reduced due to the K3 gain. As a result, the output size of the JPEG-LS and the proposed method are 282,488 bits and 281,576 bits, respectively. For the *France* image (496×672), 289 replaceable prefixes are found, while 112 bits and 33 bits are reduced as the map and K3 gains, respectively.

The *clegg* color image (880×814) in **Table 6** has 7,872 bits of occurrence gain for the red channel, 6,832 bits for the green channel, and 8,456 bits for the blue channel, since the image has 1,968, 1,708, and 2,114 replaceable prefixes, respectively. For the red color, 7,872 bits, 1,669 bits, and 536 bits are the occurrence gain, map gain, and K3 gain, respectively. For the images quantized with 16 bits, the K10 gain is a counterpart of the K3 gain, as shown in **Tables 7** and **8**.

**Table 4.** Result of 8-bit Gray Images I.

| Image (*H*×*W*) | occurrence gain | map gain | K3 gain | output bits | | |
|---|---|---|---|---|---|---|
| | | | | Original | JPEG-LS | Proposed |
| bird<br>(256×256) | 8 bits<br>(1) | 0 bits | 0 bits | 524,288 | 227,272 | 227,264 |
| bridge<br>(256×256) | 152 bits<br>(19) | 0 bits | 0 bits | 524,288 | 379,264 | 379,128 |
| cameraman<br>(256×256) | 856 bits<br>(107) | 32 bits | 22 bits<br>(12) | 524,288 | 282,488 | 281,576 |
| circles<br>(256×256) | 24 bits<br>(3) | 0 bits | 0 bits | 524,288 | 9,784 | 9,768 |
| corsses<br>(256×256) | 48 bits<br>(6) | 2 bits | 0 bits | 524,288 | 25,048 | 25,008 |
| goldhill<br>(256×256) | 64 bits<br>(8) | 2 bits | 2 bits<br>(1) | 524,288 | 345,896 | 345,840 |
| horiz<br>(256×256) | 56 bits<br>(7) | 4 bits | 0 bits | 524,288 | 5,928 | 5,880 |
| lena<br>(256×256) | 160 bits<br>(20) | 6 bits | 2 bits<br>(1) | 524,288 | 300,024 | 299,864 |
| montage<br>(256×256) | 504 bits<br>(63) | 34 bits | 6 bits<br>(3) | 524,288 | 178,240 | 177,712 |
| slope<br>(256×256) | 648 bits<br>(81) | 49 bits | 8 bits<br>(4) | 524,288 | 102,760 | 102,064 |
| squares<br>(256×256) | 24 bits<br>(3) | 0 bits | 0 bits | 524,288 | 4,840 | 4,824 |
| text<br>(256×256) | 184 bits<br>(23) | 11 bits | 0 bits | 524,288 | 106,728 | 106,536 |

**Table 5.** Result of 8-bit Gray Images II.

| Image ($H \times W$) | occurrence gain | map gain | K3 gain | output bits | | |
|---|---|---|---|---|---|---|
| | | | | Original | JPEG-LS | Proposed |
| barb<br>(512×512) | 132 bits<br>(22) | 11 bits | 0 bits | 2,097,152 | 1,240,584 | 1,240,448 |
| boat<br>(512×512) | 84 bits<br>(14) | 2 bits | 2 bits<br>(1) | 2,097,152 | 1,113,832 | 1,113,752 |
| France<br>(496×672) | 1,445 bits<br>(289) | 112 bits | 65 bits<br>(33) | 2,666,496 | 470,664 | 469,048 |
| frog<br>(498× 621 ) | 60 bits<br>(12) | 3 bits | 2 bits<br>(1) | 2,474,064 | 1,870,432 | 1,870,368 |
| goldhill<br>(512×512) | 24 bits<br>(4) | 1 bits | 0 bits | 2,097,152 | 1,234,912 | 1,234,896 |
| lena<br>(512×512) | 72 bits<br>(12) | 6 bits | 0 bits | 2,097,152 | 1,112,240 | 1,112,176 |
| library<br>(352×464) | 624 bits<br>(104) | 50 bits | 37 bits<br>(20) | 1,306,624 | 832,936 | 832,240 |
| mandrill<br>(512×512) | 492 bits<br>(82) | 0 bits | 6 bits<br>(3) | 2,097,152 | 1,582,216 | 1,581,728 |
| mountain<br>(480×640) | 885 bits<br>(177) | 99 bits | 68 bits<br>(39) | 2,457,600 | 1,972,616 | 1,971,592 |
| peppers<br>(512×512) | 126 bits<br>(21) | 6 bits | 2 bits<br>(1) | 2,097,152 | 1,176,472 | 1,176,344 |
| washsat<br>(512×512) | 36 bits<br>(6) | 1 bits | 0 bits | 2,097,152 | 1,082,256 | 1,082,232 |
| zelda<br>(512×512) | 6 bits<br>(1) | 0 bits | 0 bits | 2,097,152 | 1,049,760 | 1,049,752 |

**Table 6.** Result of 8-bit Color Images.

| Image ($H \times W$) | C | occurrence gain | map gain | K3 gain | output bits | | |
|---|---|---|---|---|---|---|---|
| | | | | | Original | JPEG-LS | Proposed |
| clegg<br>(880×814) | R | 7,872 | 1,669 | 965 | 5,730,560 | 1,703,848 | 1,693,352 |
| | G | 6,832 | 1,045 | 515 | 5,730,560 | 1,794,640 | 1,786,264 |
| | B | 8,456 | 1,479 | 988 | 5,730,560 | 1,783,608 | 1,772,808 |
| frymire<br>(1105×1118) | R | 4,268 | 64 | 1,185 | 9,883,120 | 2,431,024 | 2,425,608 |
| | G | 4,224 | 172 | 993 | 9,883,120 | 2,609,960 | 2,604,648 |
| | B | 3,818 | 108 | 906 | 9,883,120 | 2,459,416 | 2,454,648 |
| lena<br>(512×512) | R | 18 | 0 | 0 | 2,097,152 | 1,060,280 | 1,060,272 |
| | G | 102 | 6 | 0 | 2,097,152 | 1,207,568 | 1,207,472 |
| | B | 192 | 0 | 8 | 2,097,152 | 1,284,120 | 1,283,944 |
| monarch<br>(512×768) | R | 530 | 61 | 6 | 3,145,728 | 1,469,288 | 1,468,712 |
| | G | 445 | 45 | 14 | 3,145,728 | 1,469,056 | 1,468,576 |
| | B | 415 | 38 | 12 | 3,145,728 | 1,510,704 | 1,510,240 |
| peppers<br>(512×512) | R | 114 | 7 | 1 | 2,097,152 | 1,034,904 | 1,034,784 |
| | G | 108 | 16 | 2 | 2,097,152 | 1,019,296 | 1,019,176 |
| | B | 150 | 16 | 2 | 2,097,152 | 1,029,288 | 1,029,120 |
| sail<br>(512×768) | R | 110 | 3 | 4 | 3,145,728 | 2,060,152 | 2,060,040 |
| | G | 100 | 0 | 0 | 3,145,728 | 2,042,296 | 2,042,208 |
| | B | 95 | 5 | 0 | 3,145,728 | 2,050,112 | 2,050,024 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| serrano<br>(794×629) | R | 2,972 | 247 | 235 | 3,995,408 | 826,480 | 823,072 |
| | G | 3,104 | 319 | 304 | 3,995,408 | 867,944 | 864,240 |
| | B | 2,204 | 254 | 269 | 3,995,408 | 665,408 | 662,680 |
| tulips<br>(512×768) | R | 155 | 9 | 0 | 3,145,728 | 1,578,680 | 1,578,520 |
| | G | 125 | 8 | 0 | 3,145,728 | 1,656,520 | 1,656,400 |
| | B | 160 | 15 | 2 | 3,145,728 | 1,701,432 | 1,701,272 |

**Table 7.** Result of 16-bit Gray Images.

| Image<br>($H\times W$) | occurrence gain | map gain | K10 gain | output bits | | |
|---|---|---|---|---|---|---|
| | | | | Original | JPEG-LS | Proposed |
| artificial<br>(2048×3072) | 452,775<br>(18,111) | 22,765 | 6<br>(3) | 100,663,296 | 27,167,728 | 26,692,336 |
| big_building<br>(5412×7216) | 5,060<br>(230) | 0 | 0 | 624,847,872 | 451,339,088 | 451,334,028 |
| big_tree<br>(4550×6088) | 198<br>(9) | 18 | 0 | 443,206,400 | 324,584,184 | 324,583,986 |
| bridge<br>(4049×2749) | 120<br>(5) | 12 | 0 | 178,091,216 | 135,643,808 | 135,643,688 |
| cathedral<br>(3008×2000) | 1,225<br>(49) | 84 | 2<br>(1) | 96,256,000 | 69,526,592 | 69,525,296 |
| deer<br>(2641×4043) | 240<br>(10) | 0 | 0 | 170,841,008 | 136,309,472 | 136,309,240 |
| fireworks<br>(2352×3136) | 4,464<br>(186) | 101 | 0 | 118,013,952 | 60,353,296 | 60,348,744 |
| flower_foveon<br>(1512×2268) | 1,375<br>(55) | 20 | 0 | 54,867,456 | 33,892,768 | 33,891,376 |
| hdr<br>(2048×3072) | 550<br>(22) | 28 | 0 | 100,663,296 | 62,900,288 | 62,899,720 |
| leaves_iso_200<br>(2000×3008) | 3,350<br>(134) | 125 | 4<br>(2) | 96,256,000 | 70,434,464 | 70,431,000 |
| leaves_iso_1600<br>(2000×3008) | 3,075<br>(123) | 168 | 4<br>(2) | 96,256,000 | 74,955,104 | 74,951,864 |
| nightshot_iso_100<br>(2352×3136) | 648<br>(27) | 9 | 0 | 118,013,952 | 72,684,152 | 72,683,504 |
| nightshot_iso_1600<br>(2352×3136) | 2,808<br>(117) | 60 | 0 | 118,013,952 | 88,672,408 | 88,669,552 |
| spider_web<br>(2848×4256) | 575<br>(25) | 12 | 0 | 193,937,408 | 111,630,984 | 111,630,400 |
| zone_plate<br>(2000×3000) | 1,675<br>(67) | 30 | 56<br>(30) | 96,000,000 | 95,673,640 | 95,671,872 |

**Table 8.** Result of 16-bit Color Images.

| Image ($H \times W$) | C | occurrence gain | map gain | K10 gain | output bits | | |
|---|---|---|---|---|---|---|---|
| | | | | | Original | JPEG-LS | Proposed |
| artificial (2048×3072) | R | 440,400 | 22,646 | 14 | 100,663,296 | 27,622,520 | 27,159,592 |
| | G | 455,325 | 22,522 | 5 | 100,663,296 | 26,097,064 | 25,619,352 |
| | B | 392,650 | 23,018 | 6 | 100,663,296 | 22,666,056 | 22,250,512 |
| big_building (5412×7216) | R | 1,210 | 0 | 0 | 624,847,872 | 471,972,608 | 471,971,398 |
| | G | 7,568 | 2 | 0 | 624,847,872 | 446,166,904 | 446,159,336 |
| | B | 990 | 6 | 2 | 624,847,872 | 457,438,424 | 457,437,434 |
| big_tree (4550×6088) | R | 198 | 0 | 2 | 443,206,400 | 338,047,656 | 338,047,458 |
| | G | 1,254 | 6 | 0 | 443,206,400 | 316,958,648 | 316,957,394 |
| | B | 176 | 0 | 7 | 443,206,400 | 356,661,248 | 356,661,072 |
| bridge (4049×2749) | R | 1,056 | 37 | 0 | 178,091,216 | 137,191,176 | 137,190,088 |
| | G | 48 | 3 | 0 | 178,091,216 | 139,260,144 | 139,260,096 |
| | B | 0 | 0 | 0 | 178,091,216 | 141,927,112 | 141,927,120 |
| cathedral (3008×2000) | R | 2,125 | 172 | 6 | 96,256,000 | 72,049,384 | 72,047,096 |
| | G | 2,450 | 204 | 0 | 96,256,000 | 65,756,096 | 65,753,448 |
| | B | 3,025 | 352 | 0 | 96,256,000 | 69,606,856 | 69,603,488 |
| deer (2641×4043) | R | 72 | 0 | 0 | 170,841,008 | 136,085,256 | 136,085,192 |
| | G | 0 | 0 | 0 | 170,841,008 | 143,742,960 | 143,742,968 |
| | B | 0 | 0 | 0 | 170,841,008 | 147,174,488 | 147,174,496 |
| fireworks (2352×3136) | R | 3,864 | 63 | 0 | 118,013,952 | 65,342,088 | 65,338,176 |
| | G | 2,424 | 0 | 4 | 118,013,952 | 50,603,008 | 50,600,600 |
| | B | 7,104 | 27 | 0 | 118,013,952 | 30,611,824 | 30,604,712 |
| flower_foveon (1512×2268) | R | 50 | 0 | 0 | 54,867,456 | 34,634,848 | 34,634,800 |
| | G | 1,025 | 18 | 0 | 54,867,456 | 25,886,872 | 25,885,840 |
| | B | 0 | 0 | 0 | 54,867,456 | 34,599,072 | 34,599,072 |
| hdr (2048×3072) | R | 300 | 22 | 0 | 100,663,296 | 64,317,448 | 64,317,136 |
| | G | 250 | 15 | 0 | 100,663,296 | 64,468,512 | 64,468,256 |
| | B | 175 | 11 | 2 | 100,663,296 | 66,914,480 | 66,914,288 |
| leaves_iso_200 (2000×3008) | R | 2,550 | 118 | 15 | 96,256,000 | 71,013,008 | 71,010,336 |
| | G | 5,150 | 251 | 2 | 96,256,000 | 70,067,784 | 70,062,400 |
| | B | 1,700 | 58 | 8 | 96,256,000 | 69,239,256 | 69,237,488 |
| leaves_iso_1600 (2000×3008) | R | 425 | 11 | 9 | 96,256,000 | 76,392,784 | 76,392,352 |
| | G | 4,025 | 235 | 6 | 96,256,000 | 74,091,952 | 74,087,696 |
| | B | 450 | 13 | 2 | 96,256,000 | 73,353,248 | 73,352,800 |
| nightshot_iso_100 (2352×3136) | R | 1,080 | 9 | 0 | 118,013,952 | 74,671,984 | 74,670,904 |
| | G | 1,056 | 18 | 0 | 118,013,952 | 71,826,064 | 71,825,000 |
| | B | 1,944 | 48 | 0 | 118,013,952 | 60,798,592 | 60,796,608 |
| nightshot_iso_1600 (2352×3136) | R | 72 | 0 | 0 | 118,013,952 | 90,512,384 | 90,512,320 |
| | G | 2,928 | 9 | 0 | 118,013,952 | 86,662,496 | 86,659,568 |
| | B | 2,712 | 3 | 0 | 118,013,952 | 80,695,226 | 80,668,410 |
| spider_web (2848×4256) | R | 115 | 0 | 0 | 193,937,408 | 114,490,864 | 114,490,752 |
| | G | 529 | 8 | 0 | 193,937,408 | 111,667,024 | 111,666,496 |
| | B | 207 | 2 | 0 | 193,937,408 | 114,798,856 | 114,798,656 |

## 5. Conclusion and Discussion

In this paper, a new lossless compression method is proposed. When the codeword length is longer than the position information, the codeword can be replaced with a shorter piece of position information. JPEG-LS reference software uses either 24-bit or 48-bit replaceable prefixes to avoid the long codewords of the GR code; however, it is obvious that they are still long, and that replaceable prefixes of less than 24 bits or 48 bits can be used to enhance the compression rate. Future work includes the employment of various lengths of the replaceable prefix to maximize the performance. In this paper, the potential of the location information is manifested resulting in additional gains.

## 6. Acknowledgement

This work was supported in part by a National Research Foundation of Korea (NRF) grant funded by the Korean Government (MEST) (NRF-2015R1A2A2A01004587), and Korea University.

## References

[1] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, Aug. 2000. Article (CrossRef Link)

[2] J. Rissanen and G. G. Langdon, Jr, "Universal modeling and coding," *IEEE Transactions on Information Theory*, vol. 27, pp. 12–23, Jan. 1981. Article (CrossRef Link)

[3] N. Merhav, G. Seroussi, and M.J. Weinberger, "Optimal Prefix Codes for Sources with Two-Sided Geometric Distributions," *IEEE Transactions on Information Theory*, vol. 46, no. 1, pp. 120-135, Jan. 2000. Article (CrossRef Link)

[4] S. A. Martucci, "Reversible compression of HDTV images using median adaptive prediction and arithmetic coding," in *Proc. of IEEE International Symposium on Circuits and Systems*, pp.1310-1313, 1990. Article (CrossRef Link)

[5] S. W. Golomb, "Run Length Encodings," *IEEE Transactions on Information Theory*, vol. 12, pp. 399-401, 1966. Article (CrossRef Link)

[6] R. F. Rice, "Practical Universal Noiseless Coding," in *Proc. of SPIE 0207, Applications of Digital Image Processing III*, 247, 1979. Article (CrossRef Link)

[7] V. P. Baligar, L. M. Patnaik, and G. R. Nagabhushana, "High compression and low order linear predictor for lossless coding of grayscale images," *Image and Vision Computing*, vol. 21, pp. 543–550, 2003. Article (CrossRef Link)

[8] S. Bedi, E. A. Edirisinghe, and G. Grecos, "Improvements to the JPEG-LS prediction scheme," *Image and Vision Computing*, vol. 22, pp. 9–14, Sep. 2004. Article (CrossRef Link)

[9] K. Horvath, H. Stögner, and G. Weinhandel, "Experimental study on lossless compression of biometric iris data," in *Proc. of the 7th International Symposium on Image and Signal Processing and Analysis*, pp. 379–384, Sep. 2011. Article (CrossRef Link)

[10] A. Khademi and S Krishnan, "Comparison of JPEG 2000 and other lossless compression schemes for digital mammograms," in *Proc. of 27th Annual International Conference of the Engineering in Medicine and Biology Society*, pp. 3771–3774, Jan. 2006. Article (CrossRef Link)

[11] J. Kim and C. M. Kyung, "A lossless embedded compression using significant bit truncation for HD video coding," *IEEE Transactions on Circuits and Systems for video technology*, vol. 20, no. 6, Jun. 2010. Article (CrossRef Link)

[12] G. Deng, H. Ye, and L. Cahill, "Adaptive techniques for lossless data compression," in *Proc. of the Australia and New Zealand Conference on Intelligent Information Systems*, pp. 345–350, 2001. Article (CrossRef Link)

[13] A. Martchenko and G. Deng, “Bayesian predictor combination for lossless image compression,” *IEEE Transactions on Image Processing*, vol.22, pp.5263–5270, 2013. Article (CrossRef Link)

[14] A. Masmoudi, W. Puech, and A. Masmoudi, “An improved lossless image compression based arithmetic coding using mixture of non-parametric distributions,” *Multimedia Tools and Applications*, vol. 74, no. 23, pp. 10605–10619, 2015. Article (CrossRef Link)

[15] S. Zhao, Y. Xu, H. Li, and H. Yang, “A comparison of lossless compression methods for palmprint images,” *Journal of Software*, vol. 7, no. 3, pp.594–598, Mar. 2012. Article (CrossRef Link)

[16] T. Starosolski, “Simple fast and adaptive lossless image compression algorithm,” *Software: Practice and Experience*, vol. 37, no. 1, pp. 65–91, Jan. 2007. Article (CrossRef Link)

[17] X. Wu and N. Memon, “Context-based, adaptive, lossless image codec,” *IEEE Transactions on Communications*, vol. 45, no. 4, pp. 437–444, Apr. 1997. Article (CrossRef Link)

[18] B. G. Mobasseri, R. J. Berger, M. P. Marcinak, and Y. J. NaikRaikar, “Data embedding in JPEG bitstream by code mapping,” *IEEE Transactions on Image Processing*, vol. 19, no. 4, pp. 958–966, Apr. 2010. Article (CrossRef Link)

[19] Z. Qian and X. Zhang, “Lossless data hiding in JPEG bitstream,” *Journal of Systems and Software*, vol. 85, no. 2, pp. 309–313, Feb. 2012. Article (CrossRef Link)

[20] J. J. Ding, W. Y. Wei, and G. C. Pan, “Modified Golomb coding algorithm for asymmetric two-sided geometric distribution data,” in *Proc. of The 20th European Signal Processing Conference*, pp.1548–1552, 2012. Article (CrossRef Link)

[21] J. Tian, “Reversible data embedding using a difference expansion,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 8, pp. 890–896, Aug. 2003. Article (CrossRef Link)

[22] Z. Ni, Y.-Q. Shi, and N. Ansari, “Reversible data hiding,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 3, pp. 354–362, Mar. 2006. Article (CrossRef Link)

[23] H .J. Kim, V. Sachnev, Y. Q. Shi, J. Nam, and H. G. Choo, “A novel difference expansion transform for reversible data embedding,” *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 3, pp. 1147–1156, Sep. 2008. Article (CrossRef Link)

[24] http://links.uwaterloo.ca/Repository.html

[25] http://imagecompression.info/test_images

[26] http://www.stat.columbia.edu/~jakulin/jpeg-ls/mirror.htm

[27] http://kr.mathworks.com/matlabcentral/fileexchange/53039-jpegls-codec

**Jae Hyeon Woo** received a B.S. degree in Mechanical Engineering and a M.S. degree in Electrical and Computer Engineering from Seoul National University, Seoul, Korea, in 1996, 2002, respectively. He joined Multimedia Security Laboratory at the Center of Information Security and Technology (CIST), Graduate School of Information Management and Security, Korea University, Seoul, Korea in 2007, where he is currently pursuing Ph.D. His research interests include multimedia security, reversible and robust watermarking, steganography, phychology, and big-data.

**Hyoung Joong Kim** received his B.S., M.S., and Ph.D. degrees from Seoul National University, Seoul, Korea, in 1978, 1986, and 1989,respectively. He joined the faculty of the Department of Control and Instrumentation Engineering, Kangwon National University, Korea, in 1989. He is currently a Professor of the Graduate School of Information Management and Security, Korea University, Korea since 2006. His research interests include parallel and distributed computing, multimedia computing, multimedia security, and big-data.