Information Sciences 310 (2015) 16–35

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A novel reversible image data hiding scheme based on pixel value ordering and dynamic pixel block partition

Xiang Wang, Jing Ding, Qingqi Pei *

State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China

ARTICLE INFO

Article history:
Received 23 September 2014
Received in revised form 14 February 2015
Accepted 10 March 2015
Available online 17 March 2015

Keywords:
Reversible data hiding
Pixel-value ordering
Dynamic block partition
Prediction-error expansion

ABSTRACT

Recently, various efficient reversible data-hiding schemes based on pixel value ordering have been proposed for embedding messages into high-fidelity images. In these schemes, after dividing the cover image into equal-sized blocks, the pixels within a given block are ordered according to their values, and data embedding is achieved by modifying the maximum and minimum values of each block. For a given embedding capacity, the optimal block size is exhaustively searched so that the embedding distortion is minimized. These pixel value ordering-based schemes perform fairly well, especially for low embedding capacity. However, to obtain a larger embedding capacity, a smaller block size should be used, which usually leads to a dramatic quality degradation of the marked image. In this paper, to address this drawback and to enhance the performance of pixel value ordering-based embedding further, a novel reversible data hiding method is proposed. Instead of using equal-sized blocks, a dynamic blocking strategy is used to divide the cover image adaptively into various-sized blocks. Specifically, flat image areas are preferentially divided into smaller blocks to retain high embedding capacity, whereas rough areas are divided into larger blocks to avoid decreasing peak signal-to-noise ratio. As a result, the proposed scheme can provide a larger embedding capacity than current pixel value ordering-based schemes while keeping distortion low. The superiority of the proposed method is also experimentally verified.

1. Introduction

Because of its ability to eliminate embedding distortion, reversible data hiding (RDH), also known as lossless data embedding, is increasingly attracting the attention of both academic and industrial researchers. The RDH decoder, unlike that of traditional data hiding schemes, can recover the cover image exactly as well as extracting the embedded data [3,24,31,40,41]. Therefore, RDH is usually used for copyright protection and data hiding of sensitive images, such as military and medical images. The performance of RDH schemes is evaluated on two conflicting criteria: embedding capacity (EC) and embedding distortion. In general, the goal of RDH is to minimize embedding distortion for a given EC.

Over the past few years, many different approaches have been proposed to improve RDH embedding performance. Examples include lossless compression-based algorithms [4,5,14,15], where the embedding space is produced by losslessly compressing a particular part of the cover image; the difference expansion (DE) technique [43], where the pixel difference is expanded to embed data; and the prediction-error expansion (PEE) technique [42], where the prediction error instead of the

* Corresponding author. Tel.: +86 29 88202927; fax: +86 29 88204909.
E-mail addresses: wangxiang@xidian.edu.cn (X. Wang), jding@stu.xidian.edu.cn (J. Ding), qqpei@xidian.edu.cn (Q. Pei).

http://dx.doi.org/10.1016/j.ins.2015.03.022
0020-0255/© 2015 Elsevier Inc. All rights reserved.

pixel difference is used for expansion embedding. In addition to hiding secret data in the spatial domain, another branch of reversible data hiding has been explored for use in the frequency or compressed domain, such as JPEG images [22,47] and vector quantization (VQ)-compressed images [8,25,32,46].

Recently, several PEE-based RDH methods, collectively called pixel value ordering (PVO) [28,37], have been proposed. In these methods, pixels within a block are ordered according to their values, and the prediction error expansion to hide data is calculated by using the second largest/smallest pixel to predict the maximum/minimum pixel. Because pixels in the same block usually have a strong correlation, these methods produce remarkable embedding performance. However, in PVO schemes [28,37], the cover image is divided into equal-sized blocks, and the PVO embedding performance is significantly related to block size. Using larger block size produces a sharper histogram, but obviously decreases EC. To achieve a higher EC, it is necessary to use a smaller block size, which usually results in significant image quality degradation. In addition, changing the block size causes a performance fluctuation on peak signal-to-noise ratio (PSNR), which is also experimentally demonstrated in the following sections (Fig. 5). For PVO-based schemes, determining the appropriate block size for a given EC is an important issue. Specifically, both of these PVO methods simply use exhaustive search to find the optimal block size.

This paper proposes a dynamic image-division strategy that extends PVO-based schemes to improve embedding performance, especially for relatively large EC. In the proposed method, instead of using equal-sized blocks, the size of each block is adaptively determined according to the context information and the EC. For a smaller EC, only smooth areas of the cover image are used to embed data, and a larger block size can be chosen. Compared with PVO-based methods, the proposed method performs better in most cases. To obtain a larger EC, data are embedded into both smooth and textured areas, but only smooth regions are divided into smaller blocks in the proposed method, and a larger block size can still be used in textured regions. Specifically, 2×2 and 4×4 blocks are used in smooth and textured regions respectively. Previous PVO-based methods divided the whole cover image (including both smooth and textured regions) into smaller blocks. Experimental results verify that the performance of the proposed method is superior to the Peng et al. method with optimal fixed block sizes and also to some state-of-the-art approaches; the superiority of the proposed method is more noticeable for large EC.

The structure of this paper is as follows. In Section 2, related studies, including lossless compression-based methods such as DE and PEE techniques, are briefly reviewed. In Section 3, the PVO-based scheme proposed by Li et al. [28] and its extension proposed by Peng et al. [37] are introduced in detail. The proposed improvement is presented in Section 4. The detailed implementation of the proposed method, including the embedding and extraction procedures and the treatment of underflow and overflow, is described in Section 5. Experimental results compared with those from PVO-based methods [28,37] and some state-of-the-art research are reported in Section 6. Finally, the last section presents conclusions.

2. Related work

Many RDH methods have been proposed in the literature. Early RDH methods used lossless compression algorithms to create a reversible embedding space [4,5,14,15]. However, these methods cannot provide a high EC even using multi-pass embedding. Tians DE technique [43], which can provide a much higher EC, is the cornerstone of most recent RDH methods. By using the pixel difference to embed data, DE takes better advantage of local correlations within the image than algorithms that directly use lossless compression algorithms. Because two pixels are used to embed one bit of data, the EC of DE is bounded by 0.5 bit per pixel (bpp). Alattar [1] and Wang et al. [48] generalized the DE technique to blocks of arbitrary size rather than pixel pairs by using reversible integer transforms. As a result, the EC is improved from DEs 0.5 bpp to almost 1 bpp. Another benefit of the integer transform is that it can be easily extended to transform domains, such as the DCT domain [7] and the wavelet domain [2,26]. Note that DE embedding can generate pixels with values smaller than 0 (underflow) or larger than 255 (overflow). These so-called overflow and underflow problems are inevitable in both Tians original DE technique and its extensions. The location mapping technique is widely used to restrict the embedded pixels to the range of $[0, 255]$. Usually, the location map is a binary matrix used to label the pixel pairs of the cover image. In the location map, 1 means that the corresponding pixel block can be used for embedding without causing overflow or underflow problems, and 0 indicates that using the pixel block may cause overflow or underflow. To ensure reversibility, the location map should be embedded together with the data and should be first extracted by the decoder to locate the pixel pairs carrying the hidden data. However, the location map usually occupies a huge portion of the embedding space even when compressed.

Inspired by Tians DE technique, Thodi et al. [42] used the prediction error instead of the pixel difference for expansion embedding. This expansion embedding technique is usually called PEE. The prediction error is the difference between the pixel and its predicted value computed according to the context. Compared with DE, PEE can take advantage of image correlation in a larger local image region to achieve better performance [6,11,13,19,20,44,45]. The classical PEE model includes two main steps: prediction and histogram shifting. In the latter, data hiding is achieved by expanding the histogram bins around 0, shifting the other bins to create vacant space. The performance of histogram shifting and data hiding depend strongly on the prediction results. A better predictor produces a sharper histogram, i.e., a histogram with higher bins around 0. As a result, with a better predictor, the PEE shifts fewer pixels for the same EC, and hence embedding distortion is reduced. Therefore, various efficient predictors have been proposed to improve PEE performance, such as MED (median edge detector, [42,50]), GAP (gradient adjusted prediction, [36,51]), SGAP (simplified GAP, [9]), rhombus prediction [39], PDE prediction (partial differential equation, [35]), interpolation techniques [33], AGSP (accurate gradient selection predictor, [27]), and EGBSW (extended gradient-based selective weighting, [12]).

The properties inherent in the pixel intensity histogram can also be used to improve embedding performance. Ni et al. [34] proposed a remarkable RDH scheme using the peak and zero points of the image histogram. The peak points of the image histogram are used to embed data using expansion embedding, and the zero points are used to reduce the number of shifted bins. Although with the method proposed by Niet al., distortion caused by embedding is low, the EC is fairly limited as well (e.g., approximately 5500 bits for the standard 512×512 gray-scale image Lena). Various solutions have been proposed to increase the EC of the Ni et al. method, such as constructing a sharper histogram [16,44] or reducing the size of auxiliary information [21,18].

Sorting is an effective technique for both DE and PEE. Kamstra and Heijmans [23] first improved Tians DE method by sorting pixel pairs according to the local variance. After sorting, only smooth pixel pairs were selected for data embedding. Another benefit of sorting is that the location map of sorted pixel pairs consists primarily of 1s and can be compressed markedly using entropy coding. Therefore, the size of auxiliary information is reduced, improving embedding performance. More recently, Sachnev et al. [39] introduced sorting to PEE and proposed a PEE method incorporating an embedding-position selection strategy. This method has been experimentally verified to be superior to most contemporary RDH schemes, and its advantage is mainly due to the sorting technique. According to this line of thinking, embedding information into smooth regions of the cover image has proved to be very helpful to embedding performance [10,17,18,29,30,38].

Instead of expanding the difference of a pixel pair (DE) or the prediction error (PEE), Li et al. proposed an algorithm which expands the difference of the largest or smallest pixel in a block. In this method, the cover image is divided into equal-sized blocks. Pixels in each block are then ordered according to their values. The difference to be expanded to hide data is calculated by using the second largest/smallest pixel to predict the maximum/minimum pixel. Because the pixels of a block are first sorted to find the maximum and the minimum, this method is called pixel value ordering (PVO). Peng et al. [37] later extended the work of Li et al. by constructing a new and different histogram, further improving the performance of this approach.

3. Preliminaries

In this section, as a review, the PVO-based RDH scheme presented by Li et al. [28] and its extension proposed by Peng et al. [37] are briefly introduced.

3.1. PVO scheme of Li et al.

The PVO-based PEE scheme, presented by Li et al., modifies the maximum and minimum of a pixel block to create embedding space which can embed sufficient payload into the cover image with low distortion. This subsection briefly introduces this method, but only the maximum-modification-based embedding stage is presented, with the minimum-modification-based one being omitted.

The PVO-based scheme first divides the cover image into non-overlapping equal-sized blocks. Given a block $X = \{x_1, \ldots, x_n\}$, sort its value $(x_1, \ldots, x_n)$ in ascending order to obtain $(x_{\sigma(1)}, \ldots, x_{\sigma(n)})$, where $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ is a unique one-to-one mapping such that: $x_{\sigma(1)} \leqslant \ldots \leqslant x_{\sigma(n)}, \sigma(i) < \sigma(j)$ if $x_{\sigma(i)} = x_{\sigma(j)}$ and $i < j$. Then the second-largest value, $x_{\sigma(n-1)}$, is used to predict the maximum $x_{\sigma(n)}$. The corresponding prediction error is

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

After the prediction errors of all blocks have been calculated, a histogram consisting of $PE_{\max}$ is generated. Because $PE_{\max}$ is always positive, this histogram is defined on the interval $[0, 255]$.

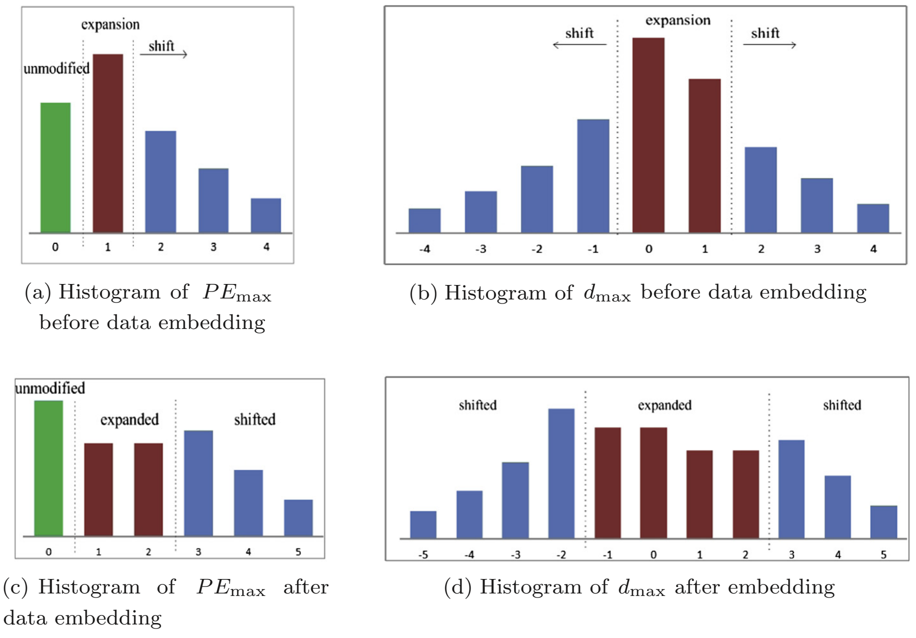
Note that bin 1 (i.e., the bin with $PE_{\max} = 1$) is usually the histogram peak. This means that the scheme proposed by Li et al. simply takes this bin as an inner region for expansion embedding. To ensure reversibility, bins larger than 1 are taken as an outer region and are shifted to make embedding space. As shown in Fig. 1(a) and (b), bin 1 (marked in red) is expanded to embed data, and bin 0 (marked in green) remains unmodified, whereas blue bins (larger than 1) are shifted. In this situation, the maximum $x_{\sigma(n)}$ is modified as follows:

$$\widetilde{x}_{\sigma(n)} = \begin{cases} x_{\sigma(n)}, & \text{if } PE_{\max} = 0, \\ x_{\sigma(n)} + b, & \text{if } PE_{\max} = 1, \\ x_{\sigma(n)} + 1, & \text{if } PE_{\max} > 1, \end{cases} \tag{2}$$

where $b \in \{0, 1\}$ is the data bit to be embedded. Let the marked value of X be denoted by $(y_1, \ldots, y_n)$. Then $y_i = x_i$ for every $i \neq \sigma(n)$, and $y_{\sigma(n)} = \widetilde{x}_{\sigma(n)}$. Note that the pixel value order preserves consistency before and after embedding, i.e., the embedded pixels $y_{\sigma(1)}, \ldots, y_{\sigma(n-1)}$ are also in ascending order, which ensures the reversibility of the PVO-based PEE scheme.

On the decoder side, after sorting $(y_1, \ldots, y_n)$ in ascending order to obtain $y_{\sigma(1)}, \ldots, y_{\sigma(n)}$, the prediction error of $y_{\sigma(n)}$ can be computed as:

$$\widetilde{PE}_{\max} = y_{\sigma(n)} - y_{\sigma(n-1)}. \tag{3}$$



(a) Histogram of $PE_{\max}$ before data embedding (b) Histogram of $d_{\max}$ before data embedding

(c) Histogram of $PE_{\max}$ after data embedding (d) Histogram of $d_{\max}$ after embedding

Fig. 1. Histogram of $PE_{\max}$ and $d_{\max}$ before and after data embedding.

As shown in Fig. 1(c), only when $\widetilde{PE}_{\max} \in \{1,2\}$ does $\widetilde{PE}_{\max}$ contain data. Accordingly, the specific decoding procedure is as follows:

- if $\widetilde{PE}_{\max} \in \{1,2\}$, the embedded data bit is $b = \widetilde{PE}_{\max} - 1$, and the original value is recovered as $y_{\sigma(n)} - b$.
- if $\widetilde{PE}_{\max} > 2$, there is no bit embedded in the prediction error, and the original value is recovered as $y_{\sigma(n)} - 1$.
- if $\widetilde{PE}_{\max} = 0, y_{\sigma(n)}$ remains at its original value, and there is no hidden bit.

Similarly to the maximum, if the minimum is decreased or unchanged, the mapping order σ remains unchanged. Inspired by this, Li et al. also embedded data by modifying the minimum. For the minimum, the difference $PE_{\min}$ is defined by $PE_{\min} = x_{\sigma(1)} - x_{\sigma(2)}$, and then a histogram defined on the interval $[0, -\infty)$ can be generated. In this histogram, the highest bin -1 is used for expansion to hide data. Bins smaller than -1 are shifted to create vacant space for expansion, and bin 0 remains unchanged. For more detailed procedures for embedding and extraction, the reader can refer to [28].

Fig. 2 presents an example of the embedding process for this PVO-based method. The 2×2 block is sorted to yield a 1×4 vector $(171, 172, 189, 189)$. The PE_{max} and PE_{min} of this vector are 0 and -1 respectively. According to (2), PE_{max} should remain unchanged, whereas PE_{min} can be used to embed one bit of data. Suppose that the embedded data in 1. PE_{min} are changed to $PE_{min} + 1 = -2$. Then the modified vector after embedding is $\{170, 172, 189, 189\}$. Finally, this vector is reverse-sorted to retrieve the embedded 2×2 block.

In this method, flat blocks are preferentially used to embed data, whereas rough ones are left unchanged. To achieve this goal, for each block, its complexity is calculated as $NL = x_{\sigma(n-1)} - x_{\sigma(2)}$. A block is classified as flat if its complexity is less than a predefined threshold T, and other blocks with complexities greater than or equal to T are considered rough. In the embedding process, only flat blocks are used to embed data because these blocks usually produce small prediction errors. On the

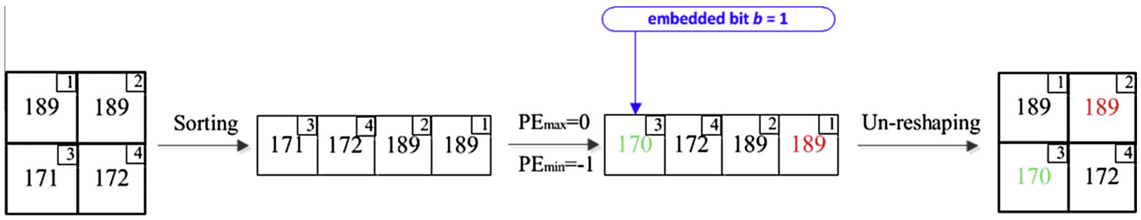


Fig. 2. Embedding example for the method of Li et al.

other hand, because $x_{\sigma(n-1)}, x_{\sigma(2)}$, and the mapping σ remain unchanged after data embedding, the decoder can achieve the same complexity used by the encoder. This guarantees correct data extraction and lossless image restoration.

3.2. PVO-based RDH scheme of Peng et al.

Although the PVO-based method proposed by Li et al. achieves remarkable embedding performance [28], Peng et al. indicated that PVO still has some deficiencies and can be improved [37].

Reviewing Fig. 1(a), bin 0, which is usually the second-highest bin, is not used for data embedding. PVO performance could be further improved if bin 0 were used to embed data. To this end, by designing a new prediction difference, Peng et al. brought bin 0 into the embedding scheme and produced a higher EC than the original PVO. On the other hand, for a given EC, this method can use larger blocks for embedding, meaning that it can take better advantage of image redundancy to yield a higher PSNR.

The notations defined in SubSection 3.1 are used here for descriptive purposes. Similarly to [28], the first step is to sort the block and to obtain $\{x_{\sigma(1)}, \ldots, x_{\sigma(n)}\}$. However, the prediction error used here, unlike $PE_{\max}$ in (1), is calculated as:

$$d_{\max} = x_u - x_v, \tag{4}$$

where

$$\begin{cases} u = \min(\sigma(n), \sigma(n-1)), \\ v = \max(\sigma(n), \sigma(n-1)). \end{cases} \tag{5}$$

The histogram of $d_{\max}$ is shown in Fig. 1(b). Bins 0 and 1 are used for embedding, and other bins are shifted to create embedding space. Then the marked value of the maximum $x_{\sigma(n)}$ is determined by:

$$\tilde{x}_{\sigma(n)} = \begin{cases} x_{\sigma(n)} + b, & \text{if } d_{\max} = 1, \\ x_{\sigma(n)} + 1, & \text{if } d_{\max} > 1, \\ x_{\sigma(n)} + b, & \text{if } d_{\max} = 0, \\ x_{\sigma(n)} + 1, & \text{if } d_{\max} < 0, \end{cases} \tag{6}$$

where $b \in \{0, 1\}$ is the embedded data. The histogram of the marked image is showed in Fig. 1(d). Note that blocks with $x_{\sigma(n)} = x_{\sigma(n-1)}$, which remain unchanged in the method of Li et al., are used here to embed data, and the highest bin of the histogram of $d_{\max}$ is usually bin 0 (which is bin 1 in [28]). In addition, because this method uses prediction error instead of absolute error as in [28] for embedding, the histograms of $PE_{\max}$ and $d_{\max}$ have the following correspondence:

$$\begin{cases} PE_{\max}(0) = d_{\max}(0), \\ PE_{\max}(k) = d_{\max}(k) + d_{\max}(-k). \end{cases} \tag{7}$$

Note that only $PE_{\max}(1)$ is used for expansion in [28], whereas the method proposed here uses $d_{\max}(0)$ and $d_{\max}(1)$ for embedding. Because $PE_{\max}(1) < d_{\max}(0) + d_{\max}(1)$, this method can produce a higher EC than [28]. Specifically, the EC growth is $d_{\max}(0) + d_{\max}(1) - PE_{\max}(1) \approx d_{\max}(0) - PE_{\max}(1)/2$.

In the decoder, the marked block is first sorted to obtain $y_{\sigma(1)}, \ldots, y_{\sigma(n)}$. Because the embedding process modifies the block only by increasing the maximum pixel or decreasing the minimum pixel, the pixel order σ before and after embedding remains unchanged. This property ensures the reversibility of the PVO-based method. The marked prediction error can be calculated as:

$$\tilde{d}_{\max} = y_u - y_v, \tag{8}$$

where u and v are defined identically to (4). If $\tilde{d}_{\max} \in [-1, 2]$, according to (6), $\tilde{d}_{\max}$ carries one bit of data. Otherwise, it is a shifted block. Accordingly, the decoding process is:

- If $\tilde{d}_{\max} > 0$, the decoder identifies $y_u > y_v$. Therefore, $\sigma(n) < \sigma(n-1), u = \sigma(n)$ and $v = \sigma(n-1)$.
  - If $\tilde{d}_{\max} \in \{1, 2\}$, the embedded data bit is $b = \tilde{d}_{\max} - 1$, and the original maximum is $x_{\sigma(n)} = y_u - b$.
  - If $\tilde{d}_{\max} > 2$, no data are embedded in the maximum, and the original maximum is $x_{\sigma(n)} = y_u - 1$.
- If $\tilde{d}_{\max} \leqslant 0$, the decoder obtains $y_u \leqslant y_v$. Therefore, $\sigma(n) > \sigma(n-1), u = \sigma(n-1)$ and $v = \sigma(n)$.
  - If $\tilde{d}_{\max} \in \{0, -1\}$, the embedded data bit is $b = -\tilde{d}_{\max}$, and the original maximum is $x_{\sigma(n)} = y_v - b$.
  - If $\tilde{d}_{\max} < -1$, no data are embedded in the maximum, and the original maximum is $x_{\sigma(n)} = y_v - 1$.

The corresponding minimum-modification-based data embedding scheme is similar to the maximum-based scheme described above. The prediction error is calculated accordingly:

$$d_{\min} = x_s - x_t, \tag{9}$$

where

$$\begin{cases} s = \min(\sigma(1), \sigma(2)), \\ t = \max(\sigma(1), \sigma(2)). \end{cases} \tag{10}$$

The marked value of the minimum $x_{\sigma(1)}$ is determined as:

$$\tilde{x} = \begin{cases} x_{\sigma(1)} - b, & \text{if } d_{\min} = 1, \\ x_{\sigma(1)} - 1, & \text{if } d_{\min} > 1, \\ x_{\sigma(1)} - b, & \text{if } d_{\min} = 0, \\ x_{\sigma(1)} - 1, & \text{if } d_{\min} < 0. \end{cases} \tag{11}$$

In the decoder, based on the fact that the mapping σ remains unchanged, data extraction and image restoration according to the sorted marked values $(y_1, \ldots, y_n)$ can be accomplished. Define the following:

$$\tilde{d}_{\min} = y_s - y_t, \tag{12}$$

where (s, t) is defined in (10).

- If $\tilde{d}_{\min} > 0$, then $y_s > y_t$. Therefore, $\sigma(1) > \sigma(2), s = \sigma(2)$ and $t = \sigma(1)$
  - if $\tilde{d}_{\min} \in \{1, 2\}$, the hidden data bit is $b = \tilde{d}_{\min} - 1$ and the original minimum is $x_{\sigma(1)} = y_t + b$.
  - if $\tilde{d}_{\min} > 2$, there are no hidden data in the minimum, and the original minimum is $x_{\sigma(1)} = y_t + 1$.
- If $\tilde{d}_{\min} \leqslant 0$, then $y_s \leqslant y_t$. Therefore, $\sigma(1) < \sigma(2), s = \sigma(1)$ and $t = \sigma(2)$
  - if $\tilde{d}_{\min} \in \{0, -1\}$, the hidden data bit is $b = -\tilde{d}_{\min}$, and the original minimum is $x_{\sigma(1)} = y_s + b$.
  - if $\tilde{d}_{\min} < -1$, there are no hidden data in the minimum, and the original minimum is $x_{\sigma(1)} = y_s + 1$.

Moreover, the block complexity defined in the method of Li et al. is used here to find the smooth blocks.

Fig. 3 presents a simple example of this method. Given a 2×2 block $(155, 159, 158, 158)$, sorting its pixels in ascending order generates $(155, 158, 158, 159)$, where $\sigma(1) = 1,\ \sigma(2) = 3,\ \sigma(3) = 4,\ and\ \sigma(4) = 2$. Therefore, $d_{\max} = x_u - x_v = 159 - 158 = 1$. To embed $b = 1$, the marked value is changed to $\tilde{x} = x_{\sigma(4)} + b = 160$. For the minimum, $d_{\min} = x_s - x_t = 155 - 158 = -3$. According to (11), the minimum should be shifted to $\tilde{x} = x_{\sigma(1)} - 1 = 154$.

The decoding process is the reverse of the encoding process. The marked pixel values in the decoder are $(y_1, \ldots, y_4) = (154, 160, 158, 158)$. Sorting them in ascending order generates $(154, 158, 158, 160)$, where $\sigma(1) = 1, \sigma(2) = 3, \sigma(3) = 4, \sigma(4) = 2$. Note that the pixel orders in the decoder and in the encoder are identical. The prediction error is calculated as $\tilde{d}_{\max} = x_u - x_v = 160 - 158 = 2$. Therefore, the hiding bit $b = \tilde{d}_{\max} - 1 = 1$, and the original value $x_\sigma(4) = 160 - b = 159$. For the minimum, $d_{\min} = x_s - x_t = 154 - 158 = -4$, which means that there is no hidden bit. The original value is recovered as $x_\sigma(1) = 154 + 1 = 155$. Finally, the marked block is recovered as $(155, 159, 158, 158)$.

3.3. Performance comparison

Fig. 4 presents a comparison between the PVO method proposed by Li et al. [28], the PVO extension proposed by Peng et al. [37], the classical DE proposed by Tian [43], and the PEE proposed by Thodi et al. (PE2) [42]. The test images were Lena and Airplane. It is clear that PVO-based methods can offer much better performance than classical DE and PEE.

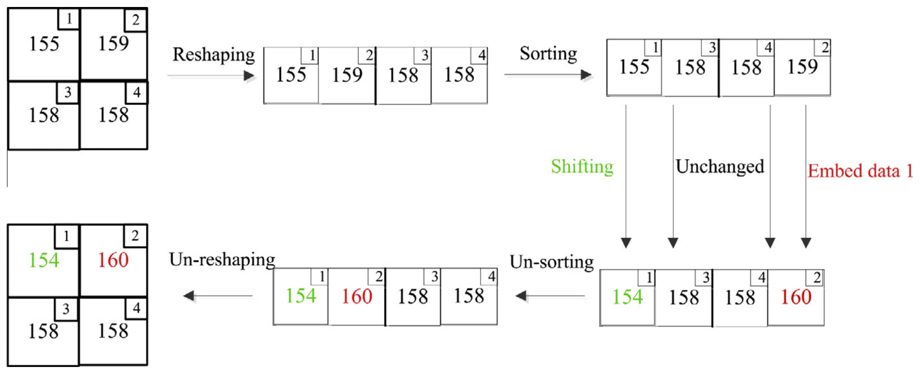


Fig. 3. Embedding example for the method of Peng et al.

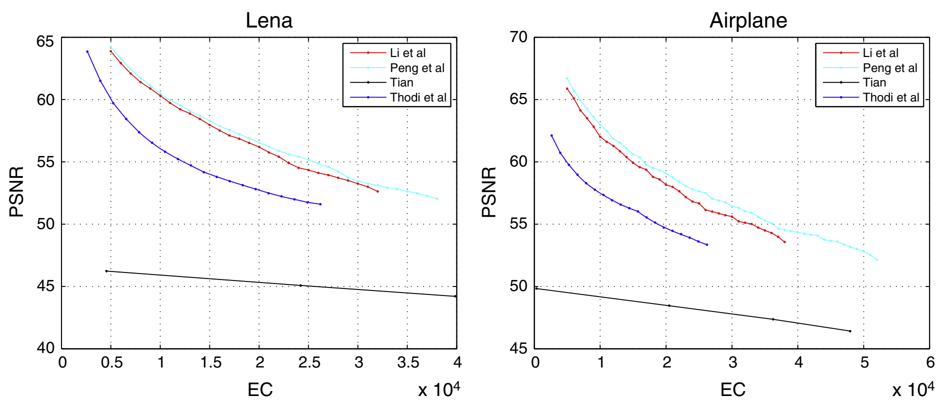


Fig. 4. Performance comparison between the Li et al. method, [28], the Peng et al. method [37], Tian DE [43], and Thodi et al. PEE [42] methods.

Specifically, the increase in PSNR is nearly 4 dB over PEE and nearly 9 dB over DE. In addition, the method of Peng et al. achieved slightly better performance than PVO.

4. Proposed scheme

In PVO-based RDH methods, the embedding process is executed block by block. The embedding performance of PVO-based methods [28,37] is significantly related to block size. As shown in Fig. 4 of [28], using larger blocks generates better PSNR for low EC, whereas using smaller blocks gives better performance for high EC. This property can also be observed in the method of Peng et al. [37]. Fig. 5 shows that the PSNR using a 4×4 block is 64.02 dB for an EC of 5000 bits, whereas the PSNR using a 3×3 block is 63.44 dB for the same EC. When the EC increases to $12,000$ bits, the PSNR using a 4×4 block is 58.75 dB, which is less than the 59.44 dB obtained using a 3×3 block. Clearly, a relatively larger block size, such as 4×4, works better on small EC, whereas a smaller block size is more suitable for high EC. In addition, the maximum EC provided by different block sizes is rather different. Obviously, using a smaller block provides a relatively high maximum EC.

Because different block sizes yield different performance for the same EC, to achieve the best embedding performance, the PVO-based methods repeatedly use the embedding procedure with a $n_1 \times n_2$ block, where $n_1, n_2 \in \{2, 3, 4, 5\}$. The block size producing the highest PSNR is finally chosen to embed data. The optimal block size for PVO is calculated by exhaustive search, which is time-consuming and not adaptively suited for different EC. Usually, the optimal size for a high EC is smaller than that for a low EC. However, although using a larger block size generates a relatively smaller EC, it can yield a much higher PSNR. The reason for this property is that the difference between the second largest/smallest pixel and the maximum/minimum pixel tends to be smaller in a larger block. In textured areas, the advantage of using a larger block size is obvious, but the performance improvement decreases in smooth areas. This occurs because the pixels in smooth areas have similar pixel values, and therefore the prediction process in the small block is accurate enough. This means that if a smaller

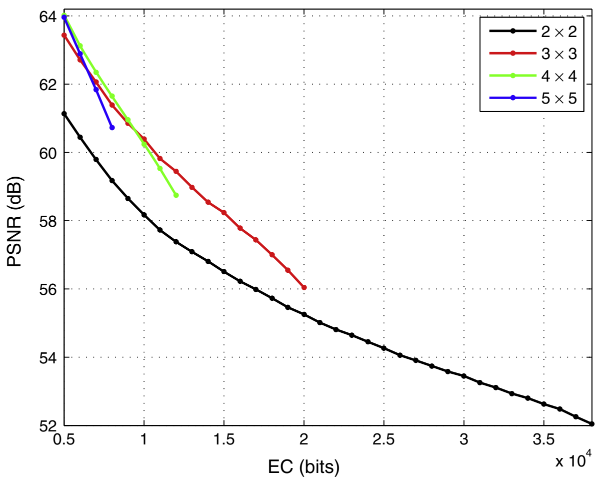


Fig. 5. Performance of the scheme proposed by Peng et al. with different block sizes for the Lena image.

block size can be used in smooth areas to achieve a high EC and a larger block size can be used in textured areas to calculate a small prediction error to keep the PSNR high, PVO performance will be further improved.

To this end, the proposed method uses two block sizes in the embedding process: a 4×4 block for textured areas, and a 2×2 block for smooth areas. For high EC, the smaller block size is adaptively applied only to smooth areas instead of to the whole cover image, and the larger block size is used in other areas.

Block complexity, which is used in the PVO-based method, is also used in the proposed method to locate flat blocks. Similarly, *NL* is used to denote the complexity of a block. The detailed definition of *NL* and the proof of its consistency property are presented later. In [28,37], the complexity is used only to find the embedding blocks. In the proposed method, *NL* can be used to estimate the textural complexity of a block as well as to locate embedding blocks. Therefore, according to *NL*, it is possible to choose different block sizes for blocks with different degrees of textural complexity.

Specifically, the proposed method uses two thresholds, T_1, T_2 $(-1 \leqslant T_2 \leqslant T_1 \leqslant 255)$, to differentiate the blocks in smooth areas from others. According to the complexity *NL* of a 4×4 block *X* and the thresholds T_1 and T_2, three situations must be taken into account.

- Case 1: when $NL > T_1$, block *X* is classified as a rough block. In this situation, block *X* is probably in a textured area, and such rough blocks are excluded from the embedding process. This means that the pixels of *X* remain unchanged, and no data are embedded into *X*.
- Case 2: when $T_2 < NL \leqslant T_1$, block *X* is denoted as a normal block.
  - In this situation, *X* is in a moderately smooth area and can be used for embedding. However, such an area is not suitable for using a small block size, i.e., not suitable for further division into 2×2 sub-blocks, because the sub-blocks of *X* are not smooth enough for embedding. Therefore, two bits of data (one bit for maximum and one for minimum) are directly embedded into block *X*. Similarly to the method of Peng et al., *X* is first sorted in ascending order, and the ascending array is expressed as $\{x_{\sigma(1)}, \ldots, x_{\sigma(16)}\}$. The procedure for modifying the maximum and minimum of the block is as shown in (6) and (11).
  - On the decoder side, after sorting the normal blocks, $\widetilde{d}_{\max}$ and $\widetilde{d}_{\min}$ are computed based on (8) and (12) respectively. Then, according to the decoding methods described in Section 2.2, data extraction and image restoration can be accomplished.
- case 3: when $NL \leqslant T_2$, block *X* is denoted as a flat block.
  - In this situation, *X* is in a smooth area. *X* can therefore be further divided into four 2×2 blocks to embed more data bits. For each 2×2 block, its pixels are sorted in ascending order, and the differences $d_{\max}$ and $d_{\min}$ are computed based on (4) and (9). Then the data can be embedded using (6) and (11). Because each 2×2 block carries 2 bits of data, 8 bits of data can be embedded into a flat 4×4 block.
  - The decoder first divides a flat block into four 2×2 sub-blocks. Then the pixels in each marked 2×2 block are sorted in ascending order, and $\widetilde{d}_{\max}$ and $\widetilde{d}_{\min}$ are calculated based on (8) and (12). Finally, the decoder can perform data extraction and image restoration using the methods described in Section 2.2.

To ensure reversibility, the complexity *NL*, whether for a flat, normal, or rough block, should preserve consistency between the encoder and the decoder. Because the block size is dynamically adjusted in the proposed method, the calculation of complexity in the proposed method is different than in previous PVO-based methods. This discussion will first introduce how to compute the complexity *NL* of the block $X = \{x_1, x_2, \ldots, x_N\}$. It will then prove that *NL* remains unchanged after data embedding. *X* is divided into four sub-blocks, which are denoted as $S_k = \{x_1^k, \ldots, x_n^k\}$, where $k \in \{1, 2, 3, 4\}$ and $n = N/4$. Sorting the four sub-blocks in ascending order yields:

(i) $S_1 : (x_1^1, \ldots, x_n^1) \rightarrow (x_{\sigma_1(1)}^1, \ldots, x_{\sigma_1(n)}^1)$.
(ii) $S_2 : (x_1^2, \ldots, x_n^2) \rightarrow (x_{\sigma_2(1)}^2, \ldots, x_{\sigma_2(n)}^2)$.
(iii) $S_3 : (x_1^3, \ldots, x_n^3) \rightarrow (x_{\sigma_3(1)}^3, \ldots, x_{\sigma_3(n)}^3)$.
(iv) $S_4 : (x_1^4, \ldots, x_n^4) \rightarrow (x_{\sigma_4(1)}^4, \ldots, x_{\sigma_4(n)}^4)$.

where for $k \in \{1, 2, 3, 4\}$, each σ_k is a one-to-one mapping from $\{1, \ldots, n\}$ to $\{1, \ldots, n\}$. The complexity is defined as:

$$NL = max\{x_{\sigma_1(n-1)}^1, x_{\sigma_2(n-1)}^2, x_{\sigma_3(n-1)}^3, x_{\sigma_4(n-1)}^4\} - min\{x_{\sigma_1(2)}^1, x_{\sigma_2(2)}^2, x_{\sigma_3(2)}^3, x_{\sigma_4(2)}^4\}. \tag{13}$$

It will now be proved that the complexity *NL* of a block has the same value in the encoder and the decoder.

(1) If *X* is a rough block, it will not be modified in the embedding process, and its complexity *NL* clearly remains unchanged.
(2) If *X* is a normal block, the maximum and minimum of block *X* are modified to embed data. Suppose that block *X* sorted in ascending order is $\{x_{\sigma(1)}, \ldots, x_{\sigma(n)}\}$. Obviously,

$$\sigma(1) = min\{\sigma_1(1), \sigma_2(1), \sigma_3(1), \sigma_4(1)\}, \tag{14}$$

$$\sigma(N) = max\{\sigma_1(n), \sigma_2(n), \sigma_3(n), \sigma_4(n)\}. \tag{15}$$

According to (11), $x_{\sigma(1)}$ is either modified to $x_{\sigma(1)} - 1$ or remains unchanged. Then according to (14), the minimum of the four sub-blocks is either decreased or unchanged, which means that the minimum pixel of the sub-block is still the minimum after embedding. By a similar analysis, it can be determined that the maximum pixel of the sub-block is still the maximum after embedding. Therefore, the complexity of a normal block remains unchanged based on (13).

(3) If X is a flat block, each maximum and minimum of sub-block S_k will be modified to embed data. In this situation, $x^k_{\sigma_k(1)}$ will become $x^k_{\sigma_k(1)} - 1$ or remain unchanged, and $x^k_{\sigma_k(n)}$ will be modified to $x^k_{\sigma_k(n)} + 1$ or remain unchanged. It is easy to determine that $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ retain the same order, and therefore NL remains unchanged.

An illustration of a complexity calculation is shown in Fig. 6. The 4×4 block X is first divided into four 2×2 sub-blocks S_1, S_2, S_3, and S_4. Then the sub-blocks are sorted in ascending order, and the second-largest and second-smallest pixel values (marked in green) are selected to calculate NL.

To illustrate this idea more clearly, embedding and decoding examples for a normal block and a flat block are presented here. In these examples, $T_1 = 4$ and $T_2 = 2$. In Fig. 7, the original 4×4 block X is first divided into four 2×2 sub-blocks, which are $(154, 156, 152, 153), (154, 158, 153, 149), (157, 150, 155, 153)$, and $(155, 152, 152, 152)$. Then, according to the calculation process of NL (Fig. 6 and (13)), the complexity of this 4×4 block is $max\{154, 154, 155, 152\} - min\{153, 153, 153, 152\} = 3$. Because $T_2 < 3 < T_1, X$ is a normal block, and as mentioned earlier, a normal block uses a 4×4 block size for embedding. Therefore, the 16 pixels of X are sorted in ascending order, and the index $\sigma = (8, 10, 5, 12, 13, 15, 16, 6, 7, 14, 1, 3, 11, 13, 2, 9, 4)$. According to (6) and (11), $d_{max} = x_{min(\sigma_{16},\sigma_{15})} - x_{max(\sigma_{16},\sigma_{15})} = x_4 - x_9 = 1$, and $d_{min} = x_{min(\sigma_1,\sigma_2)} - x_{max(\sigma_1,\sigma_2)} = x_8 - x_{10} = -1$. For $d_{max} = 1$, (6) can be used to embed one bit of data. As shown in Fig. 7, the next bit 1 is embedded into d_{max} by $\widetilde{d}_{max} = d_{max} + 1 = 2$. Accordingly, the maximum pixel $x_{\sigma_{16}}$ is modified to $\widetilde{x}_{\sigma_{16}} = x_{\sigma_{15}} + |\widetilde{d}_{max}| = x_{\sigma_{16}} + b_1 = 158 + 1 = 159$. For $d_{min} < 0$, the minimum pixel should be shifted to $\widetilde{x}_{\sigma(1)} = x_{\sigma(1)} - 1 = 149 - 1 = 148$ according to (11), and no data are embedded into $x_{\sigma(1)}$. In the decoder, according to Fig. 7(b), the pixel order is unchanged after embedding, and therefore the complexity of the block also remains unchanged. It can be determined that NL is still 3, and therefore this block is a normal block. Hence, only the maximum and minimum pixels of the 4×4 block are modified. Similarly, $d'_{\max} = 2 \in \{1, 2\}$, and $d'_{\min} = -2 \notin \{-1, 0\}$. Then, for the maximum, a bit 1 is extracted, and the restored pixel is $x'_{\max} - \widetilde{b_1} = 158$. For the minimum, the shifted pixel is recovered as $x_1 + 1 = 149$.

Fig. 8 presents an example of a flat block. The embedding procedure is shown in Fig. 8(a). As in the previous example, the complexity of the original block is first calculated; in this example, the complexity is $NL = 1$. Because the complexity of the original block X is less than the given threshold T_2, X is a flat block. Therefore, X is divided into four 2×2 sub-blocks for embedding. For the first sub-block $XS_1 = (155, 157, 153, 153)$ whose index in X is $(1, 2, 5, 6)$, sorting yields $(153, 153, 155, 157)$. Note that the two equal pixel values 153 are sorted according to their index. Therefore, the sorted index σ_1 of S_1 is $(5, 6, 1, 2)$. As shown in the gray box, its prediction errors can be calculated as $d_{max} = -2$ and $d_{min} = 0$. Hence, according to (6) and (11), the maximum adds 1 to make room for expanding ($\widetilde{x}_{\sigma_1(4)} = x_{\sigma_1(4)} + 1 = 158$), and the minimum minus $b_1 = 1$ is to hide b_1 ($\widetilde{x}_{\sigma_1(1)} = x_{\sigma_1(1)} - b_1 = 152$). For the second sub-block, its $d_{max} = -2$ and $d_{min} = -2$, and therefore both the maximum and minimum pixels should be shifted: $\widetilde{x}_{\sigma_2(4)} = x_{\sigma_2(4)} + 1 = 158$ and $\widetilde{x}_{\sigma_2(1)} = x_{\sigma_2(1)} - 1 = 151$. d_{max} and d_{min} of the third sub-block are 1 and 0 respectively. Therefore, $b_2 = 1$ and $b_3 = 0$ are embedded into XS_3 by $\widetilde{x}_{\sigma_3(4)} = x_{\sigma_3(4)} + b_2 = 157$ and $\widetilde{x}_{\sigma_3(1)} = x_{\sigma_3(1)} - b_3 = 154$. For the last sub-block $XS_4, d_{max} = 1$ and $d_{min} = -1, b_4 = 1$ are embedded into the maximum by $\widetilde{x}_{\sigma_4(4)} = x_{\sigma_4(4)} + b_4 = 156$, and the minimum is shifted by $\widetilde{x}_{\sigma_4(1)} = x_{\sigma_4(1)} - 1 = 152$.

In the decoder, the complexity is unchanged after the embedding procedure, as shown in Fig. 8(b). For the marked sub-block YS_1, its sorted array is $(152, 153, 155, 158)$, and the corresponding index is $(5, 6, 1, 2)$, which is exactly identical to the original order. Then, $\widetilde{d}_{\max} = 3 \notin \{1, 2\}$ can be calculated. Hence, this block is shifted in the encoder, and the original value should be $YS_{\sigma_1(4)} - 1 = 157$. Because the minimum $\widetilde{d}_{\min} = -1 \in \{0, -1\}$, the hidden bit $b_1 = -\widetilde{d}_{\min} = 1$ is extracted, and the

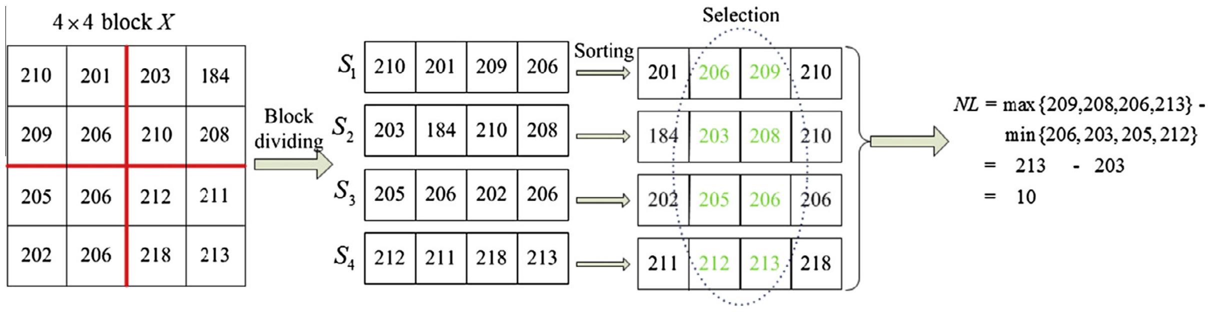
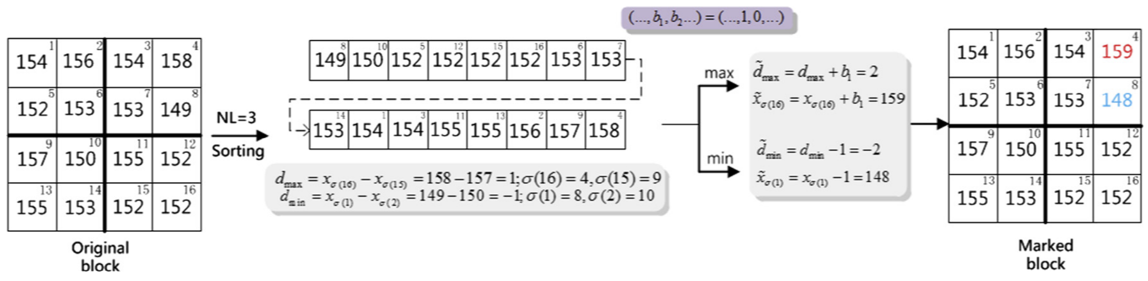


Fig. 6. Illustration of complexity calculation.


(a) Embedding example for a normal block

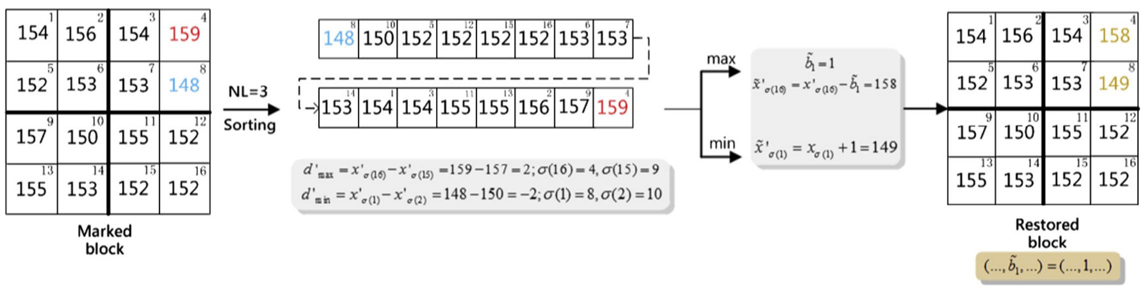

(b) Extraction example for a normal block

Fig. 7. Example of a normal block. Thresholds are defined as $T_1 = 4, T_2 = 2$. The pixel index is always marked in the top right corner. Pixels to be expanded are marked in red, those to be shifted are in blue, and restored pixels are in green. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

original pixel value is recovered as $YX'_{\sigma_1(1)} + b_1 = 153$. Similarly, it is possible to extract $b_2 = 1$ and $b_3 = 0$ from YS_3 and $b_4 = 1$ from YS_4, and to proceed to recover the rest of the marked sub-blocks.

Note that the scheme presented by Peng et al. [37] is a special case of the proposed method. If $T_1 = T_2$, all 4×4 blocks will be divided into 2×2 sub-blocks. In this case, the proposed method is the same as the Peng et al. scheme of taking 2×2 blocks. When $T_2 = -1$, the 4×4 block is directly used to embed data, i.e., is not divided into sub-blocks, and therefore the proposed method with $T_2 = -1$ is identical to the Peng et al. scheme with 4×4 blocks.

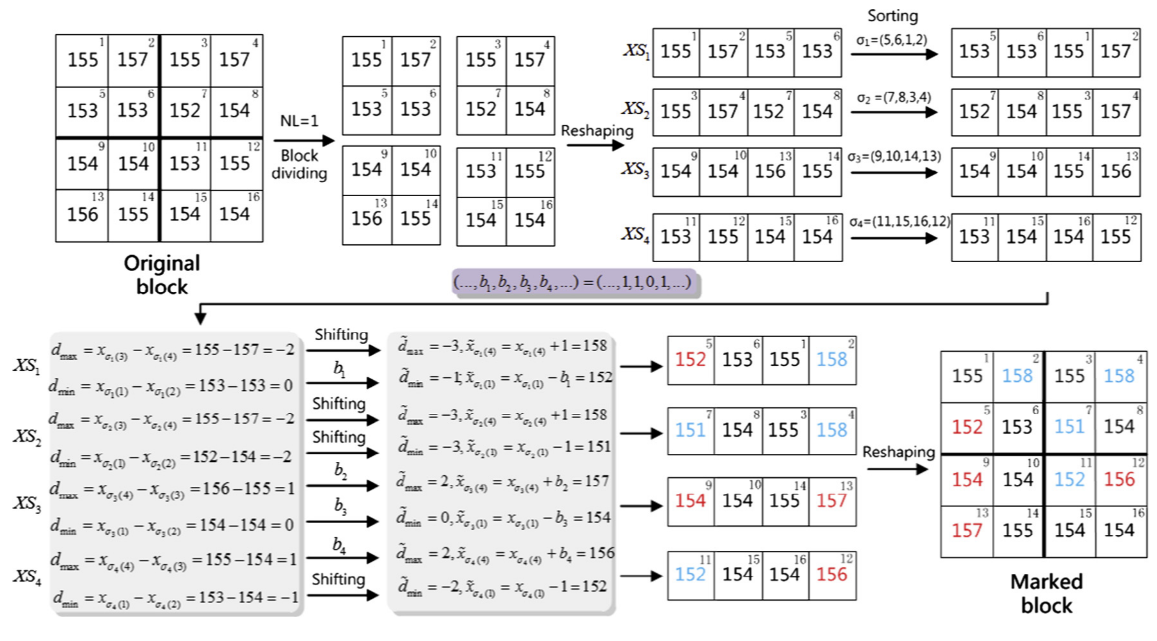
5. Implementation of the proposed scheme

This section presents the proposed method step by step, including the embedding and extraction process and the construction of the location map. Fig. 9 presents a flowchart of the embedding and decoding process in the proposed method. First, the proposed dynamic block data embedding procedure will be introduced.

Step 1 (Image partition)
Divide the cover image into non-overlapping 4×4 blocks $\{X_1, \ldots, X_\xi\}$. For each block X_i $(i \in \{1, \ldots, \xi\})$, label it with its value in raster scanning order.
Step 2 (Location map construction)
This step defines the overflow/underflow location map LM. For each block X_i with values $(x_1, \ldots, x_{16})$, if $min\{x_1, \ldots, x_{16}\} = 0$ or $max\{x_1, \ldots, x_{16}\} = 255$, the block is an exceptional block that would cause overflow or underflow, and $LM_i = 1$. Otherwise, $LM_i = 0$. Consequently, LM is a binary sequence of length ξ. Then LM is losslessly compressed using arithmetic coding, and the length of the compressed location map is denoted as l_{clm}.
Step 3 (Data embedding)
The data embedding procedure is shown in this step, for $i \in \{1, \ldots, \xi\}$.

- If $LM_i = 0$ and $NL_i \leqslant T_2, X_i$ is a flat block. Its four equal-sized sub-blocks $XS_k, k \in \{1,2,3,4\}$ are also regarded as flat blocks. In this case, four equal sub-blocks are used to embed data without causing overflow or underflow. The maximum and minimum of $XS_k, k \in \{1,2,3,4\}$, are shifted or expanded to embed data according to (6) and (11).
- If $LM_i = 0$ and $T_2 < NL_i \leqslant T_1, X_i$ is a normal block. The maximum and minimum of X_i are shifted or expanded to embed data according to (6) and (11).
- If $LM_i = 0$ and $NL_i \geqslant T_1, X_i$ is a rough block and is unsuitable for embedding, and therefore nothing is done with X_i.
- If $LM_i = 1$, modifying X_i may cause an overflow or underflow problem, and therefore X_i remains unchanged.

This step continues until all data bits have been embedded; ξ_{end} is the index of last data-carrying block.


(a) Embedding example for a flat block

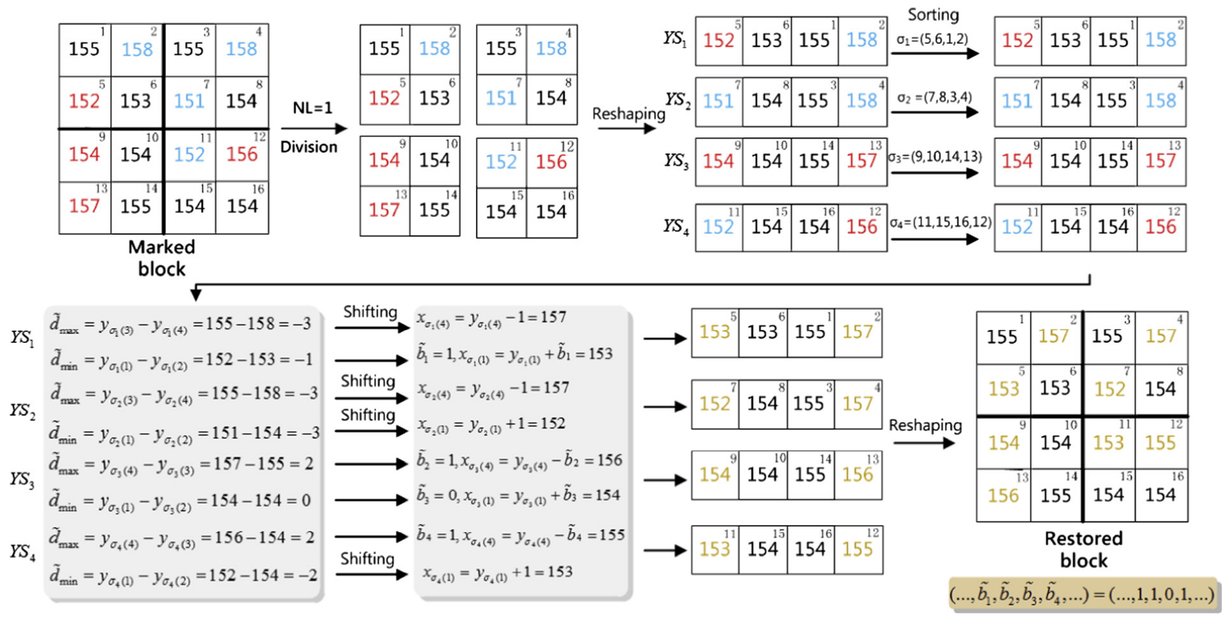
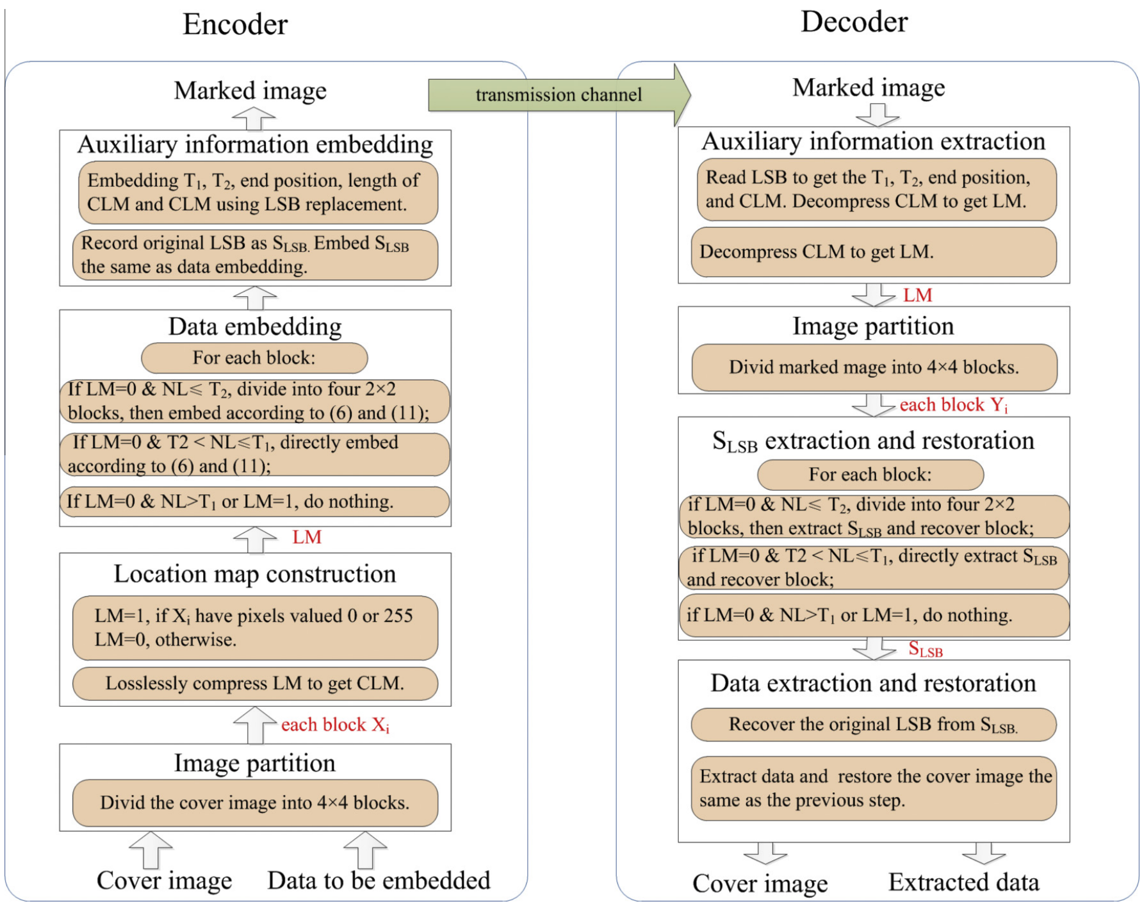

(b) Extraction example for a flat block

Fig. 8. Example of a flat block. Pixels to be expanded are marked in red, those to be shifted are in blue, and restored pixels are in green. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Step 4 (Auxiliary information and location map embedding)
Record LSBs of the first $16 + 2\lceil \log_2(16\xi)\rceil + l_{\mathrm{clm}}$ pixels to obtain a binary sequence S_{LSB}, where $\lceil\cdot\rceil$ is the ceiling function. Then replace these LSBs by the following auxiliary information and the compressed location map defined in *Step 2*:

- the block complexity threshold T_1 (8 bits), T_2 (8 bits),
- the end position ξ_{end} ($\lceil \log_2(16\xi)\rceil$ bits),
- the length of the compressed location map l_{clm} ($\lceil \log_2(16\xi)\rceil$ bits),
- the compressed location map (l_{clm} bits).



Encoder
Decoder
Marked image
transmission channel
Marked image
Auxiliary information embedding
Embedding T1, T2, end position, length of CLM and CLM using LSB replacement.
Record original LSB as SLSB. Embed SLSB the same as data embedding.
Data embedding
For each block:
If LM=0 & NL⩽ T2, divide into four 2×2 blocks, then embed according to (6) and (11);
If LM=0 & T2 < NL⩽T1, directly embed according to (6) and (11);
If LM=0 & NL>T1 or LM=1, do nothing.
LM
Location map construction
LM=1, if Xi have pixels valued 0 or 255 LM=0, otherwise.
Losslessly compress LM to get CLM.
each block Xi
Image partition
Divid the cover image into 4×4 blocks.
Cover image
Data to be embedded
Auxiliary information extraction
Read LSB to get the T1, T2, end position, and CLM. Decompress CLM to get LM.
Decompress CLM to get LM.
LM
Image partition
Divid marked mage into 4×4 blocks.
each block Yi
SLSB extraction and restoration
For each block:
if LM=0 & NL⩽ T2, divide into four 2×2 blocks, then extract SLSB and recover block;
if LM=0 & T2 < NL⩽T1, directly extract SLSB and recover block;
if LM=0 & NL>T1 or LM=1, do nothing.
SLSB
Data extraction and restoration
Recover the original LSB from SLSB.
Extract data and restore the cover image the same as the previous step.
Cover image
Extracted data

Fig. 9. Flowchart of the proposed method.

Finally, embed the sequence S_{LSB} into the remaining blocks $\{X_{\xi_{end}+1}, \ldots, X_{\xi}\}$ using the same method in *Step 3* to obtain the marked image.

As the inverse process of data embedding, data extraction and image recovery are described below.

Step 1 (Auxiliary information and location map extraction)
Read the LSBs of the first $16 + 2\lceil \log_2(16\xi) \rceil$ pixels of the marked image to retrieve the auxiliary information, including the values of T_1, T_2, ξ_{end} and l_{clm}. Then read the LSBs of the next l_{clm} pixels to retrieve the compressed location map. Recover the location map LM by decompressing the compressed location map.
Step 2 (Sequence S_{LSB} extraction and image restoration)
First, the marked image is divided and labeled as shown in Section 3.1 *Step 1*. The complexity of a block is calculated using (13). In the following procedure, the sequence S_{LSB} defined in Section 3.1 *Step 4* will be extracted from the block sequence $\{X_{\xi_{end}+1}, \ldots, X_{\xi}\}$. For $i \in \{\xi_{end} + 1, \ldots, \xi\}$:

- If $LM_i = 0$ and $NL_i \leqslant T_2$, the block is a flat block. For each of its sub-blocks, according to the decoding methods described in Section 2.4, the data can be extracted and the original values recovered from $YS_{\sigma_k(1)}$ and $YS_{\sigma_k(4)}$, where $k \in \{1, 2, 3, 4\}$.
- If $LM_i = 0$ and $T_2 < NL_i \leqslant T_1, Y_i$ is a normal block. According to the decoding methods described in Section 2.4, the data can be extracted and the original values recovered from $Y_{\sigma(1)}$ and $Y_{\sigma(16)}$.
- Otherwise, the block is a rough block. Therefore, it contains no hidden data, and the maximum and minimum are unchanged.

Step 3 (Data extraction and image restoration)
Replace the LSBs of the first $16 + 2\lceil \log_2(16\xi) \rceil + l_{clm}$ image pixels by the sequence S_{LSB} extracted in *Step 2*. Then the hidden data in $\{Y_1, \ldots, Y_{\xi_{end}}\}$ can be extracted and the cover image retrieved by the same method as in *Step* 2.

Note that in the proposed method, data embedding with all thresholds $-1 \leqslant T_1 \leqslant T_2 \leqslant 255$ is attempted, and the maximum PSNR of each combination of T_1, T_2 is recorded. The thresholds T_1, T_2 with which the embedding achieves the best performance for the given EC are used to embed data. Parameters T_1, T_2, and the compressed location map size l_{CLM} for embedding 10,000 bits and 20,000 bits of data are presented in Table 2. According to Table 2, it is clear that there is no general law relating T_1 and T_2.

6. Experimental results

This section presents experimental results from the proposed method. The performance of the proposed method will be compared with those of the Peng et al. [37], Sachnev et al. [39], and Li et al. (TIFS) [30] schemes. Eight 512 × 512 grayscale images, Airplane, Elaine, Barbara, Baboon, Boat, Lake, Lena, and Peppers, were selected as test images. Except for Barbara, all images were downloaded from the USC-SIPI database.

In Fig. 10, each case described in Section 3 (i.e., rough, normal, and flat blocks) is marked in a different color for a given EC: rough blocks are marked in red, normal blocks in yellow, and flat blocks in green. In Fig. 10(b), (c) and (d), the thresholds are the optimal combination after searching every available value. Smooth areas were preferentially used for embedding. With increasing EC, textured areas are gradually involved in the embedding process, such as the hair area in Lena. Fig. 10(e) presents an example of using sub-optimal thresholds, where $T_1 = 23$ and $T_2 = 20$ are used, and a maximum EC larger than 30,000 bits can be produced. Compared with the optimal threshold that yields a similar EC (Fig. 10(c)), it was found that the optimal threshold classified the blocks of the cover image more accurately. Based on the observations from Fig. 10 and Table 2, the following two conclusions can be drawn:

- ► The complexity *NL* used here can reflect the complexity of a block exactly.
- ► The selection of thresholds T_1, T_2 depends on the given EC and the cover image, and there is no general relationship between T_1 and T_2.

Hence, in these experiments, all combinations of (T_1, T_2) were tested. Computation times for each test image are listed in Table 1. The computation times mentioned in this table were measured on an Intel Haswell CPU (i5-4570, 3.2 GHz) Windows 7 PC with 8.00 GB RAM. The proposed method spent the longest time on Elaine (33.79 s) and the shortest time on Baboon (27.64 s) due to its high block complexity. The average running time for the eight images was 32.45 s.

Fig. 11 presents a comparison between the Peng et al. PVO scheme and the proposed approach on Lena. For clarity, only the performance of the Peng et al. method with 2 × 2 and 4 × 4 blocks is presented. It will become apparent later that even if this PVO scheme uses the optimal block size obtained by exhaustive search, the proposed method still produces better performance.

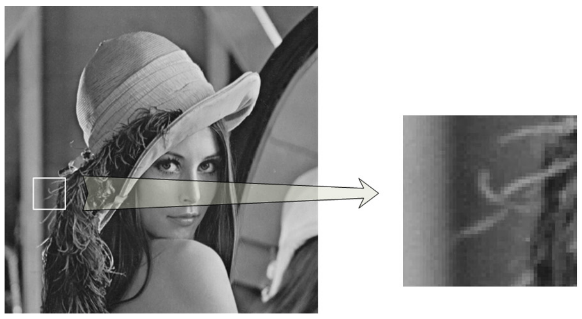
In Fig. 11, if EC is less than 13,000, the Peng et al. PVO uses 4 × 4 blocks for embedding, and the proposed method achieves a PNSR increase of 1.9 dB. This example proves the point stated earlier that using variable block sizes does help to make better use of smooth areas. When EC is larger than 13,000 bits, the Peng et al. PVO uses 2 × 2 blocks for embedding; the proposed method yields better performance in this case also. In addition, as mentioned before, changing the block size causes a significant change in PSNR, which generates an obvious inflection point when EC is 13,000 bits. In comparison, according to EC, the proposed method adaptively selects the block size for different image areas. When EC increases, only the smooth areas decrease their block size, and as a result, the proposed method can produce a smooth performance curve.

Table 1
Run time of the proposed method for each test image.

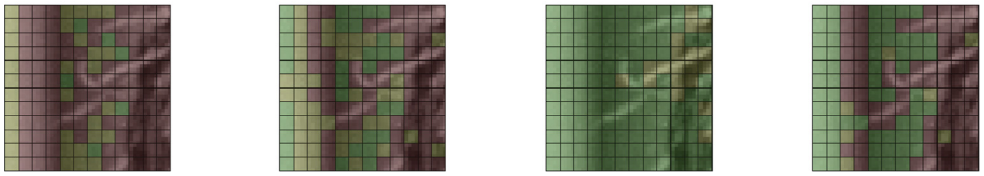
Image	Lena	Baboon	Barbara	Airplane	Peppers	Boat	Elaine	Lake	Average
Run time (s)	33.65	27.64	30.39	34.92	34.50	33.06	33.79	31.67	32.45

Table 2
Complexity thresholds T_1, T_2, and the size of the compressed location map l_{CLM} for EC of 10,000 and 20,000 bits.

Image	EC = 10,000 bits		EC = 20,000 bits		l_{CLM}
	T_1	T_2	T_1	T_2	
Lena	10	2	12	5	40
Baboon	135	24	–	–	184
Barbara	12	3	13	8	40
Airplane	3	1	6	2	40
Peppers	14	3	19	8	72
Boat	18	5	58	15	120
Elaine	19	5	29	15	64
Lake	14	7	28	14	40



(a) Selected area of Lena



(b) $(T_1, T_2) = (12, 5)$, and $EC = 20,000$

(c) $(T_1, T_2) = (25, 10)$, and $EC = 30,000$

(d) $(T_1, T_2) = (155, 49)$, and $EC = 38,000$

(e) $(T_1, T_2) = (23, 20)$

Fig. 10. Various division schemes under various thresholds around the selected area. Blocks in green denote that these blocks are divided into 2×2 blocks (flat blocks); blocks in yellow denote 4×4 blocks which are used for embedding (normal blocks), and blocks in red denote blocks that remain unchanged (rough blocks). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Fig. 12 presents a comparison between the Peng et al. scheme (using 2×2 and 4×4 blocks) and the proposed method on Lena, Baboon, Airplane, and Barbara. Compared with the use of 2×2 blocks, the proposed method achieves a substantial improvement in PSNR. For example, the increase is nearly 2.0 dB for embedding 11,000 bits of data in Barbara. When EC is low (e.g., less than 8000 bits on Lena), the performance of the proposed method is similar to that of the Peng et al. scheme with 4×4 blocks. This is because the proposed method uses a small T_2 for low EC, and as a result, most 4×4 blocks are not divided into sub-blocks. When increasing EC, the capacity-dependent block strategies [28,37] tend to use smaller blocks such as 3×3 or 2×2. Obviously, although using a smaller block size does help to improve EC, it significantly decreases PSNR. Fig. 12 indicates that the proposed dynamic block size strategy achieves superior performance to the Peng et al. scheme with 2×2 blocks. Note that the proposed method does not improve the maximum EC of the PVO algorithms. The maximum EC of the proposed method is almost the same as that of the Peng et al. method with 2×2 blocks. However, for relatively large EC, the proposed method produces a much higher PSNR, which is the most important advantage of this method.

The PEE scheme proposed by Sachnev et al. combines the virtues of sorting and PE. By using an embedding-position selection strategy, this method is superior to most contemporary RDH schemes. Fig. 13 shows that the proposed method performs better than this widely used algorithm in most cases, producing the greatest advantages on Barbara and Peppers. Specifically, as shown in Table 3, the PSNR achieved by the proposed method is 58.97, whereas the Sachnev et al. scheme achieves 55.56 when EC is 10,000 bits for Peppers, which is equivalent to an increase as large as 3.41 dB for the proposed method. The average increase over the Sachnev et al. method is 2.29 dB when EC is 10,000 bits and 1.75 dB when EC is 20,000 bits (Table 4). However, for some images, the proposed method performs more poorly than that of Sachnev et al. when the capacity reaches its maximum (e.g., for Baboon when EC is larger than 11,000 bits). The reason for this is the same as with the Peng et al. method: in such cases, because EC is high, the proposed method is required to use 2×2 blocks and cannot take much advantage of pixel correlation.

The scheme recently proposed by Li et al. [30] is based on two-dimensional difference histogram modification and difference-pair-mapping. In the Li et al. scheme, a pixel-pair-selection strategy is proposed to enhance RDH embedding performance. Fig. 13 shows that the embedding performance of the proposed method is significantly better than that of the Li et al. method for most test images. Specifically, for Lake and Peppers, the increase in PSNR is more than 1 dB at nearly all EC. The superiority of the proposed method is less obvious on smooth images (e.g., Airplane) because the Li et al. pixel-pair selection strategy can accurately locate the target pixels for embedding in smooth areas, and therefore their strategy works much better on smooth images than on other images. By comparison, the proposed method can also yield good performance on textured images by adaptively dividing the blocks. Tables 3 and 4 show that the performance of the proposed method improves by 0.87 dB on average for an EC of 10,000 bits and by 0.86 dB for an EC of 20,000 bits over the performance of the Li et al. scheme.

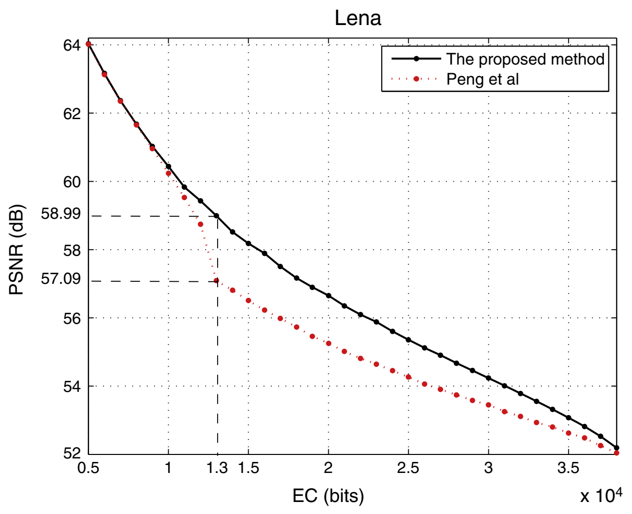


Fig. 11. Comparison between the Peng et al. scheme (with 2 × 2 and 4 × 4 blocks) and the proposed method, on Lena.

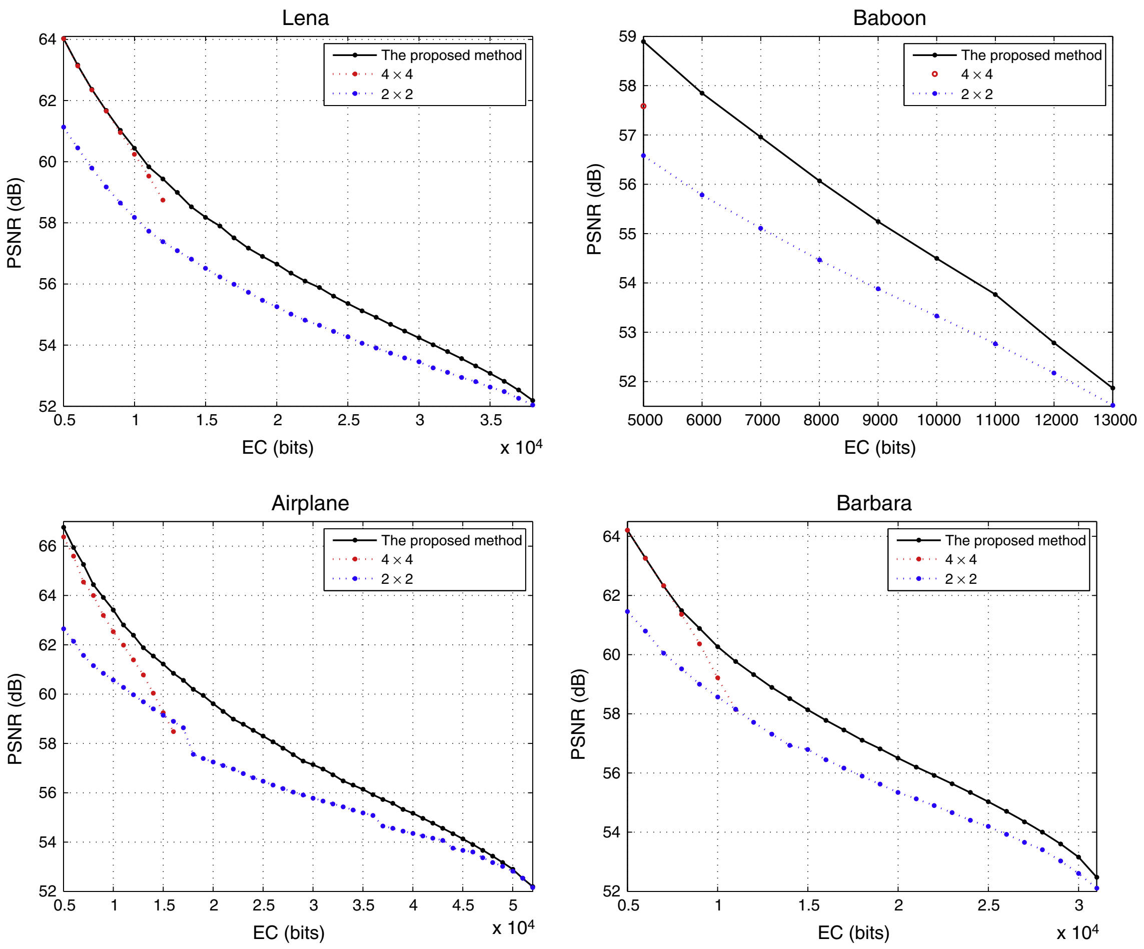


Fig. 12. Comparison between the Peng et al. scheme [37] (with 2 × 2 and 4 × 4 blocks) and the proposed method.

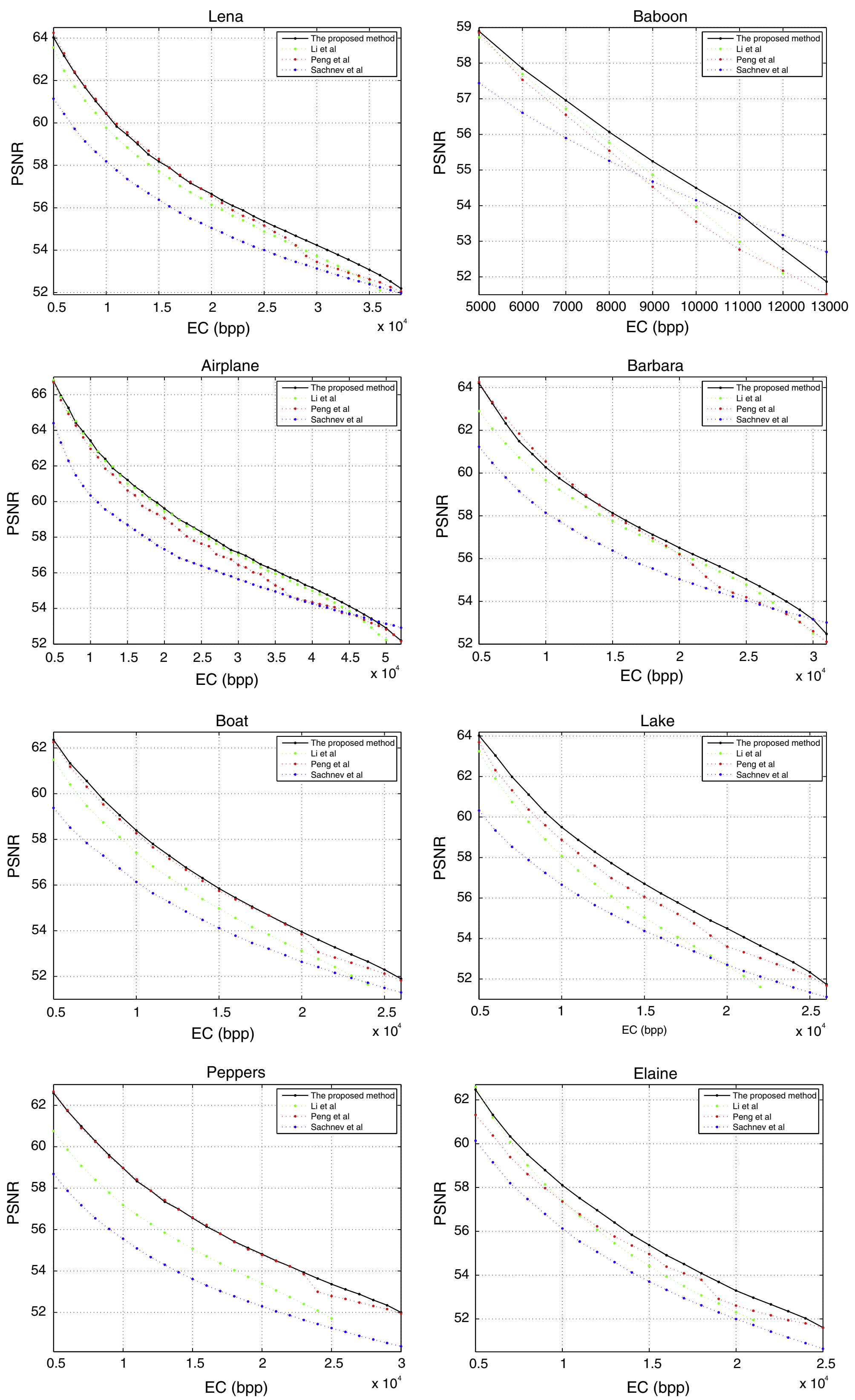


Fig. 13. Performance comparison between the Peng et al. [37], Sachnev et al. [39], and Li et al. [30] methods and the proposed approach.

Table 3
Comparison of PSNR (in dB) between the Peng et al. [37], Sachnev et al. [39], and Li et al. [30] methods and the proposed approach for an EC of 10,000 bits.

Image	Lena	Baboon	Barbara	Airplane	Peppers	Boat	Elaine	Lake	Average
Proposed method	60.44	54.50	60.27	63.41	58.97	58.39	58.10	59.50	59.20
Peng et al. [37]	60.47	53.55	60.55	62.96	58.98	58.27	57.36	58.87	58.87
Sachnev et al. [39]	59.19	54.15	58.14	60.34	55.56	56.14	56.13	56.65	56.91
Li et al. [30]	59.76	53.96	59.67	63.18	57.19	57.42	57.39	58.08	58.33

Table 4
Comparison of PSNR (in dB) between the Peng et al. [37], Sachnev et al. [39], and Li et al. [30] methods and the proposed approach for an EC of 20,000 bits.

Image	Lena	Baboon	Barbara	Airplane	Peppers	Boat	Elaine	Lake	Average
Proposed method	56.65	–	56.50	59.61	54.81	53.96	53.29	54.50	55.62
Peng et al. [37]	56.54	–	56.20	59.07	54.77	53.84	52.61	53.60	55.23
Sachnev et al. [39]	55.04	49.37	55.04	57.31	52.29	52.64	51.99	52.70	53.85
Li et al. [30]	56.14	–	56.24	59.45	53.39	53.12	52.31	52.66	54.76

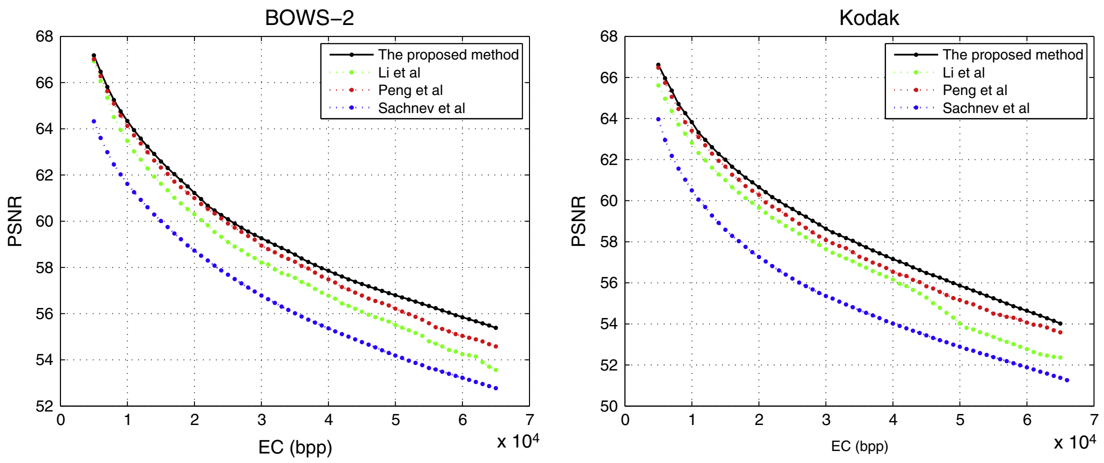


Fig. 14. Performance comparison on the BOWS-2 and Kodak image databases.

Table 5
Comparison of MSSIM between the Peng et al. [37], Sachnev et al. [39], and Li et al. [30] methods and the proposed approach, for an EC of 10,000 bits.

Image	Lena	Baboon	Barbara	Airplane	Peppers	Boat	Elaine	Lake	Average
Proposed method	0.9994	0.9991	0.9995	0.9996	0.9993	0.9993	0.9993	0.9994	0.9994
Peng et al. [37]	0.9992	0.9992	0.9993	0.9995	0.9991	0.9992	0.9992	0.9992	0.9992
Sachnev et al. [39]	0.9989	0.9988	0.9989	0.9992	0.9982	0.9986	0.9987	0.9985	0.9987
Li et al. [30]	0.9991	0.9989	0.9991	0.9995	0.9987	0.9989	0.9989	0.9988	0.9990

Table 6
Comparison of MSSIM between the Peng et al. [37], Sachnev et al. [39], and Li et al. [30] methods and the proposed approach, for an EC of 20,000 bits.

Image	Lena	Baboon	Barbara	Airplane	Peppers	Boat	Elaine	Lake	Average
Proposed method	0.9984	–	0.9986	0.9991	0.9979	0.9981	0.9978	0.9982	0.9983
Peng et al. [37]	0.9982	–	0.9984	0.9990	0.9977	0.9980	0.9978	0.9980	0.9982
Sachnev et al. [39]	0.9967	0.9978	0.9979	0.9985	0.9963	0.9972	0.9971	0.9969	0.9973
Li et al. [30]	0.9982	–	0.9983	0.9989	0.9972	0.9978	0.9973	0.9976	0.9979

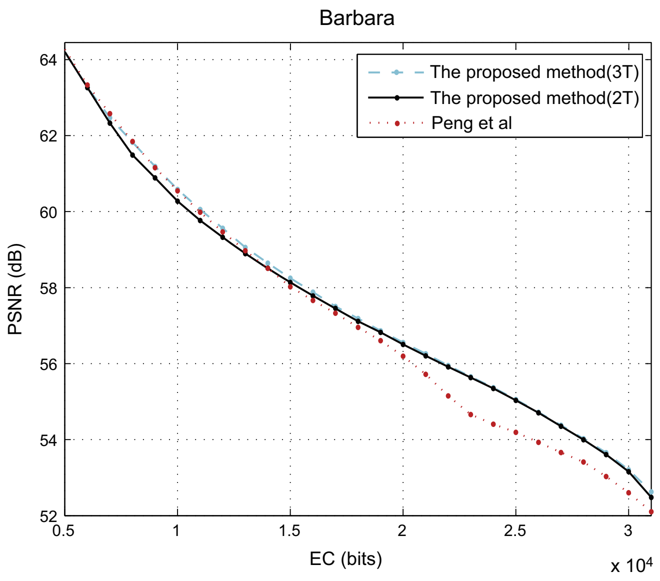


Fig. 15. Performance comparison between the proposed method with three and two thresholds and the original scheme for Barbara.

Referring to Fig. 13, the proposed method improves the performance of the Peng et al. scheme, especially for relatively high EC. For example, the improvement is 0.54 dB for embedding 20,000 bits of data and increases to nearly 1 dB when EC is larger than 30,000 bits. This phenomenon also appears for other images, especially Baboon and Barbara. The reason for this is that with high EC, the Peng et al. scheme generally uses 2×2 blocks to provide capacity. However, the proposed method preferentially divides flat blocks to improve capacity and keeps the normal 4×4 blocks to improve PSNR. When EC is low, the proposed method uses 4×4 blocks for both flat and normal blocks, whereas the Peng et al. method uses medium-sized blocks ($n_1, n_2 \in \{3, 4\}$) for embedding. Therefore, the performance gain is not as obvious as when EC is high, and the proposed method even performs a little worse than the Peng et al. scheme for Barbara when $EC \in [6000, 13000]$.

To present the advantages of the proposed method more clearly, a further comparison was performed on 100 images randomly selected from BOWS-2 data and 24 images from Kodak data. Fig. 14 presents the comparison results, with EC varying from 5000 to 65,000 bits (only a few images can provide an EC as large as 65,000 bits), and the PSNR is the average value for each EC. In Fig. 13, the Peng et al. method performs better than the proposed approach on certain specific images, but Fig. 14 clearly indicates that the proposed approach achieves a much better average performance on a large image database on all EC, and as EC increases, the superiority of the proposed method becomes more obvious. Specifically, when EC is larger than 50,000 bits, the increase in PSNR over the Peng et al. method is greater than 1 dB.

MSSIM [49] was also used as a criterion to evaluate the subjective quality of the proposed method. Tables 5 and 6 present the results for ECs of 10,000 and 20,000 bits. Although all four methods yielded very high MSSIM results (greater than 0.99), it is clear that the proposed method still produces the highest MSSIM performance. Note that the Peng et al. method produces an MSSIM similar to the proposed method at low EC, but the advantage of the proposed method becomes clear as EC is increased.

Finally, note that performance can be further improved by introducing 2×4 blocks into embedding by using another threshold T_3 ($T_2 \leqslant T_3 \leqslant T_1$). T_3 controls whether a 4×4 block is divided into two 2×4 blocks. Recalling that when only two thresholds, T_2 and T_1, are used, the 4×4 normal blocks ($T_2 < NL \leqslant T_1$) are directly used for embedding. Here, the threshold T_3 is defined to classify normal blocks into two groups: (1) normal blocks with $T_2 < NL \leqslant T_3$ are divided into 2×4 blocks to embed four bits of data; and (2) the rest of the normal blocks with $T_3 < NL \leqslant T_1$ remain as 4×4 blocks for embedding. This strategy is an extension of the dynamic blocking strategy, which effectively improves the poor performance of the proposed method when the given EC is in a middle range (neither too big nor too small). However, the weakness of this extension is the additional calculation complexity. The performance of the extension is shown in Fig. 15, where the proposed method (3T) means using three thresholds for embedding and the proposed method (2T) means using two thresholds. It is obvious that the PSNR increases when one more choice is offered, that of dividing a 4×4 block into two 2×4 blocks under $EC \in [6000, 13000]$, and the average PSNR increase is 0.22 dB.

7. Conclusions

This paper has proposed a dynamic blocking algorithm which is an extension of the pixel-value ordering strategy. Instead of using a fixed block size, the proposed method can adaptively select the block size from $4 \times 4, 2 \times 2$, and even 2×4 to embed data. This strategy reduces the time required to calculate PVO thresholds and improves embedding efficiency. In addition, by using multiple block sizes, the proposed method makes better use of the neighbor correlations inherent in the cover image, thus producing superior performance. Finally, the proposed method is compared with the Peng et al. scheme and many other state-of-the-art methods.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant Nos. 61202391, U1401251 and 61373170), Program for New Century Excellent Talents in University (Grant No. NCET-11-0691) and the Natural Science Basic Research Plan in Shaanxi Province, China (Grant No. 2014JQ8308).

References

[1] A.M. Alattar, Reversible watermark using the difference expansion of a generalized integer transform, IEEE Trans. Image Process. 13 (2004) 1147–1156.
[2] M. Arsalan, S.A. Malik, A. Khan, Intelligent reversible watermarking in integer wavelet domain for medical images, J. Syst. Softw. 85 (2012) 883–894.
[3] R. Caldelli, F. Filippini, R. Becarelli, Reversible watermarking techniques: an overview and a classification, EURASIP J. Inform. Security 2010 (2010).
[4] M.U. Celik, G. Sharma, A.M. Tekalp, Lossless watermarking for image authentication: a new framework and an implementation, IEEE Trans. Image Process. 15 (2006) 1042–1049.
[5] M.U. Celik, G. Sharma, A.M. Tekalp, E. Saber, Lossless generalized-LSB data embedding, IEEE Trans. Image Process. 14 (2005) 253–266.
[6] C.C. Chang, C.C. Lin, Y.H. Chen, Reversible data-embedding scheme using differences between original and predicted pixel values, IET Inform. Security 2 (2008) 35–46.
[7] C.C. Chang, C.C. Lin, C.S. Tseng, W.L. Tai, Reversible hiding in DCT-based compressed images, Inform. Sci. 177 (2007) 2768–2786.
[8] C.C. Chang, T.S. Nguyen, C.C. Lin, A reversible compression code hiding using SOC and SMVQ indices, Inform. Sci. 300 (2015) 85–99.
[9] M. Chen, Z. Chen, X. Zeng, Z. Xiong, Reversible data hiding using additive prediction-error expansion, in: Proceeding of ACM Workshop on Multimedia and Security, 2009, pp. 19–24.
[10] G. Coatrieux, W. Pan, N. Cuppens-Boulahia, F. Cuppens, C. Roux, Reversible watermarking based on invariant image classification and dynamic histogram shifting, IEEE Trans. Inform. Forensics Security 8 (2013) 111–120.
[11] D. Coltuc, Improved embedding for prediction-based reversible watermarking, IEEE Trans. Inform. Forensics Security 6 (2011) 873–882.
[12] I.C. Dragoi, D. Coltuc, I. Caciula, Gradient based prediction for reversible watermarking by difference expansion, in: Proc. IH&MMSec, 2014.
[13] M. Fallahpour, Reversible image data hiding based on gradient adjusted prediction, IEICE Electron. Express 5 (2008) 870–876.
[14] J. Fridrich, M. Goljan, R. Du, Invertible authentication, in: Security and Watermarking of Multimedia Contents III, 2001, pp. 197–208.
[15] J. Fridrich, M. Goljan, R. Du, Lossless data embedding – new paradigm in digital watermarking, EURASIP J. Appl. Signal Process. 2002 (2002) 185–196.
[16] X. Gao, L. An, Y. Yuan, D. Tao, X. Li, Lossless data embedding using generalized statistical quantity histogram, IEEE Trans. Circ. Syst. Video Technol. 21 (2011) 1061–1070.
[17] W. Hong, An efficient prediction-and-shifting embedding technique for high quality reversible data hiding, EURASIP J. Adv. Signal Process. 2010 (2010).
[18] W. Hong, Adaptive reversible data hiding method based on error energy control and histogram shifting, Opt. Commun. 285 (2012) 101–108.
[19] W. Hong, T.S. Chen, C.W. Shiu, Reversible data hiding for high quality images using modification of prediction errors, J. Syst. Softw. 82 (2009) 1833–1842.
[20] Y. Hu, H.K. Lee, J. Li, DE-based reversible data hiding with improved overflow location map, IEEE Trans. Circ. Syst. Video Technol. 19 (2009) 250–260.
[21] J. Hwang, J. Kim, J. Choi, A reversible watermarking based on histogram shifting, in: Proceeding of International Workshop on Digital-forensics and Watermarking (IWDW), 2006, pp. 348–361.
[22] S.W. Jung, Adaptive post-filtering of JPEG compressed images considering compressed domain lossless data hiding, Inform. Sci. 281 (2014) 355–364. Multimedia Modeling.
[23] L. Kamstra, H.J.A.M. Heijmans, Reversible data embedding into images using wavelet techniques and sorting, IEEE Trans. Image Process. 14 (2005) 2082–2090.
[24] A. Khan, A. Siddiqa, S. Munib, S.A. Malik, A recent survey of reversible watermarking techniques, Inform. Sci. 279 (2014) 251–272.
[25] J.D. Lee, Y.H. Chiou, J.M. Guo, Lossless data hiding for VQ indices based on neighboring correlation, Inform. Sci. 221 (2013) 419–438.
[26] S. Lee, C.D. Yoo, T. Kalker, Reversible image watermarking based on integer-to-integer wavelet transform, IEEE Trans. Inform. Forensics Security 2 (2007) 321–330.
[27] H. Leung, L. Cheng, F. Liu, Q. Fu, Adaptive reversible data hiding based on block median preservation and modification of prediction errors, J. Syst. Softw. 86 (2013) 2204–2219.
[28] X. Li, J. Li, B. Li, B. Yang, High-fidelity reversible data hiding scheme based on pixel-value-ordering and prediction-error expansion, Signal Process. 93 (2013) 198–205.
[29] X. Li, B. Yang, T. Zeng, Efficient reversible watermarking based on adaptive prediction-error expansion and pixel selection, IEEE Trans. Image Process. 20 (2011) 3524–3533.
[30] X. Li, W. Zhang, X. Gui, B. Yang, A novel reversible data hiding scheme based on two-dimensional difference-histogram modification, IEEE Trans. Inform. Forensics Security 8 (2013) 1091–1100.
[31] X. Li, W. Zhang, B. Ou, B. Yang, A brief review on reversible data hiding: current techniques and future prospects, in: Proc. ChinaSIP, 2014.
[32] C.C. Lin, X.L. Liu, S.M. Yuan, Reversible data hiding for vq-compressed images based on search-order coding and state-codebook mapping, Inform. Sci. 293 (2015) 314–326.
[33] L. Luo, Z. Chen, M. Chen, X. Zeng, Z. Xiong, Reversible image watermarking using interpolation technique, IEEE Trans. Inform. Forensics Security 5 (2010) 187–193.
[34] Z. Ni, Y.Q. Shi, N. Ansari, W. Su, Reversible data hiding, IEEE Trans. Circ. Syst. Video Technol. 16 (2006) 354–362.
[35] B. Ou, X. Li, Y. Zhao, R. Ni, Reversible data hiding based on PDE predictor, J. Syst. Softw. 86 (2013) 2700–2709.
[36] Q. Pei, X. Wang, Y. Li, H. Li, Adaptive reversible watermarking with improved embedding capacity, J. Syst. Softw. 86 (2013) 2841–2848.
[37] F. Peng, X. Li, B. Yang, Improved pvo-based reversible data hiding, Digital Signal Process. 25 (2014) 255–265.
[38] C. Qin, C.C. Chang, Y.H. Huang, L.T. Liao, An inpainting-assisted reversible steganographic scheme using a histogram shifting mechanism, IEEE Trans. Circ. Syst. Video Technol. 23 (2013) 1109–1118.
[39] V. Sachnev, H.J. Kim, J. Nam, S. Suresh, Y.Q. Shi, Reversible watermarking algorithm using sorting and prediction, IEEE Trans. Circ. Syst. Video Technol. 19 (2009) 989–999.
[40] Y.O. Shi, Reversible data hiding, in: Proceeding of International Workshop on Digital-forensics and Watermarking (IWDW), 2004, pp. 1–12.
[41] Y.O. Shi, Z. Ni, D. Zou, C. Liang, G. Xuan, Lossless data hiding: fundamentals, algorithms and applications, in: Proceeding of IEEE International Symposium on Circuits and Systems (ISCAS), 2004, pp. 33–36.
[42] D.M. Thodi, J.J. Rodriguez, Expansion embedding techniques for reversible watermarking, IEEE Trans. Image Process. 16 (2007) 721–730.
[43] J. Tian, Reversible data embedding using a difference expansion, IEEE Trans. Circ. Syst. Video Technol. 13 (2003) 890–896.
[44] P. Tsai, Y.C. Hu, H.L. Yeh, Reversible image hiding scheme using predictive coding and histogram shifting, Signal Process. 89 (2009) 1129–1143.

[45] H.W. Tseng, C.P. Hsieh, Prediction-based reversible data hiding, Inform. Sci. 179 (2009) 2460–2469.
[46] T.Y. Tu, C.H. Wang, Reversible data hiding with high payload based on referred frequency for VQ compressed codes index, Signal Process. 108 (2015) 278–287.
[47] K. Wang, Z.M. Lu, Y.J. Hu, A high capacity lossless data hiding scheme for JPEG images, J. Syst. Softw. 86 (2013) 1965–1975.
[48] X. Wang, X. Li, B. Yang, Z. Guo, Efficient generalized integer transform for reversible watermarking, IEEE Signal Process. Lett. 17 (2010) 567–570.
[49] Z. Wang, E. Simoncelli, A. Bovik, Multiscale structural similarity for image quality assessment, in: Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers, vol. 2, 2004, pp. 1398–1402.
[50] M.J. Weinberger, G. Seroussi, G. Sapiro, The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS, IEEE Trans. Image Process. 9 (2000) 1309–1324.
[51] X. Wu, N. Memon, Context-based, adaptive, lossless image coding, IEEE Trans. Commun. 45 (1997) 437–444.