Journal of Computer Science and Cybernetics, V.36, N.2 (2020), 139–158
DOI 10.15625/1813-9663/36/2/14084

AN EFFECTIVE REVERSIBLE DATA HIDING METHOD BASED ON PIXEL-VALUE-ORDERING

NGUYEN KIM SAO[1,*], NGUYEN NGOC HOA[2], PHAM VAN AT[1]

[1]*Faculty of Information Technology, University of Transport and Communications*
[2]*Faculty of Information Technology, VNU University of Engineering and Technology*

Abstract. This paper presents a new effective reversible data hiding method based on pixel-value-ordering (iGePVO-K) which is improvement of a recent GePVO-K method that recently is considered as a PVO-used method having highest embedding capacity. In comparison with GePVO-K method, iGePVO-K has the following advantages. First, the embedding capacity of the new method is higher than that of GePVO-K method by using data embedding formulas reasonably and reducing the location map size. Second, for embedding data, in the new method, each pixel value is modified at most by one, while in GePVO-K method, each pixel value may be modified by two. In fact, in the GePVO-K method, the largest pixels are modified by two for embedding bits 1 and by one for bits 0. This is also true for the smallest pixels. Meanwhile, in the proposed method, the largest pixels are modified by one for embedding bits 1 and are unchanged if embedding bits 0. Therefore, the stego-image quality in proposed method is better than that in GePVO-K method. Theoretical analysis and experiment results show that the proposed method has higher embedding capacity and better stego image quality than GePVO-K method.

Keywords. Reversible data hiding; Pixel-value-ordering; Pixel prediction.

1. INTRODUCTION

Data hiding is a technique for embedding data in the digital image (also in digital audio, video, etc). In traditional data hiding techniques [2, 5, 15, 18], only hidden data are extracted. However, in some applications such as medical and military images, restoring the original image is needed [17]. So, new data hiding methods, called reversible (or lossless) are developed. Reversible data hiding (RDH) not only can extract hidden data but also restore original image.

The first RDH proposed by Macq [9] is performed based on modulo addition 256. The disadvantage of this method is that the stego image quality is very low. Shortly, Fridrich and et al. [4] proposed a method which uses lossless compress to embed data in the LSB bits (least significant bit). This method has low embedding capacity. Since then, there are many proposed RDH methods, such as histogram shifting (HS) [10, 16, 19], difference expansion (DE) [6], prediction error expansion(PEE) [8, 7] and combining techniques [19].

Recently, pixel value ordering (PVO) based techniques attract interest of many researchers [6, 7, 11, 12, 13, 14, 20]. In these methods the original image is divided into non-overlapping blocks. In each bock, pixel values are sorted in ascending order and largest/smallest pixels are used for embedding.

*Corresponding author.
E-mail addresses: saonkoliver@utc.edu.vn (N.K.Sao); hoa.nguyen@vnu.edu.vn (N.N.Hoa) phamvanat83@gmail.com (P.V.At).

It is noted that, in PVO based methods such as [7, 11, 13], at most one bit can embed in largest/smallest pixel of each block, while GePVO-K [6] can embed more than one bit in largest/smallest pixels. So, GePVO-K has embedding capacity lager than other PVO based RDH methods.

In this paper, we propose a new method (called as **iGePVO-K**) that is improvement of GePVO-K. Compared with GePVO-K, our method has embedding capacity larger and stego image quality better.

The rest of this paper is organized as follows. Section 2 presents some related works. Section 3 describes our proposed method, and the experimental results are presented in Section 4. Finally, a conclusion is made at the end of the paper.

2. RELATED WORKS

2.1. PVO

PVO is proposed by Li et al. [7] in 2013, since then, many reversible data hiding methods based on PVO are proposed. In PVO, the host image is divided into non-overlapped blocks. For a given block having n pixel values $(x_1, x_2, \ldots, x_n)$, it is sorted in ascending order to get $(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$, where $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ is a unique one-to-one mapping such that

a) $x_{\sigma(1)} \le x_{\sigma(2)} \le \ldots \le x_{\sigma(n)}$.

b) If $i < j$ and $x_i = x_j$ then $\sigma(i) < \sigma(j)$.

Then the greatest value $x_{\sigma(n)}$ is predicted by the second greatest value $x_{\sigma(n-1)}$ and the smallest value $x_{\sigma(1)}$ is predicted by second smallest value $x_{\sigma(2)}$.

PVO is considered in two cases $d_{\max}$ and $d_{\min}$, where $d_{\max} = x_{\sigma(n)} - x_{\sigma(n-1)}$ and $d_{\min} = x_{\sigma(1)} - x_{\sigma(2)}$.

Embedding is performed as follows

$$d'_{\max} = \begin{cases} d_{\max}, & \text{if} \quad d_{\max} = 0 \\ d_{\max} + b, & \text{if} \quad d_{\max} = 1 \\ d_{\max} + 1, & \text{if} \quad d_{\max} > 1, \end{cases} \tag{1}$$

and

$$d'_{\min} = \begin{cases} d_{\min}, & \text{if} \quad d_{\min} = 0 \\ d_{\min} - b, & \text{if} \quad d_{\min} = -1 \\ d_{\min} - 1, & \text{if} \quad d_{\min} < -1. \end{cases} \tag{2}$$

Where b is embedded bit. Then, the stego pixel values are

$$x'_{\sigma(n)} = x_{\sigma(n-1)} + d'_{\max} \tag{3}$$

for the largest side, and

$$x'_{\sigma(1)} = x_{\sigma(2)} + d'_{\min} \tag{4}$$

for the smallest side.

Because the order of the pixel values remains unchanged after embedding the data, the hidden data can be extracted from the largest value and smallest value pixels according to reverse process of the embedding procedure. At the same time, the original value pixels are restored.

2.2. IPVO

In PVO method, differences $d_{\max}$ (or $d_{\min}$) equal 0 are not used to embed one bit. On the contrary, in IPVO [13] differences $d_{\max}$ or $d_{\min}$ equal 0 are still used for embedding. So IPVO has embedding capacity higher. Like the PVO method, each block $(x_1, x_2, \ldots, x_n)$ is sorted in ascending order to get $(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$. Then $d_{\max}$ is computed as follows

$$d_{\max} = x_u - x_v,$$

where

$$\begin{cases} u = \min(\sigma(n),\ \sigma(n-1)), \\ v = \max(\sigma(n),\ \sigma(n-1)). \end{cases}$$

Then a bit b is embedded in $x_{\sigma(n)}$ according to formulas

$$x'_{\sigma(n)} = \begin{cases} x_{\sigma(n)} + b & \text{if } \ d_{\max} = 0 \text{ or } d_{\max} = 1, \\ x_{\sigma(n)} + 1 & \text{if } \ d_{\max} < 0 \text{ or } d_{\max} > 1. \end{cases}$$

Similarly, $d_{\min}$ are calculated $d_{\min} = x_s - x_t$, where

$$\begin{cases} s = \min(\sigma(1), \sigma(2)), \\ t = \max(\sigma(1), \sigma(2)). \end{cases}$$

Then a bit b is embedded in $x_{\sigma(1)}$ as follows

$$x'_{\sigma(1)} = \begin{cases} x_{\sigma(1)} - b & \text{if } \ d_{\min} = 0 \ \text{ or } \ d_{\min} = 1, \\ x_{\sigma(1)} - 1 & \text{if } \ d_{\min} < 0 \ \text{ or } \ d_{\min} > 1. \end{cases}$$

Extracting hidden data and restoring original image are carried out similarly as in PVO method by exploiting the invariant for the order of pixel values after embedding data.

2.3. PVO-K

It is noted that if vector $(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ has k largest values or l smallest values with $k > 1$ (and $l > 1$), then in PVO method, there is not any bit can embedded. These cases are treated in PVO-K method [11] as follows.

First compute $d_{\max} = x_{\sigma(n)} - x_{\sigma(n-k)}$, $d_{\min} = x_{\sigma(1)} - x_{\sigma(l+1)}$. Then embedding one bit in k largest values and one bit in l smallest values are performed as follows

$$x'_{\sigma(i)} = \begin{cases} x_{\sigma(i)} - p & \text{if } \ 1 \le i \le l \ \text{ and } \ d_{\min} = -1 \\ x_{\sigma(i)} - 1 & \text{if } \ 1 \le i \le l \ \text{ and } \ d_{\min} < -1 \\ x_{\sigma(i)} + q & \text{if } \ n-k+1 \le i \le n \ \text{ and } \ d_{\max} = 1 \\ x_{\sigma(i)} + 1 & \text{if } \ n-k+1 \le i \le n \ \text{ and } \ d_{\max} > 1 \\ x_{\sigma(i)} & \text{otherwise,} \end{cases} \tag{5}$$

where p, q are two embedded bits.

2.4. GePVO-K

In PVO-K method, one bit is embedded in k largest pixels of a image block. GePVO-K [6] is a generation of PVO-K to embed k bits in k these pixels.

GePVO-K divides blocks of size $m \times n,\ X = (x_1, x_2, \ldots, x_{m\times n})$ into three types.

(a) Block having at least one pixel value equal $0, 1, 254$ or 255 will not be used for embedding data, because it may cause under/over flow. The position number of block in location map is set as 2 ($LM(X) = 2$).

(b) Block with all pixel values equal (flat block): $x_1 = x_2 = \ldots = x_{m\times n} = \alpha$ with α different from $0, 1, 254, 255$ will be used to embed $m \times n - 1$ bits. The position number of block is set as 1 ($LM(X) = 1$).

(c) Remaining blocks (rough block) will be used to embed data. The position number of block is set as 0 ($LM(X) = 0$).

It is noted that position number of each block is two binary bits, so location map is a binary sequence having the length equal twice number of blocks.

Next, the authors deal with each block depending on it's position number in map:

Case 1: If the position number of the block in the location map is $LM(X) = 2$, the block is not used to embed data and it is skipped.

Case 2: If the position number of block $LM(X) = 1$, i.e, all pixel values are equal in the block X, keep the first pixel value unchanged and then embed the data in the remaining pixels as follows

$$x'_i = \begin{cases} x_i & \text{if} \quad i = 1 \\ x_i + b_{i-1} & \text{if} \quad i = 2, 3, \ldots, m \times n, \end{cases} \tag{6}$$

where $b_i,\ i = 1, \ldots, m \times n - 1$ are embedded bits.

Case 3: If the position number of the block $LM(X) = 0$, X is sorted in ascending order to get $(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(m\times n)})$. Assume sorted block has k largest pixels and l second largest pixels. The difference $d_{\max} = x_{\sigma(m\times n-k+1)} - x_{\sigma(m\times n-k)}$ is calculated and embedding is performed as follows:

Case 3.1: If $d_{\max} > 1$, no data is embedded, all largest pixel values are increased by one.

Case 3.2: If $d_{\max} = 1$, k bits data are embedded into the largest pixel values. All largest and second largest pixel values are increased by one, then embedding data in the largest pixel values as follows

$$x'_{\sigma(i)} = \begin{cases} x_{\sigma(i)} & \text{if} \quad 1 \le \sigma(i) < m \times n - k - l \\ x_{\sigma(i} + 1 & \text{if} \quad m \times n - k - l + 1 \le \sigma(i) \le m \times n - k \\ x_{\sigma(i)} + 1 + b_j & \text{if} \quad m \times n - k + 1 \le \sigma(i) \le m \times n,\ j = i - m \times n + k. \end{cases} \tag{7}$$

After embedding in the largest pixel values, the authors embed data in the smallest pixel values similarly.

3. PROPOSED METHOD

In the GePVO-K method, to embed a data bit, the value of pixel may be changed at most by two. This leads to great distortion in stego-image. Moreover, in GePVO-K, each blocks is marked by two binary bits in the location map. So, the location map is large, and this leads to reducing the embedding capacity as compression code of this map must be embedded along data in original image. These drawbacks will be overcome in new iGePVO-K method. In concrete, for embedding data in iGePVO-K, pixels only are modified at most by 1, and each block is marked by one binary bit in location map. Moreover, in some cases, where GePVO-K only can embed data in largest pixels, while iGePVO-K can embed data in both largest pixels and smallest pixels.

3.1. Embedding algorithm in a block

Consider a image block sized $m \times n$

$$X = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \dots & \vdots \\ x_{m1} & \dots & x_{m\times n} \end{bmatrix},$$

and its vector form is $x = (x_1, x_2, \dots, x_{m\times n})$. Below, x is called flat if its pixels are equal $x_1 = x_2 = \dots = x_{m\times n}$. A block that is not flat is called rough.

The algorithm divides each block into 3 cases for embedding data bits.

Case 1. Rough block contains 0/255

This block is not used for embedding, it is ignored, because embedding in this block can cause overflow or underflow.

Case 2. Flat block

It means all pixel values in the block are equal.

In this case, the first pixel x_1 is unchanged, other pixels ($x_i,\ i = 2, \dots, m \times n$) are modified to embed $(m \times n - 1)$ bits ($b_j,\ j = 1, 2, \dots, m \times n - 1$) as follows

$$x'_i = \begin{cases} x_i + b_{i-1} & \text{if} \quad x_i \le 254,\ i = 2, 3, \dots m \times n \\ x_i - b_{i-1} & \text{if} \quad x_i = 255,\ i = 2, 3, \dots m \times n \\ x_i & \text{if} \quad i = 1. \end{cases}$$

Case 3. Rough block does not contain 0/255

Assume sorted block $x = (x_{\sigma(1)}, x_{\sigma(2)}, \dots, x_{\sigma(m\times n)})$ has $k1$ largest value pixels, $k2$ smallest value pixels, $l1$ second largest value pixels, and $l2$ second smallest value pixels.

Embedding data in the largest pixel values

First, we compute two difference

$$d_{\max} = x_{\sigma(m\times n-k1+1)} - x_{\sigma(m\times n-k1)},$$

$$d_{\max 2} = \begin{cases} x_{\sigma(m\times n-k1-l1+1)} - x_{\sigma(m\times n-k1-l1)} & \text{if the block has more than two distinct values} \\ 0, & \text{if the block has two distinct values.} \end{cases}$$

Case 3.1. $d_{\max} \ge 2$

This block is not used to embed, the $k1$ largest values are increased by 1.

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1,\ i = n \times m - k1 + 1, \dots, n \times m.$$

Case 3.2. $d_{\max} = 1$. In this case, $k1$ data bits will be embedded in $k1$ largest pixels.
Case 3.2.1. If all $k1$ embedded bits equal 1, all the largest pixel values are increased by one, so, stego pixels are

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1,\ i = n \times m - k1 + 1, \ldots, n \times m.$$

Case 3.2.2. If all $k1$ embedded bits are 0. This situation is divided into two cases:
Case 3.2.2.1. If $d_{\max 2} = 0$, largest pixel values are not changed

$$x'_{\sigma(i)} = x_{\sigma(i)},\ i = n \times m - k1 + 1, \ldots, n \times m.$$

It is noted that, this case can cause ambiguous with the case in flat blocks embedded data containing {0, 1}. At extracting side, to distinguish these two situations for receiver, we use a flag for each block. If block is flat, the flag is 1, if the block has $d_{\max 2}$ is 0, the flag is 0.

This flag will be embedded in previous block. The previous block, for example, can embed k bits, then $k - 1$ data bits and a flag bit will be embedded. In extracting, we will use this flag to determine the considered block is flat or not.
Case 3.2.2.2. If $d_{\max 2} > 0$, the largest and second largest values are increased by 1

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1 \quad \text{if} \quad i = m \times n - k1 - l1 + 1, \ldots, m \times n.$$

Case 3.2.3. If $k1$ embedded bits include 0 and 1. Adding $k1$ data bits into the largest pixel values

$$x'_{\sigma(i)} = x_{\sigma(i)} + b_j,\ i = n \times m - k1 + 1, \ldots, n \times m,\ j = i - m \times n + k1\ (j = 1, \ldots, k1).$$

It is noted that, if the block contains three consecutive distinct pixel values then our proposed method can embed in both largest values and smallest values, while GePVO-K only can embed in largest values. So iGePVO-K has the embedding capacity larger the GePVO-K.

Embedding data in the smallest pixel values

After embedding at largest side, we consider the smallest side as following. First, we compute two differences

$$d_{\min} = x_{\sigma(1)} - x_{\sigma(k2+1)},$$

$$d_{\min 2} = \begin{cases} x_{\sigma(k2+1)} - x_{\sigma(k2+l2+1)} & \text{if the block has more than two distinct values} \\ 0, & \text{if the block has two distinct values.} \end{cases}$$

Case 3.3. If $d_{\min} \leq -2$

This block is not used to embed, the $k2$ smallest values are decreased by 1

$$x'_i = x_i - 1,\ i = 1, \ldots, k2.$$

Case 3.4. If $d_{\min} = -1$. In this case, $k2$ data bits will be embedded in $k2$ smallest pixels.

Case 3.4.1. If all of $k2$ embedded bits are 1, the secret bits are embedded by reducing $k2$ smallest values by 1

$$x'_{\sigma(i)} = x_{\sigma(i)} - 1,\ i = 1, \ldots, k2.$$

Case 3.4.2. If all of $k2$ embedded bits are 0, this situation is divided into two cases:
Case 3.4.2.1. If $d_{\min 2} = 0$, smallest pixel values are unchanged

$$x'_{\sigma(i)} = x_{\sigma(i)},\ i = 1, \ldots, k2.$$

Case 3.4.2.2. If $d_{\min 2} < 0$, the smallest and second smallest values are decreased by 1

$$x'_{\sigma(i)} = x_{\sigma(i)} - 1 \ \text{ if } \ i = 1, \ldots, k2 + l2.$$

Case 3.4.3. If $k2$ embedded bits include 0 and 1, subtracting $k2$ data bits from the smallest pixel values

$$x'_{\sigma(i)} = x_{\sigma(i)} - b_i,\ i = 1, \ldots, k2.$$

3.2. Extracting algorithm in a block

With the stego block $x' = (x'_1, x'_2, \ldots, x'_{m\times n})$ and a bit flag extracted from the previous block in case of the flat block or the block having two distinct values. This algorithm will extract the secret bits $b_j,\ j = 1, 2, \ldots$ and restore the original pixels.

Extracting and restoring original block are performed based on the location map as follows.

Case 1. $LM(X) = 1$ (Rough block contain 0/255) This block does not carry any data bit (see 3.1.1, case 1) do nothing.

Case 2. $LM(X) = 0$ and the block have only one distinct value.

All stego pixel values in the block are the same, this block carries $m \times n - 1$ bits 0. So, extracting and restoring are follows

$$b_j = 0,\ j = 1, 2, \ldots, m \times n - 1,$$
$$x_i = x'_i,\ i = 1, 2, \ldots, m \times n.$$

Case 3. $LM(X) = 0$ and the block have two distinct values. This situation is divided into three cases.

Case 3.1. The difference between two distinct values is 1. This case comes from two situations: the flat blocks are embedded by bits 0,1 or the rough blocks having two consecutive distinct values are embedded by the all bit 0 in the largest pixels. To deal this case, we have to use the flag bit recorded in previous block:

If flag bit is 1: The host block is flat, extracting and restoring are performed as follows

$$b_i = \begin{cases} x'_{i+1} - x'_1 & \text{if} \quad x'_1 \le 254,\ i = 1, \ldots, m \times n - 1 \\ x'_1 - x'_{i+1} & \text{if} \quad x_1 = 255,\ i = 1, \ldots, m \times n - 1, \end{cases}$$

$$x_i = x'_1,\ i = 1, 2, \ldots, m \times n.$$

If flag bit is 0: The host block are embedded all bit 0, so extracting and restoring are carried out as

$$b_j = 0,\ j = 1 \dots k1,$$
$$x_{\sigma(i)} = x'_{\sigma(i)},\ i = m \times n - k1 + 1, \dots, m \times n.$$

Case 3.2. The difference between two distinct values is 2. In this case, the all $k1$ largest pixel values of the host block are embedded all bit 1. So

$$b_j = 1,\ j = 1, \dots, k1,$$
$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = n \times m - k1 + 1, \dots, n \times m.$$

Case 3.3. The difference between two distinct values greater than 2. There is no data extracted, all largest values is decreased by 1 to restore the original image

$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = n \times m - k1 + 1, \dots, n \times m.$$

Case 4. $LM(X) = 0$ and the block have more than two distinct values.

Assume that, one sorted block in stego image $(x'_{\sigma(1)}, x'_{\sigma(2)}, \dots, x'_{\sigma(m\times n)})$ has $k1$ largest pixel values, $l1$ second largest pixel values, $k2$ smallest pixel values, $l2$ second smallest pixel values. Extracting hidden bits b_i and restoring the original pixels x_i will as follows.

Extracting and restoring in the largest pixel values:

First, we calculate two differences

$$d'_{\max} = x'_{\sigma(m\times n-k1+1)} - x'_{\sigma(m\times n-k1)},$$
$$d'_{\max 2} = x'_{\sigma(m\times n-k1-l1+1)} - x'_{\sigma(m\times n-k1-l1)}.$$

Case 4.1. If $d'_{\max} \geq 3$, there is not any bit extracted and pixels x_i are restored as

$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = n \times m - k1 + 1, \dots, n \times m.$$

Case 4.2. $d'_{\max} = 2$,

$$b_j = 1,\ j = 1 \dots k1,$$
$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = n \times m - k1 + 1, \dots, n \times m.$$

Case 4.3. $d'_{\max} = 1$, this situation is divided two cases:
Case 4.3.1. $d'_{\max 2} \geq 2$,

$$b_j = 0,\ j = 1, \dots, k1,$$
$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = n \times m - k1 - l1 + 1, \dots, n \times m.$$

Case 4.3.2. If $d'_{\max 2} = 1$

$$b_j = x'_{\sigma(j)} - x'_{\sigma(m\times n-k1)},\ j = m \times n - k1 - l1 + 1 \dots m \times n,$$
$$x_{\sigma(i)} = x'_{\sigma(m\times n-k1)},\ i = m \times n - k1 - l1 + 1, \dots, m \times n.$$

Extracting and restoring in the smallest pixel values:

First, we calculate two differences

$$d'_{\min} = x'_{\sigma(1)} - x'_{\sigma(k2+1)},$$
$$d'_{\min 2} = x'_{\sigma(k2+l2)} - x'_{\sigma(k2+l2+1)}.$$

Case 4.4. $d'_{\min} \le -3$, there is not any bit extracted, and pixels x_i are restored

$$x_{\sigma(i)} = x'_{\sigma(i)} + 1,\ i = 1, \dots, k2.$$

Case 4.5. If $d'_{\min} = -2$

$$b_j = 1,\ j = 1 \dots k2,$$
$$x_{\sigma(i)} = x'_{\sigma(i)} + 1,\ i = 1, \dots, k2.$$

Case 4.6. If $d'_{\min} = -1$, this situation is divided into two cases.
Case 4.6.1. If $d'_{\min 2} \le -2$

$$b_j = 0,\ j = 1, \dots, k2,$$
$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = 1, \dots, k2 + l2.$$

Case 4.6.2. If $d'_{\min 2} = -1$

$$b_j = x'_{\sigma(k2+1)} - x'_{\sigma(i)},\ j = 1 \dots k2 + l2,$$
$$x_{\sigma(i)} = x'_{\sigma(k2+1)},\ i = 1, \dots, k2 + l2.$$

Example

To illustrate the embedding and extracting algorithm, we give 4 examples in Figures 1, 2, 3, 4: Embedding in the block has only $d_{\max}$, embedding in the block has all cases, the block can only be embedded in smallest pixels, and the last is example for extracting.

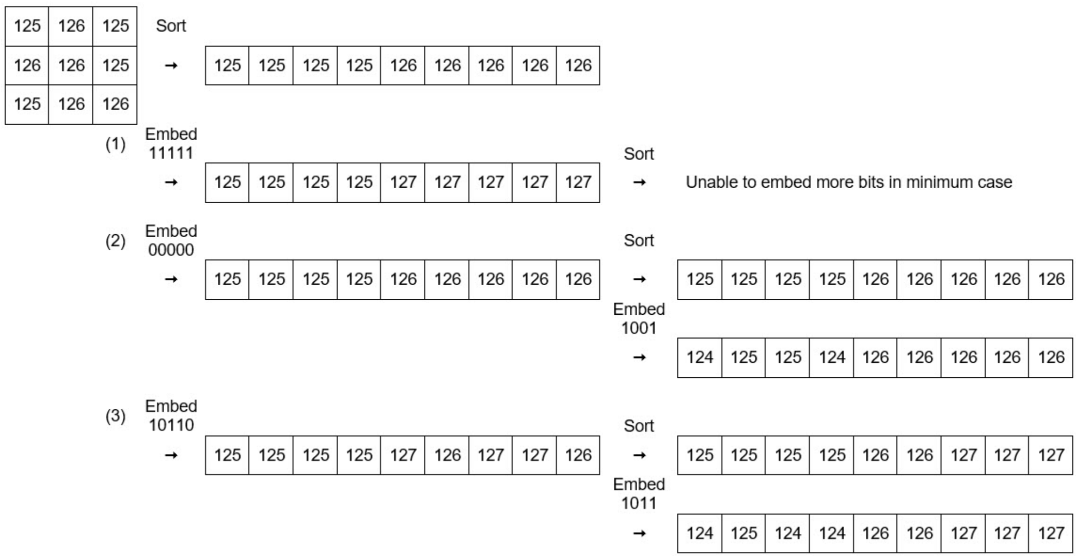


Figure 1. Embedding in block which has not $d_{\max 2}$

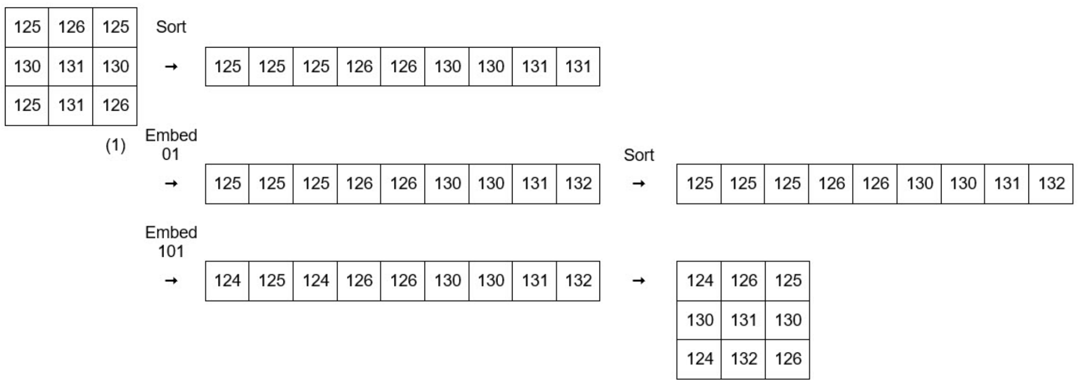


Figure 2. Embedding in block which has $d_{\max 2}$

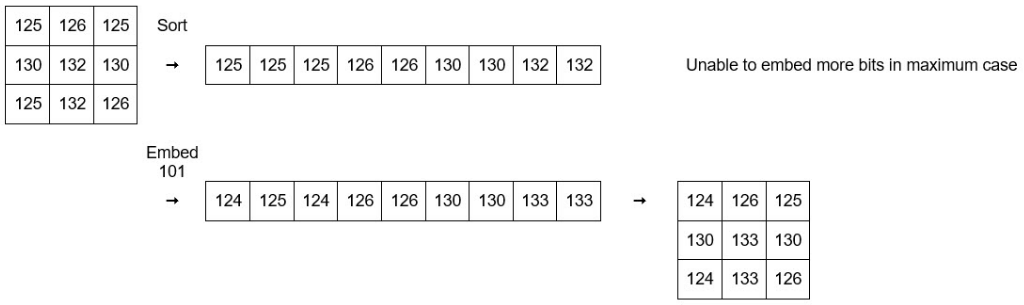
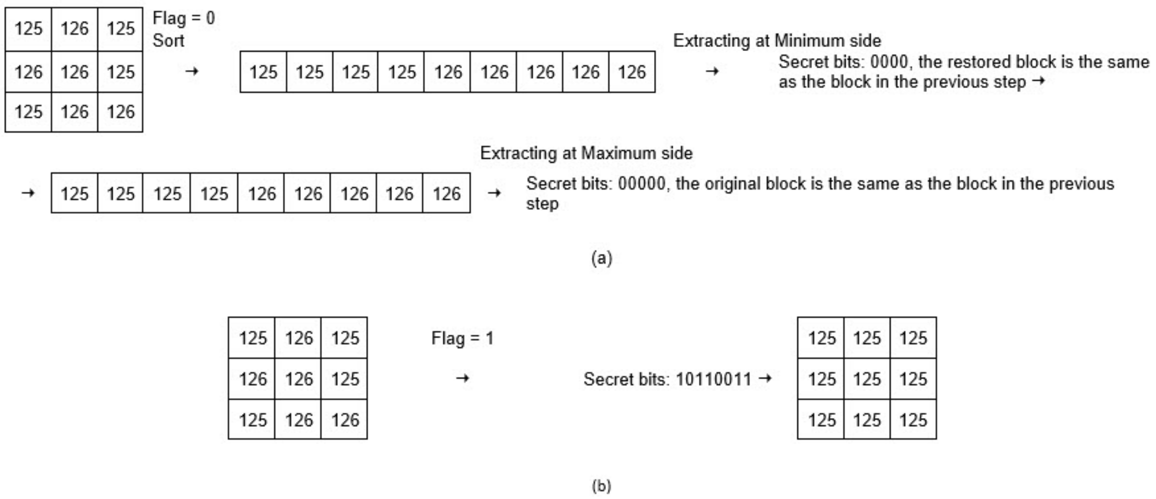


Figure 3. Embedding in block which is unable to embed in largest pixels



Flag = 0
Sort
Extracting at Minimum side
Secret bits: 0000, the restored block is the same as the block in the previous step
Extracting at Maximum side
Secret bits: 00000, the original block is the same as the block in the previous step
(a)
Flag = 1
Secret bits: 10110011
(b)

Figure 4. Extracting with flag

Figure 1 illustrates the embedding algorithm in the case when the block has two distinct values. This block is used to embed the array of bits 0, the array of bits 1 and the array containing bits 0,1 at both largest and smallest sides. Figure 2 illustrates the case block has more than two distinct values. Figure 3 considers the block which can only embed in the smallest side.

3.3. Location map and flag

At first, as same as PVO, IPVO, PVOK methods, the original image is divided into un-overlapped blocks of size $m \times n$.

Assume that, a block is transformed to a sequence $(x_1, x_2, \ldots, x_{m\times n})$. Then this sequence is sorted in ascending order to get $(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(m\times n)})$.

It is noted that a rough block which has pixel values equal 0 or 255 can cause under/over flow (after embedding pixel values can go out segment $[0; 255]$). So, this block will not be used for embedding. A binary location will be used to distinguish two type of blocks: Position number of under/over flow block in location map is set by 1 and position number of other blocks is set by 0

$$LM(X) = \begin{cases} 1 & \text{if } (x_{\sigma(1)} = 0 \quad \text{or} \quad x_{\sigma(m\times n)} = 255) \quad \text{and} \quad (x_{\sigma(1)} < x_{\sigma(m\times n)}) \\ 0 & \text{otherwise.} \end{cases}$$

It can be seen that, if using only the location number of blocks, the receiver can not distinguish two the following cases:
(a) A flat block $x_1 = \ldots = x_{m\times n} = \alpha$ (with $\alpha \leq 253$) is used to embed at least one bit 1. For example, block $X = (235, 235, 235, 235)$ after embedding three bits 110 will become $X' = (235, 236, 236, 235)$.
(b) A rough block having two consecutive distinct pixel values. For example, block $X = (235, 235, 236, 236)$ after embedding two bits 00 will be unchanged $X = (235, 235, 236, 236)$.

Both these two blocks have the same location number equal 0, but their stego blocks are the same. There are two consecutive distinct pixel values smaller 255. To solve this problem, we use a flag bit, flag 0 for the flat block and flag 1 for the remaining block.

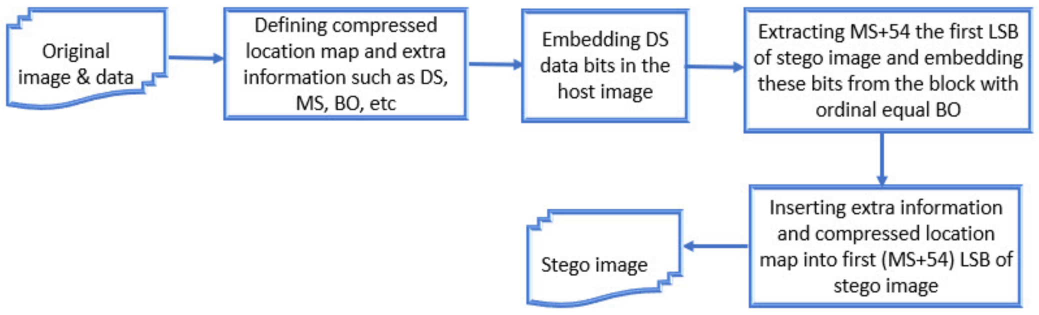
Flag of each block will be embedded in the previous block. Thus, in the previous block, the number of data bits decrease one bit, the flag bit is added.

3.4. Embedding and extracting procedure

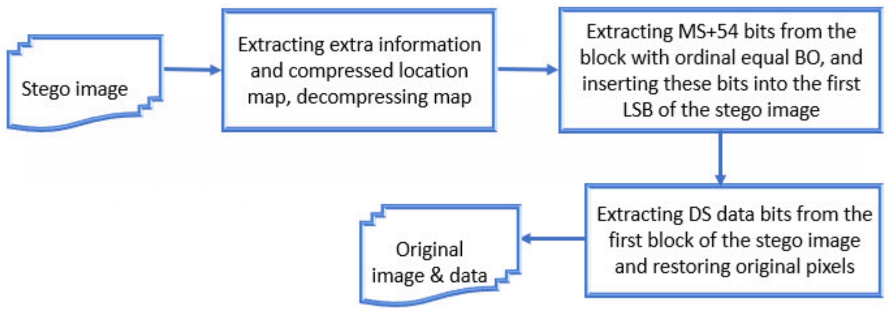
In this section, we will describe our proposed reversible data hiding method, called iGePVO-K, including the embedding and extracting procedures in more detail. Its flowchart is shown in Figure 5.

3.4.1. Embedding procedure

Embedding the data bits is performed in the following steps.
Step 1. Divide the host image into non-overlapped blocks of size $m \times n$. After that, visit each block to set up a location map (LM) as in Subsection 3.3. Then, LM is compressed using arithmetic coding. In addition, we compute some extra information (BO, MS, etc) as shown in Table 1.
Step 2. Visit from the first block, for block B if $LM(B) = 1$, skip (do nothing). In the case $LM(B) = 0$, first we determine flag bit for next block. If next block needs to use flag, we


Original image & data
Defining compressed location map and extra information such as DS, MS, BO, etc
Embedding DS data bits in the host image
Extracting MS+54 the first LSB of stego image and embedding these bits from the block with ordinal equal BO
Inserting extra information and compressed location map into first (MS+54) LSB of stego image
Stego image

(a) The flowchart of embedding procedure


Stego image
Extracting extra information and compressed location map, decompressing map
Extracting MS+54 bits from the block with ordinal equal BO, and inserting these bits into the first LSB of the stego image
Extracting DS data bits from the first block of the stego image and restoring original pixels
Original image & data

(b) The flowchart of extracting procedure

Figure 5. The flowchart of proposed method

Table 1. Extra information and location map

	Information	Notation	Purposes	Size in bit
1	Block size $m \times n$	m, n	Divide host image into blocks	6
2	The size of data bits	DS	To embed data bits	18
3	The size of compressed location map	MS	To embed compressed location map	14
4	The ordinal of this block	BO	To embed a sequence of LSB of stego pixels	16
6	The compressed location map	LM	To embed and extract ML	MS

define a flag bit as in Subsection 3.3 and insert it into data. Try to embed the data bits in largest pixels, and then try to embed the data in smallest pixels of block as in Subsection 3.1.
Step 3. When embedding data bits are completed, embed first (MS+54) least significant bits (LSB) of the pixels in the stego image into the remaining blocks from block which ordinal equal BO (see Table 1).
Step 4. Use the LSB method [3] to embed extra information and the compressed location

map into stego image from the first pixel.

3.4.2. Extracting procedure

The following steps need to perform in order to extract the data bits and restore original pixels.

Step 1. First, extract the extra information (MS, DS, OB, etc) and the compressed location map by using LSB method, and then decompress the location map to obtain LM.

Step 2. Divide the stego image into non-overlapped blocks of size $m \times n$ as in embedding procedure. Extracting according to Subsection 3.2. (MS+54) bits from the block having ordinal equal OB, and insert extracted bits into first (MS+54) LSB of the pixels in stego image. At the same time, restoring original blocks.

Step 3. Extracting DS data bits from the first block in stego image and restoring original blocks as in Subsection 3.2.

Thus, we obtain the data bits and the original image.

3.5. Comparison between methods GePVO-K and iGePVO-K

We will compare methods GePVO-K and iGePVO-K on two criteria: Stego image quality and embedding capacity.

3.5.1. Stego image quality

In GePVO-K, before embedding, the largest and second largest pixels must be increased by 1 (at largest side). Then in embedding process, largest pixels can be add by 1, so each pixel is changed at most by 2. In our method, each pixel is modified at most by 1. Therefore, the amount of modification in our method is less than GePVO-K method. This leads to the stego image quality of our method is better.

3.5.2. Embedding capacity

There are some reasons for the capacity of the proposed method more than GePVO-K. First reason, for the receiver can distinguish three types of image blocks under/over flow blocks, flat blocks and remaining blocks, GePVO-K method uses a binary location map having the length equals twice number of blocks. Meanwhile, our iGePVO-K uses a binary location map having the length equals only half of the map in GePVO-K. Moreover, iGePVO-K uses a number of flag bits equals the number of flat block with value smaller 254 plus the number of blocks having two consecutive distinct pixel values used for embedding all bit 0. So the number of flag bits, in general, is small. Therefore, the number of bits used to recognize the type of blocks in iGePVO-K is greatly smaller in GePVO-K.

The second reason is that the flat blocks contained 0/1/254/255 are ignored in GePVO-K, but, in proposed method, they are used to embed.

The third reason is that in all embedding cases, both GePVO-K and iGePVO-K have the same capacity unless the case when block having three consecutive distinct values. In this case, GePVO-K increases the second largest pixel values, it leads into the difference between the smallest values and the second smallest values smaller -1. So GePVO-K cannot embed data in the smallest pixels. Meanwhile iGePVO-K still can embed data bits in these pixels.

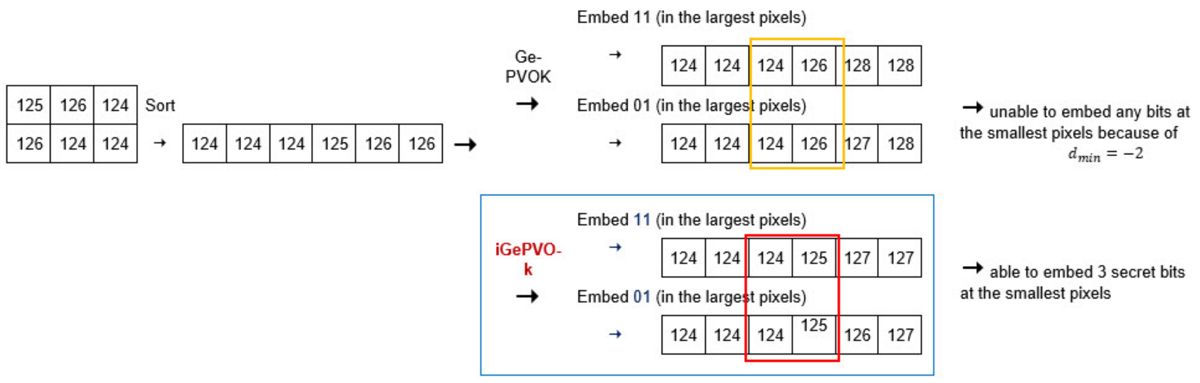

125 126 124
126 124 124
Sort
124 124 124 125 126 126
Ge-PVOK
Embed 11 (in the largest pixels)
124 124 124 126 128 128
Embed 01 (in the largest pixels)
124 124 124 126 127 128
unable to embed any bits at the smallest pixels because of $d_{min} = -2$
iGePVO-k
Embed 11 (in the largest pixels)
124 124 124 125 127 127
Embed 01 (in the largest pixels)
124 124 124 125 126 127
able to embed 3 secret bits at the smallest pixels

Figure 6. Example about enhancing capacity of our proposed method

To illustrate this, an example is given in Figure 6. In this example, the proposed method can embed three data bits while GePVO-K cannot.

Figure 7. Experimental images

4. EXPERIMENTS

To validate the efficiency of our proposed method, iGePVO-K, and to consolidate the results of theoretical analysis, we perform experiments on the sample image set in [1], with some types of images as common images, texture images, flat regions images, etc as shown in Figure 7. Images have 8-bits color and size 512×512. The data is a random bit sequence. Programs are written in the Matlab platform and ran on IdeaPad S410p Lenovo computer (Intel Core i5-4200U CPU and 4GB RAM).

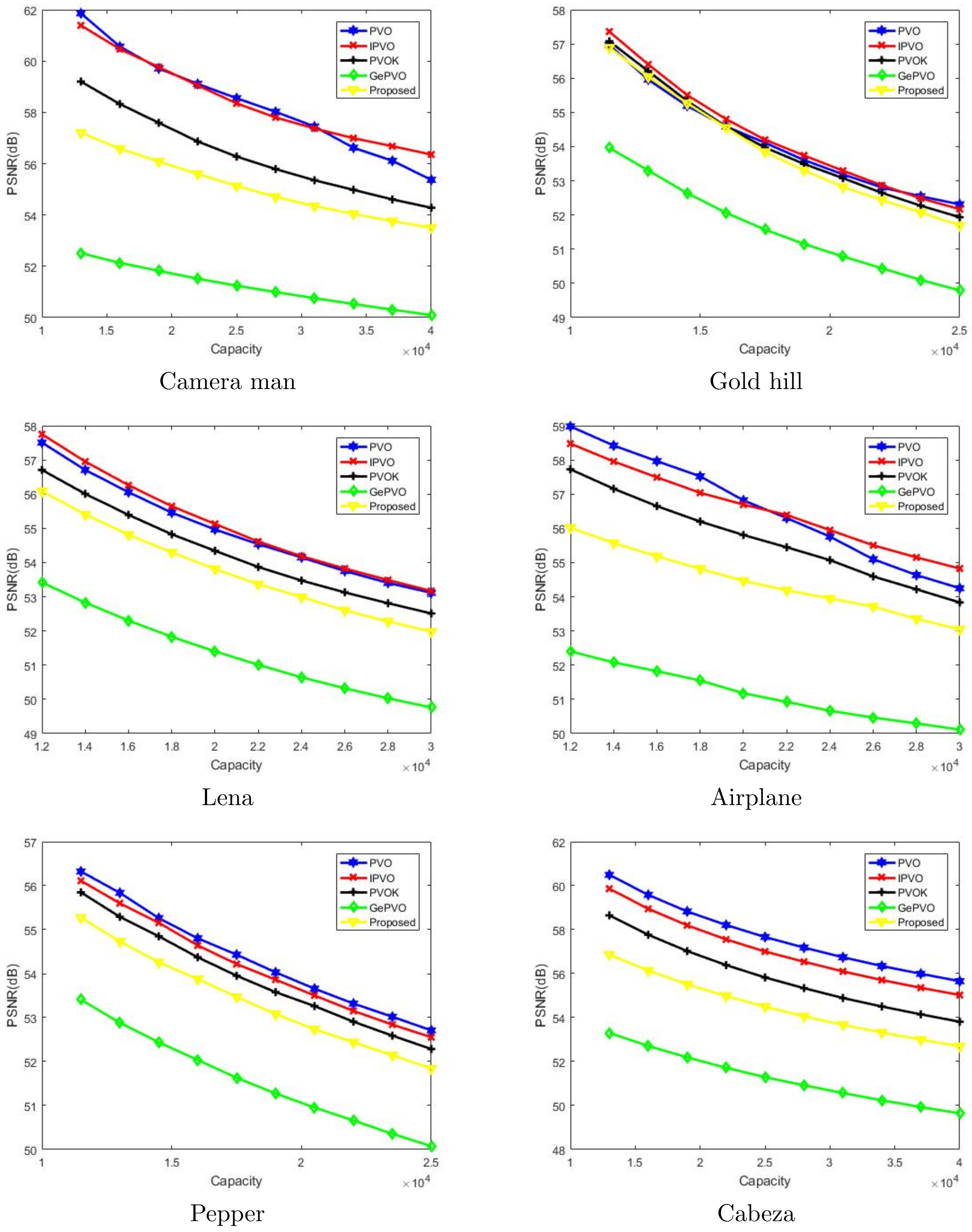
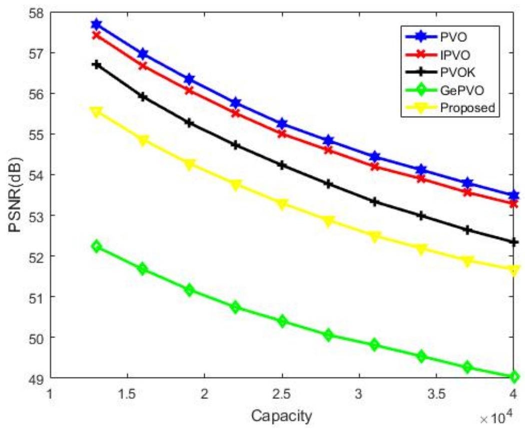
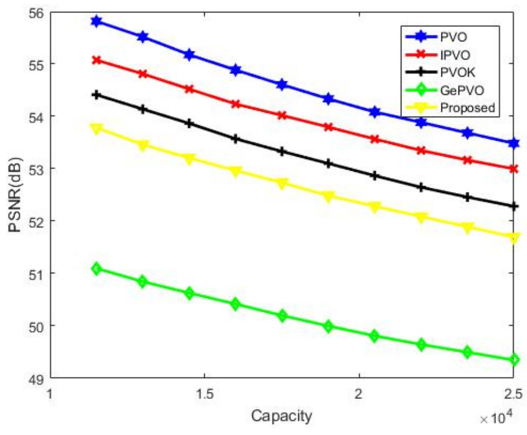


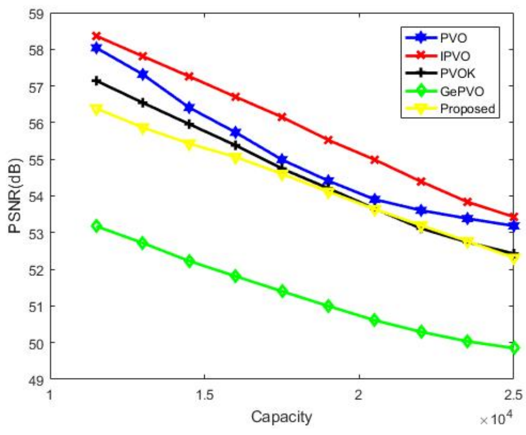
Figure 8. Performance comparison between iGePVO-K and PVO, IPVO, PVO-K, GePVO-K methods



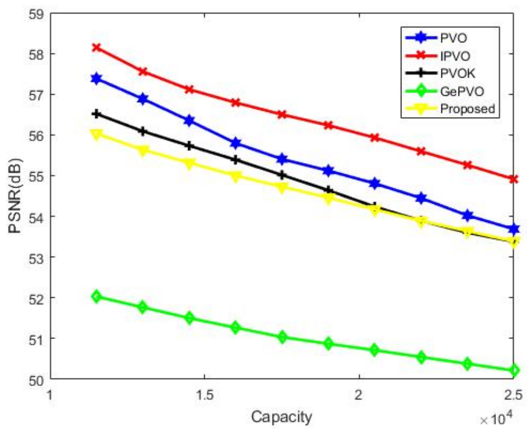
Blob



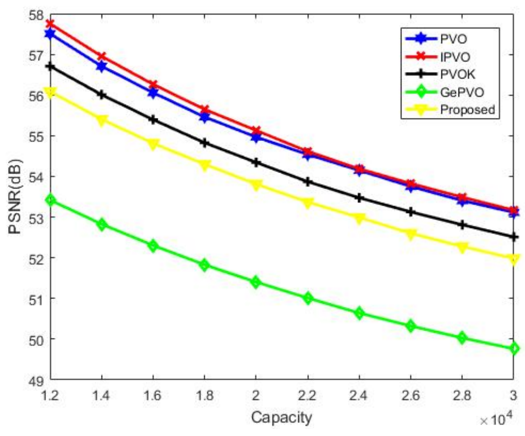
Tiffany



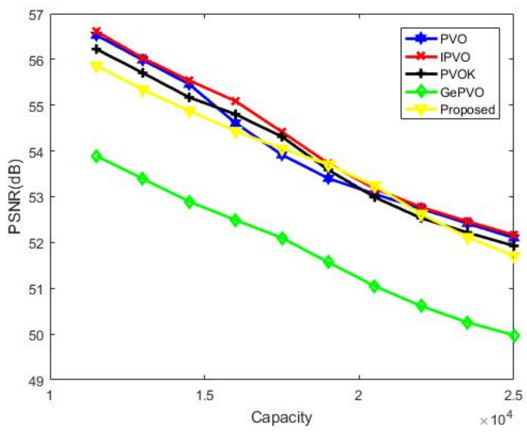
Boat



Barbara



Car



Sail boat

Figure 9. Performance comparison between iGePVO-K and PVO, IPVO, PVO-K, GePVO-K methods

Table 2. Comparisons in be embedded of PSNR (dB) with payload of 10000 bits (blocks sized 2×2)

Images	PVO	IPVO	PVOK	GePVO-K	**iGePVO-K**
Airplane	59.60	59.18	58.48	52.78	56.61
Baboon	52.60	52.42	52.35	51.26	52.11
Barbara	57.76	58.78	57.03	52.29	56.54
Blob	58.60	58.43	57.72	52.85	56.46
Boat	58.84	58.93	57.82	53.57	56.88
Cabeza	61.62	61.02	59.80	54.00	57.74
Cameraman	63.07	62.47	60.27	52.94	57.92
Car	58.27	62.50	59.41	54.68	59.82
Couple	55.83	56.77	55.09	52.82	54.92
Gold hill	58.05	58.62	58.16	54.75	57.92
Lena	58.38	58.77	57.56	54.09	56.92
Pepper	56.98	56.77	56.49	54.00	55.91
Sailboat	56.53	56.61	56.23	53.89	55.86
Tiffany	56.23	55.43	54.78	51.37	54.18
Tranmission Tower	56.66	60.92	57.38	49.73	56.93
Average	**57.93**	**58.51**	**57.24**	**53.00**	**56.45**

Table 3. Comparisons in terms of PSNR (dB) with payload of 20000 bits (blocks sized 2×2)

Images	PVO	IPVO	PVOK	GePVO-K	**iGePVO-K**
Airplane	56.82	56.70	55.81	51.18	54.46
Baboon	51.62	51.38	51.15	49.89	50.70
Barbara	54.92	56.04	54.37	50.75	54.26
Blob	56.15	55.87	55.08	51.03	54.10
Boat	54.05	55.17	53.85	50.74	53.80
Cabeza	58.62	57.98	56.81	52.02	55.33
Cameraman	59.50	59.52	57.33	51.72	55.92
Car	54.40	56.88	54.15	51.17	57.26
Couple	52.64	53.41	51.90	50.04	51.76
Gold hill	53.32	53.44	53.21	50.91	52.98
Lena	54.97	55.13	54.34	51.40	53.81
Sailboat	52.85	52.89	52.67	50.75	52.81
Tiffany	54.16	53.65	52.94	49.87	52.35
Average	**54.92**	**55.23**	**54.12**	**50.88**	**53.81**

The experiments are used to compare the methods PVO, IPVO, PVO-K, GePVO-K and iGePVO-K on two criteria: Embedding capacity and stego image quality. The results obtained are illustrated in the following figures and tables.

Figures 8, 9 and Table 2, 3 show the comparison results on stego image quality. For the same payloads, the proposed method has the stego image quality little lower than PVO, IPVO, PVO-K but much better than GePVO-K.

Tables 4, 5 allow to confirm that the proposed method outperforms other methods on the embedding capacity. Especially, with the blocks sized 3×3, proposed method has capacity equal approximately two times than PVO, PVO-K, IPVO methods. With blocks sized 2×2,

Table 4. Comparisons in terms of capacity with blocks sized 2×2 (number of bits)

Images	PVO	IPVO	PVOK	GePVO-K	**iGePVO-K**
Car	31378	47327	41594	44644	59755
Transmission tower	19603	57764	30813	22507	66877
Blob	41495	53326	52774	56473	66623
Airplane	38454	53055	51505	56305	68296
Couple	21599	29950	25127	26869	29147
Gold hill	25955	28662	29919	31210	33858
Sail boat	23569	26314	27463	28993	31417
Tiffany	29995	34776	34091	34418	42721
Baboon	13146	13462	13940	14546	14899
Barbara	29490	48066	40203	41008	55985
Boat	27392	35159	32784	33321	39603
Cabeza	54263	71946	73417	79199	93531
Camera man	43458	71097	63378	69772	90790
Lena	33115	40176	40373	43198	47773
Pepper	28093	30886	32422	34853	36554
Average	**30734**	**42798**	**39320**	**41154**	**51855**

Table 5. Comparisons in terms of capacity with blocks sized 3×3 (number of bits)

Images	PVO	IPVO	PVOK	GePVO-K	**iGePVO-K**
Airplane	17881	27060	29599	47207	49959
Baboon	7507	8031	8557	9739	9741
Barbara	13730	21787	22553	35730	39166
Blob	19471	27505	30397	46220	48083
Boat	13327	17570	18697	26062	27236
Cabeza	23600	35539	40126	63696	65693
Cameraman	18424	32896	35459	58795	66387
Car	14590	23695	24375	43629	47185
Couple	10943	16431	14631	19110	19791
Gold hill	13647	16035	17631	22669	23002
Lena	16910	21388	23695	32680	33013
Pepper	14931	17369	19399	24891	24992
Sailboat	12325	14646	16240	21209	21566
Tiffany	13975	16388	18363	27048	28774
TranmissionTower	9503	26174	19914	50806	65718
Average	**14718**	**21501**	**22642**	**35299**	**38020**

the ending rate approximately equal 0.12; 0.16; 0.15; 0.16 and 0.20 bpp(bit per pixel) for methods PVO, IPVO, PVO-K, GePVO-K and iGePVO-K, respectively. For blocks sized 3×3 embedding rates approximately equal 0.06; 0.08; 0.09; 0.13 and 0.15 for the above methods.

The location map embedded into the stego image will be used to extract the data and restore the original image. The size of the location map is smaller, the embedding capacity is larger. Table 6 shows the size of compressed location map (in the proposed method, it is added by the number of flag bits).

Table 6. Comparisons in terms of compressed map size and flag (proposed) with blocks sized 2×2 (number of bits)

Images	PVO	IPVO	PVOK	GePVO-K	**iGePVO-K**
Car	0	0	0	16907	4872
Transmission tower	79	79	79	55964	14090
Blob	0	0	0	8151	2574
Airplane	0	0	0	9792	3461
Couple	1060	1327	1327	3029	1699
Gold hill	0	0	0	1512	405
Sail boat	0	0	0	1321	460
Tiffany	3962	6979	6979	13224	8682
Baboon	65	268	268	448	308
Barbara	35	35	35	15458	4435
Boat	183	326	326	5792	1803
Cabeza	64	64	64	7259	3401
Camera man	710	1041	1041	19468	7991
Lena	0	0	0	2646	905
Pepper	50	50	50	658	314
Average	**414**	**678**	**678**	**10775**	**3693**

It can be seen that the proposed method has the compressed location map of size about 30% comparing with GePVO-K. Therefore, it has the capacity much more than GePVO-K.

5. CONCLUSION

In this paper, based on PVO technique, we proposed a new method iGePVO-K which is an improvement of GePVO-K. In the proposed method, we use reasonable data embedding formulas and reduce the location map in order to increase the embedding capacity. Each pixel value is just modified at most by one in iGePVO-K, while it can be changed by two in GePVO-K. Both the theoretical analysis and experimental results show that our proposed method iGePVO-K has larger embedding capacity and better stego image quality than GePVO-K. In comparison with PVO, IPVO, PVO-K methods, iGePVO-K has stego image quality little lower, but the embedding capacity much larger. In future research, we will further improve the stego image quality of iGePVO-K.

ACKNOWLEDGEMENTS

This work is partially supported by the national research project No. KC.01.19/16-20, granted by the Ministry of Science and Technology of Vietnam (MOST).

REFERENCES

[1] "Images data," http://decsai.ugr.es/cvg/dbimagenes and http://sipi.usc.edu/database, 2017.

[2] G. Bhatnagar and B. Raman, "A new robust reference watermarking scheme based on dwt-svd," *Computer Standards and Interfaces*, vol. 31, pp. 1002–1013, 2009.

[3] C.-K. Chan and L.-M. Cheng, "Hiding data in images by simple lsb substitution," *Pattern Recognition*, vol. 37, pp. 469–474, 2004.

[4] J. Fridrich, M. Goljan, and R. Du, "Invertible authentication," *Security and Watermarking of Multimedia Contents*, vol. 4314, pp. 197–209, 2001.

[5] L.-G. J. Gui, Guo-fu and C. He, "A new asymmetric watermarking scheme for copyright protection," *Communications and Computer Sciences*, vol. 89, pp. 611–614, 2006.

[6] J.-J. Li, Y.-H. Wu, C.-F. Lee, and C.-C. Chang, "Generalized pvo-k embedding technique for reversible data hiding," *International Journal of Network Security*, vol. 20, pp. 65–77, 2018.

[7] X. Li, J. Li, B. Li, and B. Yang, "High-fidelity reversible data hiding method based on pixel-value-ordering and prediction-error expansion," *Signal Processing*, vol. 93, pp. 198–205, 2013.

[8] Y.-C. Li, C.-M. Yeh, and C.-C. Chang, "Data hiding based on the similarity between neighboring pixels with reversibility," *Digital Signal Processing*, vol. 20, pp. 1116–1128, 2010.

[9] B. Macq, "Lossless multiresolution transform for image authenticating watermarking," *Signal Processing Conference*, vol. 10th European, pp. 1–4, 2000.

[10] Z. Ni, Y.-Q. Shi, N. Ansari, and W. Su, "Reversible data hiding," *Circuits and Systems for Video Technology, IEEE Transactions*, vol. 16, pp. 354–362, 2006.

[11] B. Ou, X. Li, Y. Zhao, and R. Ni, "Reversible data hiding using invariant pixel-value-ordering and prediction-error expansion," *Digital Signal Process*, vol. 29, pp. 760–772, 2014.

[12] X. L.-W. L. Ou, Bo and Y.-Q. Shi, "Pixel-value-ordering based reversible data hiding with adaptive texture classification and modification," *In International Workshop on Digital Watermarking*, vol. Cham, pp. 169–179, 2018.

[13] F. Peng, X. Li, and B. Yang, "Improved pvo-based reversible data hiding," *Digital Signal Process*, vol. 25, pp. 255–265, 2014.

[14] X. Qu and H. J. Kim, "Pixel-based pixel value ordering predictor for high-fidelity reversible data hiding," *Journal of Real-Time Image Processing*, vol. 111, pp. 249–260, 2015.

[15] S. Rawat and B. Raman, "A chaos-based robust watermarking algorithm for rightful ownership protection," *International Journal of Image and Graphics*, vol. 11, pp. 471–493, 2011.

[16] K. Sao Nguyen, Q. H. Le, and V. A. Pham, "A new reversible watermarking method based on histogram shifting," *Applied Mathematical Sciences*, vol. 11, pp. 445–460, 2017.

[17] Y.-Q. Shi, X. Li, X. Zhang, H.-T. Wu, and B. Ma, "Reversible data hiding: advances in the past two decades," *IEEE Access*, vol. 4, pp. 3210–3237, 2016.

[18] Q. Su and B. Chen, "Robust color image watermarking technique in the spatial domain," *Soft Computing*, vol. 21, pp. 91–106, 2018.

[19] D. M. Thodi and J. J. Rodrguez, "Reversible watermarking by prediction-error expansion," *Image Analysis and Interpretation, 6th IEEE Southwest Symposium on. IEEE*, vol. 6, pp. 21–25, 2004.

[20] X. L.-Y. Z. Wu, Haorui and R. Ni, "Improved reversible data hiding based on pvo and adaptive pairwise embedding," *Journal of Real-Time Image Processing*, vol. 16, pp. 685–695, 2019.

Received on August 01, 2019
Revised on October 06, 2019