An Improvemant of the IGEPVO-k Reversible Data Hiding Method

Le Kinh Tai[1], Nguyen Ngoc Hung[2], Dang Tran Duc[3], Can Duc Diep[4], Pham Minh Thai[5]
[1]Training department, Trade Union University, 169 Tay Son, Dong Da, Hanoi, Vietnam
[2]Instutite of Information Technology, Vietnam Academy of Science and Technology, 18 Hoang Quoc Viet, Hanoi, Vietnam
[3]VVECO Joint Stock Compa¬ny, 89 Nguyen Phong Sac, Cau Giay, Hanoi, Vietnam
[4]Electric Power University, 235 Hoang Quoc Viet, Hanoi, Vietnam
[5]Faculty of Information Technology, University of Economics-Technology for Industries 454-456, Minh Khai, Hanoi, Vietnam

Correspondence: Nguyen Ngoc Hung (nnhung@ioit.ac.vn)
Communication: received 31 Jan 2023, revised 10 May 2023, accepted 25 May 2023
Digital Object Identifier: 10.32913/mic-ict-research.v2023.n2.1207

Abstract: PVO-K method is an expansion of PVO method in which all k largest (l smallest pixels) are used as a single unit to embed a bit. In PVOK-method, each pixel of an original image is modified by one at most. GePVO-K method is again a development of the PVOK-method by embedding k bits in k largest pixels and l bits in l smallest pixels. However, this method has two drawbacks. First, each pixel of the original image is changed by two at most, so marked image quality is not good. Second, the location map consisting of two bits for each pixel is so large that in some cases compression code length of the map is greater method's embedding capacity, and then data embedding capacity is zero. Both drawbacks are overcome in the iGePVO-K method. Specifically, in this method, each pixel only is changed by one at most and the two-bit location map as in iGePVO-K method is replaced by a one-bit map and a sequence of binary flags. This paper is a development of the iGePVO-K method by discarding the sequence of binary flags. That means it is not necessary to embed flags with the data in the original image. Therefore, the proposed method not only has a larger data embedding capacity but also has a better image quality than the methods mentioned above.

Keywords: *PVO, PVOK, GePVOK, iGePVOK, reversible data hiding, flag bits.*

I. Introduction

Along with cryptography, data hiding is an important research direction in the field of information security. Data hiding is a technique in which secret data is concealed in a digital image. Traditional data-hiding methods can only extract embedded data but do not restore the original image. This limits their applications. in some sensitive fields, such as medical image processing and military communications. So, new data hiding techniques called reversible (or lossless) data hiding are proposed to overcome this limitation. Reversible data hiding (RDH) techniques not only can extract embedded data but also restore exactly the original image.

The first RDH methods were built based on lossless compression [1–4], in that to have an embedding space we use a lossless compression technique for a certain region of the original image. Since obtained embedding space usually is not large, the embedding capacity (EC) of these methods is generally not high.

After then, Tian first proposed a technique called difference expansion (DE) [5]. The basic idea of DE is to expand the difference between two neighbouring pixels to embed data. Afterwards, the technique of DE got further improvements in the paper [6–8]. Later, the technique of prediction-error expansion (PEE) was proposed by Thodi and Rodriguez [9]. Unlike DE, PEE utilizes more neighbouring pixels and gets more accurate predictions. Then, this method got further investigated [10, 11], and show superior performance. Besides, DE, Ni et al. proposed a histogram-shifting (HS) based method which is of great value [12]. In this method, the peak pixel values of the host image histogram were modified to embed data. When embedding fewer data, Ni et al.'s algorithm has good performance. However, the host image with a flat histogram has a low EC, and the shortcoming had been overcome partly in the paper [13, 14] by introducing the difference-histogram.

The smoothness of pixel blocks is often exploited by many papers [15–17] because using RDH methods on the smooth blocks not only achieves the high embedding capacity but also causes little distortion of pixels.

In [18], Li et al. proposed an excellent predictor based on pixel value ordering (PVO) for embedding data. Specifically, Li et al.'s method divided an original image into non-overlapped blocks of n pixels. Each block is sorted in ascending order, the largest pixel is predicted by the second largest pixel, the second largest pixel and the smallest pixel is predicted by the second smallest pixel. Then one bit is embedded in the largest pixel (smallest) pixel when the prediction error dmax (dmin) is equal to 1 (-1). This method has a high stego image quality, but a low embedding capacity. Peng et al. [19] were given an improvement of the PVO method called the IPVO method in which the information about the positions of the largest pixel and the second-largest pixel in the original pixel block is used to determine dmax. Specifically, dmax is equal to second largest pixel minus largest pixel or vice versa depending on second largest pixel is before or after largest pixel in the original pixel block. The prediction error dmin is computed similarly by using second smallest pixel and smallest pixel. Then, one bit is embedded in the largest (smallest) pixel when dmax (dmin) is equal to zero or one. In general, the IPVO method has an embedding capacity larger than that of the PVO method. It is noted that if a pixel block has $k \geq 2$ pixels being equal to the largest pixel or has $l \geq 2$ pixels being equal to the smallest pixel, then dmax or dmin is equal to zero. When no bit is embedded in the largest pixel or the smallest pixel by the PVO method. To overcome this weakness, Ou et al. [20] have proposed the PVO-K method in which all largest pixels are considered as a unit to embed a bit. In other words, all the largest pixels are kept unchanged or increased by one to embed a bit with a value of 0 or 1. Data embedding in the smallest pixels is carried out similarly. The embedding capacity of this is improved compared with the PVO method, but not very much.

Li et al. [21] have proposed an extension of the PVO-K method, called GePVO-K. In this method, k bits are embedded in the k largest pixels and l bits are embedded in the l smallest pixels. But in the implementation process of this method, each pixel may be modified by 2 at most. So, the marked image quality of Weng et al.'s method is low. Moreover, the location map consists of two bits for each pixel block (two-bit map) and is too large that in some cases the compression code length of the map is greater than the embedding capacity of the method. In these cases, the data embedding capacity (payload) is zero.
Sao et al.'s method [22] is a development of [21], in which the above weaknesses are partially overcome. Specifically, in this method, each pixel is changed by one at most. However, the two-bit map problem has not been completely solved yet. In [22], the two-bit map is replaced by a one-bit map and a sequence of binary flags. Embedding the flag sequence along with the data in the original image reduces the payload and marked image quality of Sao et al.'s method.

In this paper, we propose some improvements in the embedding algorithm of [22] to discard the flag sequence. That is, there is no need to use flag sequences in the embedding and extracting algorithms. So, both the payload and marked image quality of the proposed are better than above mentioned PVO-based methods.
The rest of the paper is organized as follows. The relative words are presented in section 2. The proposed method is introduced in section 3. The experiment results of comparing methods are given in section 4. Finally, some conclusions are provided in section 5.

II. Related Works

In this section, we briefly present methods PVO [18], PVO-K [20], GePVO-K [21] and iGePVO-K [22].

1. PVO method

The method PVO is proposed by Li et al. [18], in which an original gray-scale image of size $R \times C$ is divided into non-overlapped blocks of n pixels. Each block $X = (x_1, ..., x_n)$ is sorted in ascending order to get $X_\sigma = (x_{\sigma(1)}, ..., x_{\sigma(n)})$ such that $x_{\sigma(1)} \leq ... \leq x_{\sigma(n)}$. Moreover if $i < j$ and $x_{\sigma(i)} = x_{\sigma(j)}$ then $\sigma_i < \sigma_j$. One data bit b can be embedded in the largest pixel $x_{\sigma(n)}$ using the formulas:

$$dmax = x_{\sigma(n)} - x_{\sigma(n-1)}. \tag{1}$$

$$x'_{\sigma(n)} = \begin{cases} x_{\sigma(n)} + b, & \text{if } dmax = 1, \\ x_{\sigma(n)} + 1, & \text{no bit is embedded,if } dmax > 1, \\ x_{\sigma(n)}, & \text{no bit is embedded,if } dmax = 0. \end{cases} \tag{2}$$

Similarly, a bit is embedded in the smallest pixel $x_\sigma(1)$ as follows.

$$dmin = x_{\sigma(1)} - x_{\sigma(2)}. \tag{3}$$

$$x'_{\sigma(1)} = \begin{cases} x_{\sigma(1)} - b, & \text{if } dmin = -1, \\ x_{\sigma(1)} - 1, & \text{if } dmin = -1, \\ x_{\sigma(1)}, & \text{if } dmin = 0. \end{cases} \tag{4}$$

It is clear that after embedding, the PVO (pixel value ordering) of X_σ is kept unchanged, and pixels $x_{\sigma(i)}$ with $i = 2, ..., n-1$ are preserved.
At the decoder side, extracting b and restoring $x_{\sigma(n)}$ can be performed as follows:

$$dmax' = x'_{\sigma(n)} - x'_{\sigma(n-1)} \tag{5}$$

$$b = dmax^{'} - 1, \text{ if } dmax^{'} \in \{1, 2\}. \quad (6)$$

$$x_{\sigma(n)} = \begin{cases} x^{'}_{\sigma(n)} - b, \text{ if } dmax^{'} \in \{1, 2\}, \\ x^{'}_{\sigma(n)} - 1, \text{ if } dmax^{'} > 2, \\ x^{'}_{\sigma(n)}, \text{ if } dmax^{'} = 0. \end{cases} \quad (7)$$

Similarly, extracting b and restoring $x_{\sigma}(1)$ can be performed as follows:

$$dmin^{'} = x^{'}_{\sigma(1)} - x^{'}_{\sigma(2)} \quad (8)$$

$$b = abs(dmin^{'}) - 1, \text{ if } dmin^{'} \in \{-1, -2\}. \quad (9)$$

$$x_{\sigma(1)} = \begin{cases} x^{'}_{\sigma(1)} + b, \text{ if } dmin^{'} \in \{-1, -2\}, \\ x^{'}_{\sigma(1)} + 1, \text{ if } dmin^{'} < -2, \\ x^{'}_{\sigma(1)}, \text{ if } dmin^{'} = 0. \end{cases} \quad (10)$$

2. PVO-K method

The method PVO-K is proposed by Ou Li et al. [20] by embedding one bit in the largest pixels, and one bit in the smallest pixels. Suppose sequence X_{σ} has k largest pixels and l smallest pixels, that means:

$$x_{\sigma(1)} = ... = x_{\sigma(l)} < x_{\sigma(l+1)} < ... \leq x_{\sigma(n-k)} < x_{\sigma(n-k+1)} = ... = x_{\sigma(n)}$$

Then similar to (1-2), embedding one bit in k of the largest pixels simultaneously is performed by formulas:

$$dmax = x_{\sigma(n)} - x_{\sigma(n-k)},$$

$$x^{'}_{\sigma(i)} = \begin{cases} x_{\sigma(i)} + b, \text{ if } dmax = 1, \\ x_{\sigma(i)} + 1, \text{ no bit is embedded, if } dmax > 1, \\ x_{\sigma(i)}, \text{ no bit is embedded, if } dmax = 0, \end{cases}$$

with $i = n - k + 1, ..., n$.

Embedding one bit in the l smallest simultaneously is carried similar to (3-4) as follows.

$$dmin = x_{\sigma(1)} - x_{\sigma(l+1)},$$

$$x^{'}_{\sigma(i)} = \begin{cases} x_{\sigma(i)} - b, \text{ if } dmax = -1, \\ x_{\sigma(i)} - 1, \text{ no bit is embedded, if } dmin < -1, \\ x_{\sigma(i)}, \text{ no bit is embedded, if } dmin = 0, \end{cases}$$

with $i = 1, ..., l$.

At the decoder, extracting the embedded bit and restoring k largest pixels is done similarly to (5-7) by formulas:

$$dmax^{'} = x^{'}_{\sigma(n)} - x^{'}_{\sigma(n-k)}$$

$$b = dmax^{'} - 1, \text{ if } dmax^{'} \in \{1, 2\},$$

$$x_{\sigma(i)} = \begin{cases} x^{'}_{\sigma(i)} - b, \text{ if } dmax^{'} \in 1, 2, \\ x^{'}_{\sigma(i)} - 1, \text{if } dmax^{'} > 2, \\ x^{'}_{\sigma(i)}, \text{ if } dmax^{'} = 0, \end{cases}$$

with $i = n - k + 1, ..., n$.

Similar to (8-10), extracting the embedded bit and restoring l smallest pixels is performed as follows.

$$dmin^{'} = x^{'}_{\sigma(1)} - x^{'}_{\sigma(l+1)}$$

$$b = abs(dmin)^{'} - 1, \text{ if } dmin^{'} \in \{-1, -2\},$$

$$x_{\sigma(i)} = \begin{cases} x^{'}_{\sigma(i)} + b, \text{ if } dmin^{'} \in \{-1, -2\}, \\ x^{'}_{\sigma(i)} + 1, \text{ if } dmin^{'} < -2, \\ x^{'}_{\sigma(i)}, \text{ if } dmin^{'} = 0, \end{cases}$$

with $i = 1, ..., l$.

In PVO-K, a one-bit block-based location map that consists of one bit for each block is built as follows. Assign 1 for blocks having at least one pixel with a value equal to 0 or 255 and assign 0 for the remaining blocks. The blocks with a map index of 1 should not be used for embedding as that would cause underflow/overflow.

3. GePVO-K method

The GePVO-K method [21] is a generalization of the PVO-K method by embedding k bits in k largest pixels and l bits in l smallest pixels. In this method, blocks of n pixels are classified into three types:

(a) Blocks with at least one-pixel values of 0, 1, 254 and 255 should not be used for data embedding as this may cause underflow/underflow. Their index in the location map is set by 2 ($LM(X) = 2$)

(b) Blocks with all equal pixels: $x_1 = x_2 = ... = x_n = \alpha, \alpha \in \{2, ..., 243\}$ (flat blocks) will be used to embed $n - 1$ bits. Their index in the location map is set by 1 ($LM(X) = 1$)

(c) The remaining blocks (not-flat blocks) will be used to embed k bits in k largest pixels and l bits in l smallest pixels. Their index in the location map is set by 0 ($LM(X) = 0$)

Since each index in the map is a two-bit binary number, the location map is called a two-bit map that is a binary sequence with a length twice the number of blocks.

The processing of each block depends on its map index as follows.

Case 2.3.1: $LM(X) = 2$, block X is not used to embed the data and it is skipped.

Case 2.3.2: $LM(X) = 1$, keep the first pixel and add $(n-1)$ to-be-embedded bits to the remaining $(n - 1)$ pixels:

$$x^{'}_{i} = \begin{cases} x_i, \text{ if } i = 1, \\ x_i + b_{i-1}, \text{ if } i = 1, ..., n, \end{cases}$$

where b_i are to-be-embedded bits.

Case 2.3.3: $LM(X) = 0$, the block X is sorted in ascending order to get $X_{\sigma} = (x_{\sigma(1)}, ..., x_{\sigma(n)})$. Suppose X_{σ} has k_1

largest pixels and k_2 second largest pixels. Then compute $dmax = x_{\sigma(n)} - x_{\sigma(n-k_1)}$ and embed the data bits based on value of $dmax$ as follows:
Case 2.3.3.1: $dmax > 1$, no bit is embedded, and all largest pixels are increased by 1.
Case 2.3.3.2: $dmax = 1$, k_1 bits $(b_i, i = 1, ..., k_1)$ are embedded in k_1 largest pixels by formulas:

$$x'_{\sigma(i)} = \begin{cases} x_{\sigma(i)} + 1, & i = n - k_1 - k_2 + 1, ..., n - k_1, \\ x_{\sigma(i)} + 1 + b_{i-n+k_1}, & i = n - k_1 + 1, ..., n. \end{cases}$$

For the smallest pixels, suppose X_σ has l_1 smallest pixels and l_2 second smallest pixels. Compute $dmin = x_{\sigma(1)} - x_{\sigma(l_1+1)}$ and data embedding on l_1 smallest pixels is performed in the same way as for k_1 largest pixels.

4. iGePVO-K method

iGePVO-K method [22] is an improvement of GePVO-K in which each pixel is changed by one at most. The location map of this method is a one-bit based-blocked map. Blocks are classified into types:

(a) The block with at least a pixel value equal to 0 or 255 is called underflow/overflow.
(b) The block with all equal pixels is called flat, otherwise is called rough.
(c) The block which contains s different pixel values, with $s > 1$, is called a s-block.

First, a location map is established to distinguish underflow/overflow and non-underflow/overflow blocks by assigning map index 1 for the former $(LM(X) = 1)$ and 0 for the latter $(LM(X) = 0)$.

a) Data embedding in the largest pixels

Data embedding in the largest pixels is performed for the following case:
Case 2.4.1.1: X is rough and $LM(X) = 1$. X is not used to embed data, it is ignored because embedding in this block can cause underflow/overflow.
Case 2.4.1.2: X is a flat block with $x_1 = x_2 = ... = x_n = \alpha$, $0 \le \alpha \le 255$. In this case $(n-1)$ bits are embedded in X as follows:

$$x'_i = \begin{cases} x_i + b_{i-1}, & \text{if } \alpha \le 254, i = 2, ..., n, \\ x_i - b_{i-1}, & \text{if } \alpha = 255, i = 2, ..., n, \\ x_1, & \text{if } i = 1. \end{cases}$$

Case 2.4.1.3: X is a 2-block and $LM(X) = 0$. In this case, X consists of two different pixels values:

$$x_{\sigma(1)} = ... = x_{\sigma(n-k)} < x_{\sigma(n-k+1)} = ... = x_{\sigma(n)}$$

Compute:

$$dmax = x_{\sigma(n)} - x_{\sigma(n-k)}$$

2.4.1.3.1. If $dmax \ge 2$, no bit embedded and k largest pixels are increased by 1:

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1, i = n - k + 1, ..., n,$$

2.4.1.3.2. If $dmax = 1$, k bits are embedded in k largest pixels as follows:

(a) If all k bits are equal to 1, then pixels are modified as in 2.4.1.3.1:

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1, i = n - k + 1, ..., n,$$

(b) If all k bits are equal to 0, then k largest pixels are kept unchanged:

$$x'_{\sigma(i)} = x_{\sigma(i)}, i = n - k + 1, ..., n.$$

(c) If k bits include both 0 and 1: k bits are added to k largest pixels:

$$x'_{\sigma(i)} = x_{\sigma(i)} + b_{i-n+k}, i = n - k + 1, ..., n.$$

Case 2.4.1.4: X is a s-block with $s \ge 3$ and $LM(X) = 0$:

$$x_{\sigma(1)} \le ... = x_{\sigma(n-k_1-k_2)} < x_{\sigma(n-k_1-k_2+1)} = ... = x_{\sigma(n-k_1)} < x_{\sigma(n-k_1+1)} = ... = x_{\sigma(n)}$$

Compute:

$$dmax = x_{\sigma(n)} - x_{\sigma(n-k_1)}$$

2.4.1.4.1 If $dmax \ge 2$, no bit is embedded and then k_1 largest pixels are increased by 1:

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1, i = n - k_1 + 1, ..., n.$$

2.3.4.4.2 If $dmax = 1$, k_1 bits are embedded in k_1 largest pixels as follows:

(a) If all k_1 bits are equal to 1, then k_1 largest pixels are increased by 1:

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1, i = n - k_1 + 1, ..., n.$$

(b) If all k_1 bits are equal to 0, then k_1 largest pixels and k_2 second largest pixels are increased by 1:

$$x'_{\sigma(i)} = x_{\sigma(i)} + 1, i = n - k_1 - k_2 + 1, ..., n.$$

(c) If k_1 bits include both 0 and 1: k_1 bits are added to k_1 largest pixels:

$$x'_{\sigma(i)} = x_{\sigma(i)} + b_{i-n+k_1}, i = n - k_1 + 1, ..., n.$$

b) Data embedding in the smallest pixels

Data embedding in the smallest pixels only is performed in the cases 2.4.1.3 and 2.4.1.4, that is X is a s-block with $s \geq 2$.

For convenience, the pixel block obtained after performing the steps in subsection 2.4.1 is also denoted by X, then X is naturally an s block with $s \geq 2$. Sort X in ascending order to get X_σ. Consider the following two cases:

Case 2.4.2.1: s = 2. Then

$$x_{\sigma(1)} = ... = x_{\sigma(l)} < x_{\sigma(l+1)} = ... = x_{\sigma(n)}$$

Compute:

$$dmin = x_{\sigma(1)} - x_{\sigma(l+1)}$$

2.4.2.1.1. If $dmin \leq -2$, no bit embedded and l smallest pixels are decreased by 1:

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

2.4.2.1.2. If $dmin = -1$, l bits are embedded in l smallest pixels as follows:

(a) If all l bits are equal to 1, then l smallest pixels are decreased by 1 as in the previous case.

(b) If all l bits are equal to 0, then l smallest pixels are kept unchanged:

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

(c) If l bits include both 0 and 1: l bits are subtracted from l smallest pixels:

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - b_{(i-n+k)}, i = n-k+1, ..., n.$$

Case 2.4.2.2: $s \geq 3$. Then

$$x_{\sigma(1)} = ... = x_{\sigma(l_1)} < x_{\sigma(l_1+1)} = ... = x_{\sigma(l_1+l_2)} < x_{\sigma(l_1+l_2+1)} \geq ... \geq x_{\sigma(n)}$$

Compute

$$dmin = x_{\sigma(1)} - x_{\sigma(l_1+1)}.$$

2.4.2.2.1 If $dmin \leq -2$, no bit is embedded, l_1 smallest pixels are decreased by one.

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - 1, i = 1, ..., l_1$$

2.4.2.2.2 If $dmin = -1$, embed l_1 bits in l_1 smallest pixels.

(a) If all l_1 to-be-embedded bits are equal to 1: Decrease l_1 smallest pixels by one.

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - 1, i = 1, ..., l_1$$

(b) If all l_1 to-be-embedded bits are equal to 0: Decrease l_1 smallest pixels and l_2 second smallest pixels by one.

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - 1, i = 1, ..., l_1 + l_2$$

(c) If l_1 to-be-embedded bits include both 0 and 1: Subtract l_1 bits from l_1 smallest pixels:

$$x^{'}_{\sigma(i)} = x_{\sigma(i)} - b_i, i = 1, ..., l_1$$

After completing the steps in case 2.4.2.2, the marked pixel block $X^{'}$ that contains to-be-embedded bits is obtained.

For ease of presenting the next sections, the following concept is introduced. A block whose pixel value set contains two consecutive values α and β, with $|\alpha - \beta| = 1$ is called a 2-flat block. The block in case 2.4.1.3.2 of subsection 2.4.1 is a 2-flat block.

It is noted that after embedding the data, both flat and 2-flat block types can be converted to 2-flat. So, a sequence of flag bits in which flag 0 for a flat block and 1 for a 2-flat block is established for distinguishing flat and 2-flat blocks in the extracting process.

c) Extracting and restoring in smallest pixels

Given a marked block $X^{'}$, extracting data and restoring pixels on the left are performed as follows.

Case 2.4.3.1: $LM(X) = 1$, X is an underflow/overflow block. No bit is extracted, and $X = X^{'}$.

Case 2.4.3.2: $X^{'}$ is flat or 2-flat, then X is also flat or 2-flat. To determine correctly the type of X, use the sequence of flag bits. If the flag is 0, X is flat, extracting and restoring by formulas:

$$b_{i-1} = \begin{cases} x^{'}_i - x^{'}_1, & \text{if } x^{'}_1 < 255, \\ x^{'}_1 - x^{'}_i, & \text{if } x^{'}_1 = 255, \end{cases} \quad (i = 2, \ldots, n)$$

$x_i = x^{'}_i$, $(i = 1, \ldots, n)$.

Case 2.4.3.3: $X^{'}$ is 2-block:

$$x^{'}_{\sigma(1)} = \ldots = x^{'}_{\sigma(l)} < x^{'}_{\sigma(l+1)} = \ldots = x^{'}_{\sigma(n)}$$

Compute

$$\text{dmin} = x^{'}_{\sigma(1)} - x^{'}_{\sigma(l+1)}.$$

(a) If $dmin \leq -3$, no bit is extracted, l smallest pixels are increased by 1:

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

(b) If $dmin = -2$, all extracted l bits are equal to 1 and restoring l smallest pixels is carried out as in (a).

(c) If $dmin = -1$, all extracted l bits are equal to 0 and restoring l smallest pixels is as:

Case 2.4.3.4: $X^{'}$ is s-block with $s \geq 3$

$$x^{'}_{\sigma(1)} = .. = x^{'}_{\sigma(l_1)} < x^{'}_{\sigma(l_1+1)} = ... = x^{'}_{\sigma(l_1+l_2)} < x^{'}_{\sigma(l_1+l_2+1)} \leq ... \leq x^{'}_{\sigma(n)}$$

Compute

$$\text{dmin} = x'_{\sigma(1)} - x'_{\sigma(l_1+1)},$$

$$\text{dmin2} = x'_{\sigma(l_1+1)} - x'_{\sigma(l_1+l_2+1)}.$$

(a) If $dmin \leq -3$, no bit is extracted, l_1 smallest pixels are increased by 1:

$$x_{\sigma(i)} = x'_{\sigma(i)} + 1,\ i = 1, \ldots, l_1$$

(b) If $dmin = -2$, all extracted l_1 bits are equal to 1 and restoring l_1 smallest pixels is carried out as in (a).

(c) If $dmin = -1$ and $dmin2 \leq -2$, all extracted l_1 bits are equal to zero, restoring l_1 smallest pixels and l_2 second smallest pixels are performed by formulas:

$$x_{\sigma(i)} = x'_{\sigma(i)} + 1,\ i = 1, \ldots, l_1 + l_2.$$

(d) If $dmin = -1$ and $dmin2 = -1$, l_1 bits of 1 and l_2 bits of 0 are extracted. l_1 smallest pixels are increased by 1. Then it is necessary to rearrange the extracted bits as well as (l+m) the received smallest pixels as their original position in block X.

d) Extracting and restoring in largest pixels

The cases $LM(X) = 1$ and X' obtained from flat X have been considered in subsection 2.4.3. In this subsection, consider the next two cases.

Case 2.4.4.1: X' is 2-block:

$$x'_{\sigma(1)} = \ldots = x'_{\sigma(n-k)} < x'_{\sigma(n-k+1)} = \ldots = x'_{\sigma(n)}$$

Compute

$$\text{dmax} = x'_{\sigma(n)} - x'_{\sigma(n-k)}.$$

(a) If $dmax \geq 3$, no bit is extracted, k largest pixels are decreased by 1:

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

(b) If $dmax = 2$, all extracted k bits are equal to 1 and restoring k largest pixels is carried out as in (a).

(c) If $dmax = 1$, all extracted k bits are equal to 0 and restoring k largest pixels is as:

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

Case 2.4.4.2: X' is s-block with $s >= 3$:

$$x'_{\sigma(1)} \leq \ldots \leq x'_{\sigma(n-k_1-k_2)} < x'_{\sigma(n-k_1-k_2+1)} = \ldots = x'_{\sigma(n-k_1)} < x'_{\sigma(n-k_1+1)} \leq \ldots \leq x'_{\sigma(n)}$$

Compute

$$\text{dmax} = x'_{\sigma(n)} - x'_{\sigma(n-k_1)},$$

$$\text{dmax2} = x'_{\sigma(n-k_1)} - x'_{\sigma(n-k_1-k_2)}.$$

(a) If $dmax \geq 3$, no bit is extracted, k_1 largest pixels are decreased by 1:

$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = n - k_1 + 1, \ldots, n$$

(b) If $dmax = 2$, all extracted k_1 bits are equal to 1 and restoring k_1 largest pixels is carried out as in (a).

(c) If $dmax = 1$ and $dmax2 \geq 2$, all extracted k bits are equal to zero, restoring k_1 largest pixels and k_2 second largest pixels is performed by formulas:

$$x_{\sigma(i)} = x'_{\sigma(i)} - 1,\ i = n - k_1 - k_2 + 1, \ldots, n$$

(d) If $dmax = 1$ and $dmax2 = 1$, k_1 bits of 1 and k_2 bits of 0 are extracted, k_1 largest pixels are decreased by 1. Then it is necessary to rearrange the extracted bits as well as (k_1+k_2) the received largest pixels as their original position in block X

III. PROPOSED METHOD

1. Comments on the iGePVO-K algorithm

One drawback of the iGePVO-K method is that this method needs to use a sequence of flag bits to distinguish flat and 2-flat blocks in the extracting stage. For ease of understanding, we consider a flat block X_1 and a 2-flat X_2 as follows:

$$X_1 = \begin{bmatrix} 24 & 24 \\ 24 & 24 \end{bmatrix}, X_2 = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}$$

Suppose we embed bit sequence (1, 1, 0) in X_1, then by subsection 2.4.1.2, we obtain a marked block as follows:

$$X'_1 = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}$$

Similarly, if embed bit sequence (0, 0, 0, 0) in X_2, then by case 2.4.1.3.2(b) and case 2.4.2.1.2(b), we obtain a marked block:

$$X'_2 = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}.$$

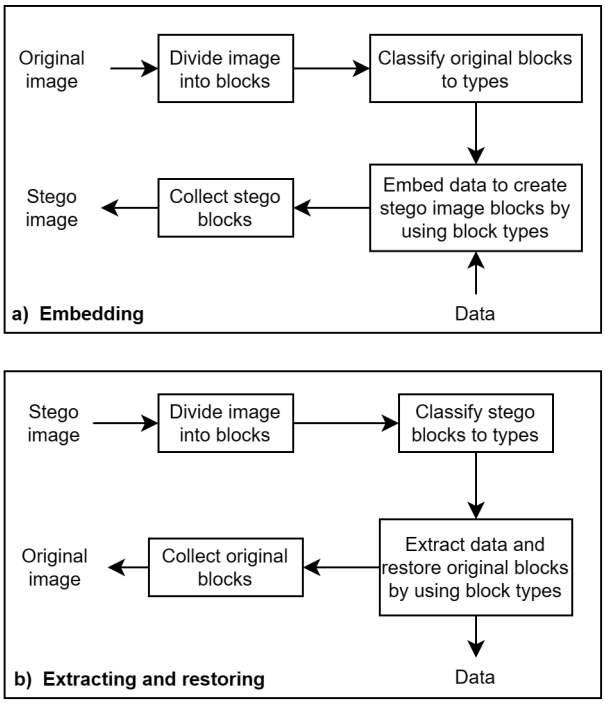

Figure 1. The framework of proposed method

Thus

$$X_1^{'} = X_2^{'} = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}.$$

Therefore, in the data extracting process, if encounter blocks $X_1^{'}$ and $X_2^{'}$, we will not know if their original blocks are flat or 2-flat. To avoid this ambiguity, we need to use the sequence of flags as follows: the flag of $X_1^{'}$ is equal to 0, then its original block X_1 is flat, while the flag of $X_2^{'}$ is equal to 1, then its original block X_2 is 2-flat.
To be reversible, the flag bits must be embedded with the data in the original image. The number of flag bits is equal to the sum of a flat block number and a flat 2 block number. In many cases, this number is relatively large. It can be in the thousands. Removing the flag bits in the proposed method expands embedding capacity considerably.

2. Embedding algorithm of the proposed method

To remove the flag sequence, it is necessary to modify the embedding formulas in the iGePVO-K method so that only flat blocks can lead to 2-flat marked blocks. To achieve this it is necessary to modify the embedding algorithm as follows.

a) Embedding data in the largest pixels

We will revise formulas in case 2.4.1.3 of the iGePVO-K method by the following new formulas in Case 3.4.1.3 below: Case 3.4.1.3: X is a 2-block and $LM(X) = 0$. In this case, X consists of two different pixel values:

$$x_{\sigma(1)} = ... = x_{\sigma(n-k)} < x_{\sigma(n-k+1)} = ... = x_{\sigma(n)}$$

Compute:

$$dmax = x_{\sigma(n)} - x_{\sigma(n-k)}$$

3.4.1.3.1. If $dmax \geq 2$, no bit embedded, k largest pixels are increased by 1, remaining pixels are decreased by 1:

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} + 1, i = n-k+1, ..., n,$$

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} - 1, i = 1, ..., n-k.$$

3.4.1.3.2. If $dmax = 1$, k bits are embedded in k largest pixels as follows:

1) If all k bits are equal to 1, then pixels are modified as in 2.4.1.3.1:

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} + 1, i = n-k+1, ..., n,$$

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} - 1, i = 1, ..., n-k.$$

2) If all k bits are equal to 0, then k largest pixels are increased by 1:

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} + 1, i = n-k+1, ..., n.$$

3) If k bits include both 0 and 1: k bits are added to k largest pixels:

$$x_{\sigma(i)}^{'} = x_{\sigma(i)} + b_{i-n+k}, i = n-k+1, ..., n.$$

Except for case 2.4.1.3, other cases are not changed.

b) Embedding data in smallest pixels

In this subsection we revise case 2.4.2.1 of the iGePVO-K method by Case 3.4.2.1 below, and do not modify the remaining cases.
Case 3.4.2.1: X is 2-block:

$$x_{\sigma(1)} = ... = x_{\sigma(l)} < x_{\sigma(l+1)} = ... = x_{\sigma(n)}$$

Compute:

$$dmin = x_{\sigma(1)} - x_{\sigma(l+1)}$$

From subsection 3.2.1 it follows that

$$dmin \leq -2$$

So, no formula is performed. In short, ignore this case and do nothing.

TABLE I
MAXIMUM PAYLOAD COMPARISON BETWEEN METHODS

Images	PVO [18]	IPVO [19]	PVO-K [20]	GePVO-K [21]	iGePVO-K [22]	Proposed
Airplane	38454	53055	51505	56305	58092	61566
Baboon	13146	13462	13940	14546	14621	14661
Barbara	29490	48066	40202	41008	47400	51775
Cabeza	54263	71946	73416	79198	78031	81389
Cameraman	43458	71097	63378	69771	72663	96500
Car	31378	47327	41594	44644	51599	56457
Chest	45781	63502	60470	64151	65287	69117
Lena	32108	38713	38951	41417	41895	42709
Phone	40855	58176	52699	53986	58566	62478
Things	33772	49418	44221	43473	51179	55094
Average	36271	51476	48038	50850	53933	59175

TABLE II
PSNR COMPARISON BETWEEN METHODS WITH PAYLOAD = 10000

Images	PVO [18]	IPVO [19]	PVO-K [20]	GePVO-K [21]	iGePVO-K [22]	Proposed
Airplane	63.134	63.463	60.612	55.610	58.252	61.058
Baboon	54.306	54.298	54.428	52.999	54.228	54.290
Barbara	63.893	64.316	60.554	55.040	58.807	62.180
Cabeza	61.942	61.299	60.164	54.296	56.741	59.144
Cameraman	64.043	63.580	60.412	54.593	57.162	61.075
Car	63.257	64.224	60.517	56.280	59.274	62.069
Chest	63.624	63.280	60.630	55.143	57.642	60.517
Lena	60.110	60.056	59.009	55.066	57.726	58.349
Phone	63.798	63.582	60.552	54.926	58.267	61.066
Things	63.071	63.955	60.326	54.475	58.839	61.756
Average	62.118	62.205	59.720	54.843	57.694	60.150

TABLE III
PSNR COMPARISON BETWEEN METHODS WITH PAYLOAD = 20000

Images	PVO [18]	IPVO [19]	PVO-K [20]	GePVO-K [21]	iGePVO-K [22]	Proposed
Airplane	59.018	59.614	57.355	53.324	56.226	57.521
Baboon						
Barbara	59.299	60.835	57.329	52.959	56.671	58.411
Cabeza	58.951	58.179	57.053	52.273	54.886	56.134
Cameraman	60.984	60.883	57.736	53.007	55.787	57.900
Car	58.042	60.707	57.088	53.451	57.248	59.032
Chest	60.068	59.861	57.556	53.128	55.857	57.307
Lena	56.431	56.551	55.651	52.228	54.730	55.010
Phone	59.728	60.217	57.365	52.974	56.199	57.607
Things	58.649	60.076	57.111	52.338	56.485	57.939
Average	59.019	59.658	57.138	52.854	56.010	57.429

TABLE IV
PSNR COMPARISON BETWEEN METHODS WITH PAYLOAD = 30000

Images	PVO [18]	IPVO [19]	PVO-K [20]	GePVO-K [21]	iGePVO-K [22]	Proposed
Airplane	56.21	57.150	55.263	51.563	54.600	55.34
Baboon						
Barbara		58.403	55.041	51.177	54.762	56.027
Cabeza	57.183	56.632	55.265	50.916	53.570	54.321
Cameraman	58.929	59.002	56.017	51.822	54.611	56.204
Car	53.281	57.314	54.535	51.309	54.865	56.526
Chest	57.685	57.790	55.670	51.588	54.360	55.288
Lena	53.441	54.066	53.347	50.327	52.731	52.901
Phone	56.612	57.902	55.316	51.308	54.492	55.473
Things	54.933	57.436	54.820	50.665	54.568	55.516
Average	56.034	57.299	55.030	51.186	54.284	55.288

3. Extracting and restoring algorithm

a) Extracting and restoring in smallest pixels

We only need to revise case 2.4.3.3 by Case 3.4.3.3 as follows

Case 3.4.3.3: $X^{'}$ is 2-block:

$$x^{'}_{\sigma(1)} = ... = x^{'}_{\sigma(l)} < x^{'}_{\sigma(l+1)} = ... = x^{'}_{\sigma(n)}$$

Compute

$$dmin = x^{'}_{\sigma(1)} - x^{'}_{\sigma(l+1)}.$$

(a) If $dmin \leq -4$, no bit is extracted, l smallest pixels are increased by 1, other pixels are decreased by 1

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

$$x_{\sigma(i)} = x^{'}_{\sigma(i)} - 1, i = l + 1, ..., n.$$

(b) If $dmin = -3$, all extracted l bits are equal to 1 and restoring n pixels is carried out as in (a).

(c) If $dmin = -2$, all extracted l bits are equal to 0 and $n - l$ largest pixels are decreased by 1:

$$x_{\sigma(i)} = x^{'}_{\sigma(i)} - 1, i = l + 1, ..., n.$$

b) Extracting and restoring in the largest pixels

The extracting and restoring algorithm in the largest pixels of the iGePVO-K method (subsection 2.4.4) consists of two subsections: 2.4.4.1 and 2.4.4.2. In subsection 2.4.4.1, X is a 2-block. For this case, extracting embedded bits and restoring the original block are performed in subsection 3.3.1. So here, nothing to do. We only need to perform steps in 2.4.4.2. After completing the steps in subsections 3.3.1 and 3.3.2, obtain the embedded bits and restore the original.

4. Illustrating examples

a) Example 1

Given a pixel block

$$B = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix},$$

and two bits 1, 1.

3.4.1.1 Embedding process

(a) **Preprocessing**: Scanning row by row, convert B into a sequence$X = (24, 25, 25, 24)$. Sort X in ascending order to get:

$$X_\sigma = (24, 24, 25, 25), \sigma = (1, 4, 2, 3).$$

In this case $n = 4, k = 2$, and $dmax = 1$.

(b) **Embedding in largest pixels**: Since $dmax = 1$, and all bits are equal to 1, we have case 3.4.1.3.2 (a). In this case, largest pixels are increased by 1, other pixels are decreased 1, so:

$$X^{'}_\sigma = (23, 23, 26, 26).$$

(c) **Embedding in smallest pixels**: Compute

$$dmin = 23 - 26 = -3.$$

So by subsection 3.2.2, case 3.4.2.1, nothing to do. In summary, we get

$$X^{'}_\sigma = (23, 23, 26, 26)$$

(d) **Creating a marked block**: Rearrange $X^{'}_\sigma$ according to positions $\sigma = (1, 4, 2, 3)$, we have

$$X^{'} = (23, 26, 26, 23).$$

Converting $X^{'}$ into the block, it follows that the marked block of B is

$$B^{'} = \begin{bmatrix} 23 & 26 \\ 26 & 23 \end{bmatrix}.$$

3.4.1.2 Extracting and restoring process

Given

$$B^{'} = \begin{bmatrix} 23 & 26 \\ 26 & 23 \end{bmatrix}.$$

(a) **Preprocessing**: Similar to 3.4.1(a), from $B^{'}$ we have:

$$X^{'}_\sigma = (23, 23, 26, 26), \sigma = (1, 4, 2, 3).$$

Compute:

$$dmin = 23 - 26 = -3$$

(b) **Extracting in smallest pixels**: Because $dmin = -3$, so by 3.3.1 (b) it follows that two embedded bits are equal to 1. The largest pixels are decreased by 1, other pixels are increased by 1. As a result, we get:

$$X_\sigma = (24, 24, 25, 25)$$

(c) **Extracting in the largest pixels**: According to 3.3.2, nothing to do. Thus we have:

$$X_\sigma = (24, 24, 25, 25)$$

(d) **Restoring the original block**: Rearrange X_σ by positions in σ, we have:

$$X = (24, 25, 25, 24).$$

Convert X into a block. As a result, we get the original block:

$$B = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}.$$

b) Example 2

Given a pixel block

$$B = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix},$$

and two bits 0, 0.

3.4.2.1 Embedding process

In Example 1, two embedded bits are equal to 1. But in this example, two embedded bits are equal to 0. So in 3.4.1.1 (b), we consider 3.4.1.3.2 (b) instead of 3.4.1.3.2 (a). That

means, only increasing the largest pixels by one. Thus, we obtain:

$$X_{\sigma}^{'} = (24, 24, 26, 26)$$

So,

$$B^{'} = \begin{bmatrix} 24 & 26 \\ 26 & 24 \end{bmatrix}.$$

3.4.2.2 Extracting and restoring process
In Example 1,

$$dmin = -3.$$

But in this example,

$$dmin = 24 - 26 = -2.$$

So by 3.3.1 (c), it follows that two embedded bits are equal to 0, and the largest pixels are decreased by 1. As a result, we get

$$X_{\sigma} = (24, 24, 25, 25)$$

From this, deduce that the original B is equal to

$$B = \begin{bmatrix} 24 & 25 \\ 25 & 24 \end{bmatrix}.$$

IV. Experimental results

To illustrate the results of the theoretical analysis above, we perform experiments on 8 standard grayscale images of size 512×512, they are shown in Fig. 2 for comparing 6 methods: PVO, IPVO, PVO-K, GePVO-K, iGePVO-K, and proposed method. The embedded data is a random binary bit sequence generated by the Matlab function Randi. The programs are written in the Matlab platform and run on the IdeaPad S410p Lenovo computer.

1. Data hiding capacity comparison

Table I presents the maximum payloads of methods in test images. The payload (or data hiding capacity) is equal to the embedding capacity minus the length of the compression code of the location map. Table I shows that the data-hiding capacity of the proposed method is always greater than that of all other relative methods.

2. Stego image quality comparison

The stego image quality comparison of 6 methods in 8 test images is performed on data sets of length 10000 bits (Table II), 20000 bits (Table III) and 30000 bits (TableIV). From these tables, we can conclude that the stego image quality of the proposed always is high. Specifically, the stego image quality is larger 59 dB for 10000 bits, larger 56 dB for 20000 bits, and larger 54 dB for 30000 bits. The proposed has stego image quality lower than PVO and IPVO methods but higher than GePVO-K, iGePVO-K and PVO-K methods.

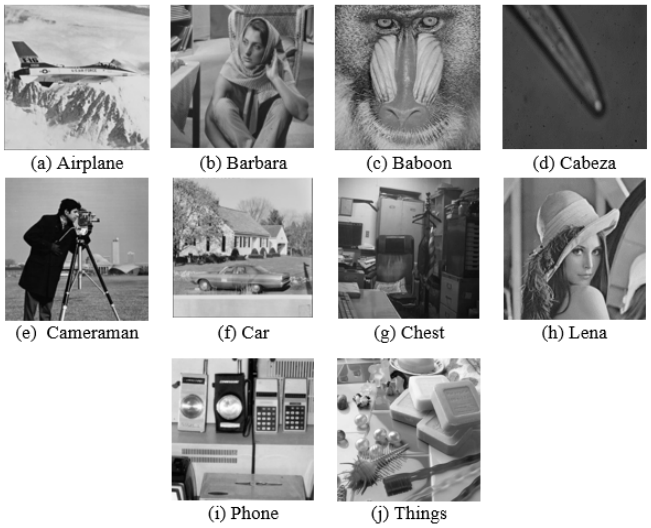

Figure 2. Experimental images

V. Conclusion

The the iGePVO-K method is an extension of PVO-K and GePVO-K methods. It has the embedding capacity larger than PVO-K and GePVO-K methods. However, this method still has a drawback in that it must use a flag bit sequence. Since the flag bit sequence needs to be embedded with the data in the original image, this reduces both its embedding capacity and its image quality. In this paper, we propose some improvements in data embedding formulas of iGePVO-K method to discard the flag bit sequence. So, the proposed method not only has larger embedding capacity but also has better stego image quality than iGePVO-K method. The experimental results have shown that the embedding capacity of the proposed method is higher than the relative existing methods while maintaining good stego image quality.

References

[1] M. Celik, G. Sharma, A. Tekalp, and E. Saber, "Lossless generalized-lsb data embedding," *IEEE Transactions on Image Processing*, vol. 14, no. 2, pp. 253–266, 2005.

[2] M. Celik, G. Sharma, and A. Tekalp, "Lossless watermarking for image authentication: a new framework and an implementation," *IEEE Transactions on Image Processing*, vol. 15, no. 4, pp. 1042–1049, 2006.

[3] J. Fridrich, M. Goljan, and R. Du, "Invertible authentication," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 4314, pp. 197–208, 08 2001.

[4] J. J. Fridrich, M. Goljan, and R. Du, "Lossless data embedding—new paradigm in digital watermarking," *EURASIP Journal on Advances in Signal Processing*, vol. 2002, pp. 185–196, 2002.

[5] 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, 2003.

[6] A. Alattar, "Reversible watermark using the difference expansion of a generalized integer transform," *IEEE Transactions on Image Processing*, vol. 13, no. 8, pp. 1147–1156, 2004.
[7] L. Kamstra and H. Heijmans, "Reversible data embedding into images using wavelet techniques and sorting," *IEEE Transactions on Image Processing*, vol. 14, no. 12, pp. 2082–2090, 2005.
[8] W.-L. Tai, C.-M. Yeh, and C.-C. Chang, "Reversible data hiding based on histogram modification of pixel differences," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 6, pp. 906–910, 2009.
[9] D. M. Thodi and J. J. Rodriguez, "Expansion embedding techniques for reversible watermarking," *IEEE Transactions on Image Processing*, vol. 16, no. 3, pp. 721–730, 2007.
[10] Y. Hu, H.-K. Lee, and J. Li, "De-based reversible data hiding with improved overflow location map," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 2, pp. 250–260, 2009.
[11] X. Li, B. Yang, and T. Zeng, "Efficient reversible watermarking based on adaptive prediction-error expansion and pixel selection," *IEEE Transactions on Image Processing*, vol. 20, no. 12, pp. 3524–3533, 2011.
[12] Z. Ni, Y.-Q. Shi, N. Ansari, and W. Su, "Reversible data hiding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 3, pp. 354–362, 2006.
[13] X. Chen, S. Xingming, H. Sun, L. Xiang, and Y. bin, "Histogram shifting based reversible data hiding method using directed-prediction scheme," *Multimedia Tools and Applications*, vol. 74, pp. 5747–5765, 02 2015.
[14] S. Kukreja, S. S. Kasana, and G. Kasana, "Histogram based multilevel reversible data hiding scheme using simple and absolute difference images," *Multimedia Tools Appl.*, vol. 78, no. 5, p. 6139–6162, 2019.
[15] X. Ma, Z. Pan, S. Hu, and L. Wang, "High-fidelity reversible data hiding scheme based on multi-predictor sorting and selecting mechanism," *Journal of Visual Communication and Image Representation*, vol. 28, pp. 71–82, 01 2015.
[16] B. Ou, X. Li, W. Li, and Y.-Q. Shi, "Pixel-value-ordering based reversible data hiding with adaptive texture classification and modification," *Digital Forensics and Watermarking*, vol. 11378, pp. 169–179, 2019.
[17] X. Wang, J. Ding, and Q. Pei, "A novel reversible image data hiding scheme based on pixel value ordering and dynamic pixel block partition," *Inf. Sci.*, vol. 310, p. 16–35, 2015.
[18] X. Li, J. Li, B. Li, and B. Yang, "High-fidelity reversible data hiding scheme based on pixel-value-ordering and prediction-error expansion," *Signal Process.*, vol. 93, no. 1, p. 198–205, jan 2013.
[19] F. Peng, X. Li, and B. Yang, "Improved pvo-based reversible data hiding," *Digit. Signal Process.*, vol. 25, no. C, p. 255–265, feb 2014.
[20] B. Ou, X. Li, Y. Zhao, and R. Ni, "Reversible data hiding using invariant pixel-value-ordering and prediction-error expansion," *Image Commun.*, vol. 29, no. 7, p. 760–772, aug 2014.
[21] 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, 01 2018.
[22] N. K. Sao, N. N. Hoa, and P. V. At, "An effective reversible data hiding method based on pixel-value-ordering," *Journal of Computer Science and Cybernetics*, vol. 36, no. 2, p. 139–158, May 2020.

Le Kinh Tai received B.A. degree from Academy of Finance in 2006, degree of master's in human resource management from Trade Union University in 2012, and B.A degree in English from Ha Noi Open University in 2014. He currently works in Trade Union University. His interests are data hiding and information security.
Email: tailk@dhcd.edu.vn

Nguyen Ngoc Hung received his B.E. degree from University of Transport and Communications, in 2009. He achieved M.E. degree from Vietnam National University of Engineering and Technology, in 2014. His interests are data hiding, watermarking, information security.
Email: nnhung@ioit.ac.vn

Dang Tran Duc received his B.E., M.E. degree from University of Transport and Communications, in 2009, 2015 respectively. Currently, he is CEO of VVECO Joint Stock Company. His interests are AI, NLP, and information security.
Email: tranducdang@vveco.vn

Cao Duc Diep received his B.E., M.E. degree from University of Transport and Communications, in 2009, 2015 respectively. Currently, he is a software project management. His interests are software engineering, and information security.
Email: diepcd@epu.edu.vn

Pham Minh Thai received his bachelor's degree in information technology from Nha Trang University in 2008. He received a master's degree in computer science from the Military Technical Academy in 2011. His interests are data hiding, watermarking, information security.
Email: pmthai@uneti.edu.vn