International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 22, No. 6 (2008) 1215–1240

World Scientific
www.worldscientific.com

HIDING IMAGES USING MODIFIED SEARCH-ORDER CODING AND MODULUS FUNCTION

YU-JIE CHANG*,‡, RAN-ZAN WANG† and JA-CHEN LIN*

*Department of Computer Science
National Chiao Tung University, Hsinchu, 300, Taiwan, R.O.C.

†Department of Computer Science and Engineering
Yuan Ze University, Chung-Li, 320, Taiwan, R.O.C.
‡yjchang@cis.nctu.edu.tw

This paper proposes a method for hiding an important image in a cover image whose size is limited. In this method, in order to save space, a modified search-order coding (MSOC) technique first transforms the important image, then, a randomization procedure permutes the transformed image to further increase the security. Finally, a modulus function embeds the permuted code in a cover image; notably, in the modules function, the modulus base used for a pixel is determined according to the variance of its neighboring pixels. Experimental results show that the images are of high quality. Comparisons with reported methods are provided.

Keywords: Data hiding; modulus function; search-order coding; steganography.

1. Introduction

With the rapid development of the Internet and WWW technologies, digital media, such as text, image, audio and video are transmitted via a network. Since Internet is public, the transmitted material is exposed. As a result, how to protect important or private message during a transmission becomes an interesting and important research issue. One of the possible solutions for protecting important data against illegal access is using encryption techniques.[4] An encryption scheme encodes an important image in such a way that it is nearly impossible for any person who does not possess the secret key to decode the cipher-image. The cipher-image might appear noisy, and so assure the content of the important image is not readable before decoding. However, looking noisy might thus attract the attention of attackers during the transmission activity; and hence increases the risk of facing attacks. Steganography is an art of concealed communication which aims to send the important image silently under the cover of some carrier signals,[10] and thus compensates the aforementioned drawback of encryption technique. Unlike encryption techniques which make the content of the important image

unreadable, steganography tries to cheat the hackers by concealing the existence of the content. (After hiding the important image in a cover image, they become a so-called stego-image, which still looks like the cover image, although the important image can be extracted from the stego-image.) Usually, if encryption is combined with a steganography scheme, the protection becomes more powerful.

In the literature, many image steganography methods have been proposed. A very simple approach is the least-significant-bits (LSB) substitution scheme.[1,14] This kind of method replaces directly certain bits of the cover image's pixels with the pixels' bits of the important image; and a random permutation process to the important data is often conducted to enhance the security level of the method.[11] In 2001, Wang *et al.*[14] used a genetic algorithm to approximate a theoretically-optimal solution of the simple LSB substitution method; however, due to the nature of genetic algorithms, the computation time is quite huge. Chan and Cheng[1] used another pixel adjustment process to enhance the quality of the stego-image obtained by the simple LSB substitution method, and its extra computational cost is relatively small compared with Ref. 14. Chang *et al.*[2] also used another algorithm to search for optimal substitution for the simple LSB substitution method. Although their PSNR performance is similar to Wang *et al.*'s method[14] (to hide a 256×256 important image Jet, the PSNR of the 512×512 Lena stego-images in Refs. 14 and 2 are, respectively, 44.172 and 44.169 dB); their embedding takes only about 1/7 of the time needed by Wang *et al.*'s method.[14]

To explore more steganography techniques, other studies based on vector quantization (VQ) have also been introduced.[3,6] In Ref. 3, Chung *et al.* proposed to hide an image using singular value decomposition (SVD) and VQ techniques. By using partial SVD procedure on both the important image and cover image, they hid some quantized singular values of a diagonal matrix corresponding to the important image in the less significant SVD components of the cover image. They can hide an important image whose size is as large as that of the cover image, but the degradation of the extracted important image is not very small.

In Ref. 6, before the embedding process, Hu and Lin first encoded the important image according to a given codebook. Then, the obtained indices and related parameters are encrypted by a cipher scheme such as the Data Encryption Standard (DES).[4] Finally, the encrypted data is embedded in the cover image using the LSB substitution. The hiding capacity of their scheme can be large, and the scheme can even embed multiple important images by using a codebook of smaller size and increasing the number of LSBs used in each pixel of the cover image. However, the quality of the extracted important images usually degraded. To reveal the important image without any loss, Wu and Tsai[15] utilized the difference among two adjacent pixels in the cover image to hide the important data. However, the hiding capacities of their so-called PVD steganographic scheme depend on the nature of the cover image (smooth or noisy), and are usually smaller comparing with the aforementioned VQ-based method. In order to improve the hiding capacity, Wu *et al.*[16] utilized the simple LSB substitution method in smooth area of the cover

image, and still used the PVD method[15] in edge area. Compared with the PVD method,[15] this mixed version[16] obtains not only larger hiding capacity, but also better stego-image quality.

In this paper, a steganographic method based on the modified search-order coding (MSOC) and a variance-based modulus function is proposed. The MSOC scheme utilizes the feature of high correlation among adjacency pixels (i.e. many neighboring pixels are with similar gray-values) to encode the important image. An adjustable threshold T is used in the MSOC; and this T directly controls the quality of the extracted image. Finally, to embed the MSOC output code in a cover image, at each pixel of the cover image, its {Northwest, North, West} three-neighbor variance is evaluated to estimate the hiding capacity of the pixel, and the MSOC output code is embedded in the cover image using two sets of modulus function. Notably, in order to have the ability of extracting the important image in the future, the evaluation of the aforementioned variance uses the stego-pixel-values (rather than the original cover-pixel-values) of the three neighbors. This is because the {Northwest, North, West} three-neighboring-pixels are already modified to hide data in them, and we do not keep the cover image after embedding; therefore, in the future decoding-phase, the evaluation of the variance can only use stego-image neighbors, not the original cover-image neighbors.

The rest of the paper is organized as follows. Section 2 takes a brief review of the SOC and the modulus embedding function. The details of the proposed method are described in Sec. 3. Experimental results are shown in Sec. 4. The discussions are in Sec. 5, and the summary is in Sec. 6.

2. Related Works

In this section, the related works are briefly reviewed to provide some necessary background knowledge. To begin with, the search-order coding (SOC)[5] is briefly reviewed in Sec. 2.1. Then, the embedding methods using modulus function[12,13] are briefly discussed in Sec. 2.2.

2.1. *Review of the search-order coding (SOC) tool*

Vector quantization (VQ) is a simple technique to compress images. According to a given codebook, an index file is generated as the compression result for each given image. To reduce the size of the index file further, Hsieh and Tsai[5] proposed the use of the search-order coding (SOC). The SOC algorithm in Ref. 5 encodes traditional VQ indices with fewer bits by utilizing the fact that there exists high correlation among adjacent indices, i.e. there exist many blocks whose VQ indices also appear in their neighborhood blocks. Recall that in traditional VQ, each block of the given image is represented by an index. So, the whole image is represented by an index file in which the number of indices (counting repetition) equals the number of blocks. The SOC algorithm encodes the index file of an image in an index by index manner (or equivalently, block by block). All of the indices that appear in a predefined search

path, which just cover a small area of the neighborhood blocks, are called search points (SP), and the nonsearch points are just those indices whose block location are beyond the range covered by the predefined search path. To encode the current not-yet-processed index, people begin a search along a predefined path. The search is in order to find whether a nearby block also has the same VQ index used by the current block. If a match is found in nearby area, then the original index value (OIV) of the current block is replaced by a search-order code (SOC) that indicates the position of the matched block in the search path; the SOC uses fewer bits than the OIV. On the other hand, if no match can be found in nearby area, then Ref. 5 still uses the OIV of the current block. In general, a SOC is defined as an order in which the indices of the already-processed neighborhood blocks are compared with the index of the current block. Of course, to let the decoder distinguish between the SOC and OIV, an extra indicator bit (the flag) is added in front of the resulting compression code for each index.

An example of the SOC is shown in Fig. 1. The current block is at location (3, 3), and the index value for the current block is 76. A predefined search path is shown with arrows. In this example, assume the starting search point is the neighboring block (3, 2), and a search-order code of $N = 2$ bits is used, where N is the number of bits used to record the order of the matched position in the SOC searching. Since $2^N = 2^2 = 4$, there are at most four SPs used for comparison excluding the repetition points. The path 75-75-74-74 is the level 1 search. Since there is no value that matches 76, we begin the level 2 search 75-76-... in the outer loop; and stop at the first matched value 76 located at (2, 1). Since the position of block (2, 1) is "10" (the third kind of value) in the search code, the block (3, 3) is encoded as the search-order code "10". Of course, a flag bit is also needed. Therefore, block (3, 3) is coded using $1 + n = 1 + 2 = 3$ bits, which is more economic than, for example, the 8-bits index needed in traditional VQ if there are $256 = 2^8$ distinct code-blocks in the codebook. (In traditional VQ, each block of the given image is represented by a code-block found in this VQ codebook, and only the code-block's

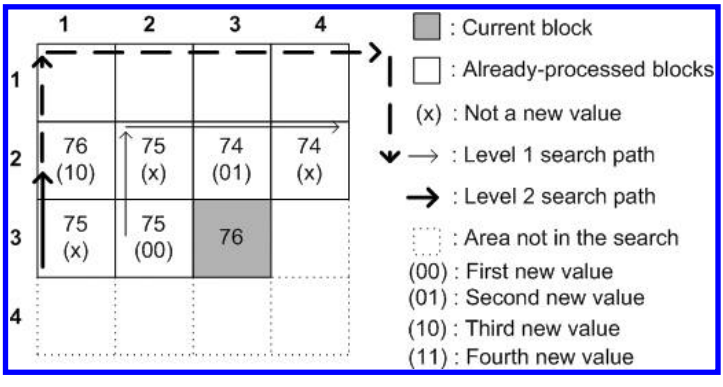


Fig. 1. An example of the SOC algorithm.

index (rather than the code-block itself) is transmitted. So, 8-bits for an index if there are $256 = 2^8$ distinct code-blocks.)

2.2. *Review of the modulus embedding function*

In 2003, Thien and Lin[12] applied a modulus function to embed data in still images; their scheme can hide data efficiently and the base value is not necessarily in $\{2, 4, 8, 16, \dots\}$. Their scheme is simple and fast, and their experimental results show that the qualities of the obtained stego-images are much better than that of the simple LSB substitution method. In 2005, Wang[13] observed that the quality of the stego-image in Ref. 12 may degrade too much if a large number of bits are embedded in a pixel of the cover image whose pixel value is small. Therefore, Wang extended further the modulus embedding method to a more flexible extent. In his method, the pixels of the cover image are classified into two groups: one group is G_U which contains the pixels whose values are greater than a predetermined V, and the other group G_L which contains the pixels whose value are at most V. Then, a modulus function with a large modulus base M_U is applied to embed data in G_U, and another modulus function with a small modulus base M_L is preformed to embed data in G_L. This indicates that more data bits are embedded in the pixels of G_U, and fewer data bits are embedded in the pixels of G_L.

The modulus embedding procedure[13] are reviewed below. We first start with a system using a single modulus base. Let the integer parameter M_O denote a modulus base. To embed a numerical data x $(0 \le x < M_O)$ in a pixel value y_{ij} $(0 \le y_{ij} \le 255)$ at position (i, j) of the cover image, we show below how to construct the new value $\hat{y}_{ij}$ which can be utilized to extract x. First, calculate the plain difference value

$$d_{ij} = x - (y_{ij} \bmod M_O). \tag{1}$$

Then, evaluate the modified difference value d'_{ij} by the rule

$$d'_{ij} = \begin{cases} d_{ij} & \text{if } \left(-\left\lfloor \dfrac{M_O - 1}{2} \right\rfloor\right) \le d_{ij} \le \left\lceil \dfrac{M_O - 1}{2} \right\rceil; \\ d_{ij} + M_O & \text{if } (-M_O + 1) \le d_{ij} < \left(-\left\lfloor \dfrac{M_O - 1}{2} \right\rfloor\right); \\ d_{ij} - M_O & \text{if } \left\lceil \dfrac{M_O - 1}{2} \right\rceil < d_{ij} < M_O. \end{cases} \tag{2}$$

Then, replace the old pixel value y_{ij} by the new value

$$\hat{y}_{ij} = d'_{ij} + y_{ij}. \tag{3}$$

Of course, a boundary checking procedure is needed to ensure that the gray value $\hat{y}_{ij}$ falls in a valid range between V_b and V_t. Do the following adjustment if necessary:

$$\hat{y}_{ij} = \begin{cases} \hat{y}_{ij} + M_O & \text{if } \hat{y}_{ij} < V_b; \\ \hat{y}_{ij} - M_O & \text{if } \hat{y}_{ij} > V_t. \end{cases} \tag{4}$$

Note that in this method in Ref. 13, the pixel values y_{ij} of the cover image are classified into two groups G_U and G_L, and each group has its own modulus base values: M_U for G_U, and M_L for G_L. If a pixel value y_{ij} of the cover image belongs to G_L, then the value of V_b is set to 0 and the value of V_t is set to V. On the other hand, if y_{ij} belongs to G_U, then the value of V_b is V and the value of V_t is 255.

3. The Proposed Method

In this section, we introduce our image hiding method. The flowchart of the proposed method is depicted in Fig. 2. First, the MSOC is used to encode the important image, and the output code is then permuted using a pseudo-random method. Finally, the randomized code is embedded in the cover image by using the modulus embedding function. Therefore, there are three major parts in our scheme: (a) the MSOC scheme, (b) the pseudo-random permutation process, and (c) the modulus embedding process. The details of these three parts are described in Secs. 3.1–3.3, respectively. The extraction procedure to unveil the embedded important image from stego-image is presented in Sec. 3.4.

3.1. *The modified search-order coding (MSOC)*

Instead of processing the vector-quantization (VQ) indices of an image as the traditional SOC scheme[5] did, our MSOC scheme processes the pixel data directly.

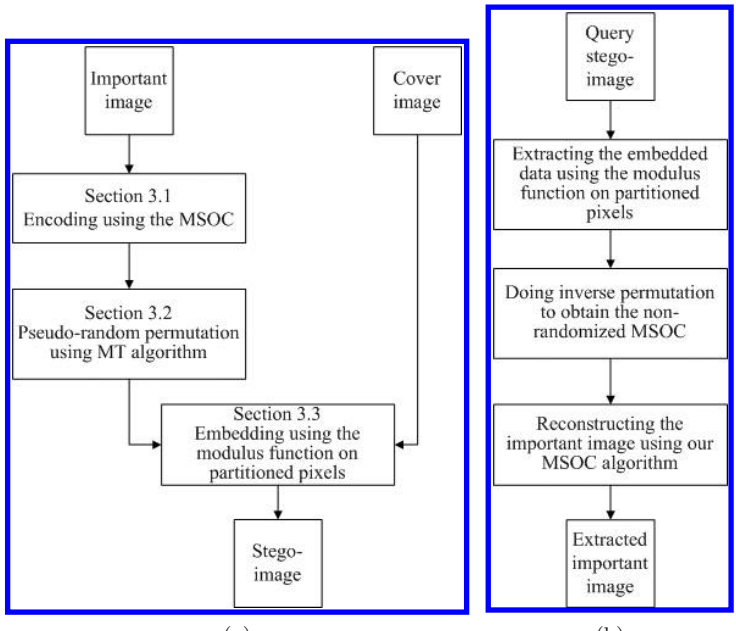


Fig. 2. Two flowcharts showing our proposed method. (a) The encoding procedure, and (b) the decoding procedure.

Also, due to the fact that adjacent pixels of an image often have similar gray values, we modify the match condition used in traditional SOC method,[5] and use a predetermined threshold T to encode the important image. The purpose of these modifications is that, when a set of numbers is compressed, the MSOC can yield more compact codes than SOC does, and hence produce a better-quality stego-image in the embedding stage later. The notation and details of MSOC are stated below:

Notation

- N: The length of the code for MSOC position, i.e. the number of bits used to record a matched position in the searching path.
- T: A threshold indicating the tolerance level in the MSOC matching equations (5) and (6) (Set $T = 1$ will make the extracted important image free of error.)
- SP: The pixels that appear in the predefined search path.
- P_c: The currently-processed pixel (of the important image).
- S_c: The set of candidate SPs for P_c [see Eq. (5)], in which the gray values of all candidates must not differ too much from the gray value of P_c.
- V_{cut}: The threshold to decide which modulus base is to be used in the embedding phase (Sec. 3.3).

The MSOC Encoding Algorithm

Input: The important image, and two positive integers N and T. (Set T to 1 if the extracted important image is required to be error-free.)

Output: The MSOC code of the important image.

Step 1: According to raster scan order, take the next not-yet-processed "pixel" P_c from the important image. Note that P_c is a pixel in MSOC algorithm, rather than a block.

Step 2: Without the loss of generality, assume that the left adjacent pixel of P_c is the starting search point. Generate 2^N SPs from a predefined search path. Note that a pixel whose value equals to a value that has already appeared in some previous SPs (of P_c) of the current search path shall be skipped, and no SP value will be assigned to a skipped pixel. From the set of SPs, we will be only interested in the SPs whose values are similar to the value of P_c (up to a threshold T), i.e.

$$S_c = \{SP_j : |\text{value}(SP_j) - \text{value}(P_c)| < T,\ 0 \le j \le 2^N - 1\}. \tag{5}$$

Step 3: If S_c is not empty, then the original pixel value of P_c is replaced by $\text{MSOC}(P_c)$, which is defined as the element in S_c whose value is most similar to the value of P_c, i.e.

$$\text{MSOC}(P_c) = \arg\min_{j\ SP_j \in S_c} \{|\text{value}(SP_j) - \text{value}(P_c)|\}, \tag{6}$$

where the arg(·) operator returns the position-code (a N-bits binary string since the predefined search path in Step 2 has 2^N SPs) of the point SP_j^* in S_c satisfying $|\text{value}(SP_j^*) - \text{value}(P_c)| = \min_{SP_j \in S_c}\{|\text{value}(SP_j) - \text{value}(P_c)|\}$. However, if S_c is empty, then the original pixel value (OPV) of P_c is used as the output. As in traditional SOC scheme, we also add an extra indication bit (the flag) in order to distinguish between an MSOC and an OPV.

Step 4: Repeat Steps 1 to 3 until all pixels of the important image is processed.

Below we use Fig. 1 again to explain MSOC. Note that every cell in Fig. 1 should now be explained as a pixel, rather than an image block or a VQ-index. Assume the threshold T is set to 2, and the value of the pixel (1, 1) is 77. Let $P_c = (3, 3)$ be the current pixel. By Step 2 of the MSOC algorithm mentioned above, the gray values in the candidate set S_c of pixel (3, 3) is {75 (00), 76 (10)}. Since the value 76 is the closest (in fact, identical) to the value of the current pixel (3, 3), the value of (3, 3) will be encoded using the code "$(\underline{\mathbf{0}}10)_2$" by Step 3 above. The underlined bit $\underline{\mathbf{0}}$ is the indicator bit, and the following two bits "10" indicate the pixel value can be found using a previous pixel at position $(10)_2$ in the search-path.

In the other example, assume that the values of the pixels (2, 2), (3, 1), and (3, 2) are all 78 rather than 75, and the value of the pixel (2, 1) is 79 rather than 76. Then, since none of the values {78, 79, 74} is so close to value$(P_c) = 76$ that the difference is less than the threshold $T = 2$; Step 2 of the MSOC algorithm implies that the candidate set S_c of the pixel (3, 3) is an empty set. In other words, no tolerable match of the pixel (3, 3) can be found. Therefore, the pixel (3, 3) is encoded as "$(\underline{\mathbf{1}}01001100)_2$" where $\underline{\mathbf{1}}$ is the flag bit, and $(01001100)_2 = (76)_{10}$ is the original pixel value (OPV) of the pixel (3, 3).

3.2. *The pseudo-random permutation of location*

Before embedding the produced MSOC code in cover image, to increase the level of security, we will permute the MSOC code by using a pseudo-random process. Reference 4 had adopted a mono-alphabetic substitution cipher algorithm[11] to randomize their important image before doing their LSB-related embedding process. However, we do not intend to use the mono-alphabetic substitution cipher algorithm in the current approach, for the reason stated below. Figure 3(b) shows the result of applying the mono-alphabetic substitution cipher algorithm to the image shown in Fig. 3(a), and it is obvious that there are still some regular patterns in Fig. 3(b). In other words, the result is not quite random. Therefore, we design here another pseudo-random permutation method based on the Mersenne Twister (MT) pseudo-random number generator.[9]

Assume that the output code of the MSOC is treated as a binary string, and the bit locations in the binary string are numbered sequentially from 0 to S–1, where S is the size of the binary string. Our pseudo-random permutation is to transform each bit location i of the binary string to a new location $f(i)$.

The pseudo-random permutation algorithm

Input: A binary stream which is the MSOC output code of an important image.

Parameter settings: Randomly choose a 32-bit integer greater than 0. This is the so-called "seed" for the MT pseudo-random number generator. With this seed, a series of pseudo-random real numbers R_j $(j = 0, 1, 2, K, 2^{19937} - 1)$, which are uniformly distributed on [0, 1]-interval, can be generated by the Mersenne Twister (MT) pseudo-random number generator.[9]

Output: The randomized form of the MSOC output code.

Step 0: Initially, set $i = -1$.

Step 1: Increase i by 1. The ith bit of the MSOC binary string will be permuted to a new location $f(i)$, as computed by Steps 2–4 below.

Step 2: Get next not-yet-taken fraction-number R_j from the MT series.

Step 3: The new location $f(i)$ is computed by

$$f(i) = \text{int}(R_j \times (S - 1)) \tag{7}$$

where the int(·) operator means getting the integer part of a real number.

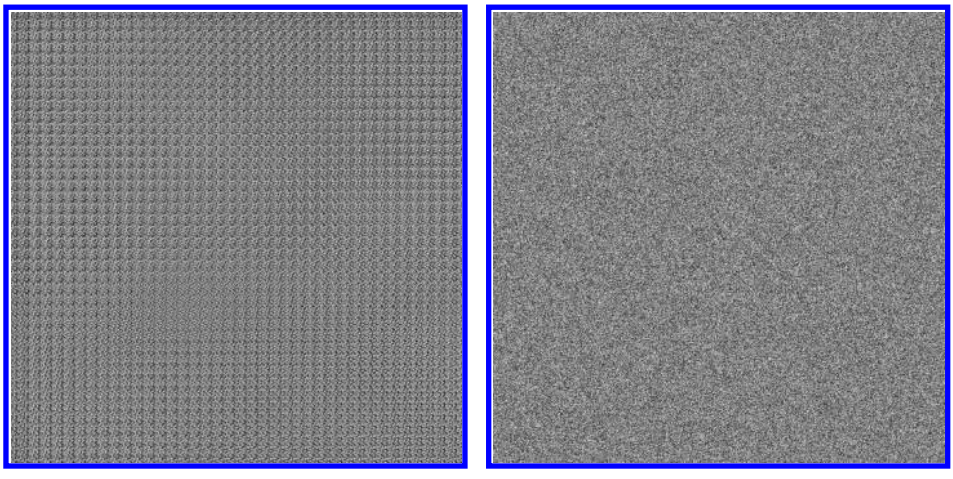
Step 4: If the new location $f(i)$ is a repetition, return to Step 2 to get the next R_j.

Step 5: Repeat Steps 1 to 4 until the entire binary strings are processed $(i = S-1)$.

By the above pseudo-random permutation algorithm, the output code of the MSOC procedure mentioned in Sec. 3.1 is permuted into a pseudo-random code to enhance the security property of our scheme. The seed used in the MT algorithm can be regarded as a secret key; and only the authorized extractor who owns the same secret key can obtain the same sequence of pseudo-random real numbers to recovery the MSOC code in the decoding phase.

Of course, the above pseudo-random permutation algorithm can also be used to permute pixels of any image. Just replace the term "bit location" by "pixel location", and treat the whole input image as a sequence of S pixels. When we use this "pixel" version to randomize the Pepper image in Fig. 3(a), the result is in Fig. 3(c). Compare Figs. 3(b) and 3(c), it is quite clear that our pseudo-random permutation algorithm has better randomness. In this example, the seed of the MT algorithm is set to 4357, and $S = 512 \times 512 = 262{,}144$, because Fig. 3(a) has 512×512 pixels. When the seed is 4357, the MT generator creates a series of fraction numbers whose first five numbers R_0–R_4 are {0.817330, 0.999061, 0.510354, 0.131533, 0.035416}. Hence, by Step 3 of the above pseudo-random permutation algorithm, the new locations $f(0) - f(4)$ are {214,257, 261,896, 133,785, 34,480, 9284}. It means that the location 0 is transformed to the new location 214,257, location 1 is transformed to the new location 261,896, and so forth.

(a) Pepper



(b) (c)

Fig. 3. The results of randomization of the image in (a). Here, the mono-alphabetic substitution cipher algorithm[11] (b) is used and (c) using our process described in Sec. 3.2.

3.3. *The modulus embedding phase on partitioned pixels*

After the aforementioned pseudo-random permutation phase, the randomized code, which is a binary string, is ready to be embedded in a cover image. To obtain high hiding-capacity and yet still keep good image quality of the stego-image, we need to estimate the hiding capacity pixel-by-pixel in the cover image. We will use a phenomenon found in human visual system (HVS), namely, hiding more data in the area where the gray values change much.

As defined in Sec. 2.2, the symbol y_{ij} denotes the gray value of the pixel at position (i, j) in the cover image. Apparently, $0 \leq y_{ij} \leq 255$, and the cover image

is $\{y_{ij} : 0 \le i \le h-1, \text{ and } 0 \le j \le w-1\}$ if the size of the cover image is h-by-w. Analogously, let $\hat{y}_{ij}$ denote the gray value of the pixel at position (i, j) in the stego-image. Then, for each position (i, j), to create pixel (i, j) of the stego-image, the variance var_{ij} is defined according to the values of the three already-obtained adjacent "stego" pixels $(i, j-1)$, $(i-1, j-1)$, and $(i-1, j)$, with lead pixel (i, j) when the image is transformed from cover to stego. Without the loss of generality, we may just define

$$var_{ij} = \begin{cases} \infty & \text{if } i = 0 \text{ or } j = 0; \\ \dfrac{(\hat{y}_{ij-1} - \bar{y})^2 + (\hat{y}_{i-1j-1} - \bar{y})^2 + (\hat{y}_{i-1j} - \bar{y})^2}{3} & \text{otherwise,} \end{cases} \tag{8}$$

where $\bar{y}$ denotes the mean of $\hat{y}_{ij-1}$, $\hat{y}_{i-1j-1}$, and $\hat{y}_{i-1j}$. Note that, for simplicity, the var_{ij} value of the pixels which are either in the first row or in the first column of the cover image are set to infinity. After calculating the var_{ij} of each pixel in the cover image, the pixels of the cover image are classified into two groups: one is the edge group G_E which contains those pixels whose var_{ij} values are larger than a predetermined threshold V_{cut}, and the other group is the smooth group G_S which contains pixels whose var_{ij} values are not more than V_{cut}. Apparently, according to Human Visual System, we can hide more bits in the pixels in edge group than in the smooth group. Therefore, a modulus function with a large modulus base M_E will be applied to embed data in G_E, and another modulus function with a small modulus base M_S is utilized to embed data in G_S.

The embedding procedure using variance-based modulus function is as follows. According to the raster scan-order, we sequentially take a pixel (i, j) from the cover image, and call its pixel value $y_{ij} (0 \le y_{ij} \le 255)$. Then, if the var_{ij} value of this pixel is greater than V_{cut}, we set the modulus base M_O to M_E; else set M_O to M_S. Now, sequentially grab next $\lfloor \log_2 Mo \rfloor$ not-yet-embedded bits from the randomized code to form a short length data x $(0 \le x < M_O)$. Then we embed x in pixel (i, j) by replacing its gray value from y_{ij} to the resulting pixel value $\hat{y}_{ij}$. The embedding equation is

$$\hat{y}_{ij} = x + M_O \times \text{rounding}\left(\frac{y_{ij} - x}{M_O}\right) \tag{9}$$

where the rounding$(\cdot)$ operator means rounding its content to the nearest integer. Of course, we need to check whether $\hat{y}_{ij}$ falls in the valid range $0 \le \hat{y}_{ij} \le 255$. If it is out of the range, then $\hat{y}_{ij}$ should be adjusted by

$$\hat{y}_{ij} = \begin{cases} \hat{y}_{ij} + M_O & \text{if } \hat{y}_{ij} < 0; \\ \hat{y}_{ij} - M_O & \text{if } \hat{y}_{ij} > 255. \end{cases} \tag{10}$$

By sequentially processing each pixel (i, j) of the cover image using the above proposed embedding procedure introduced here, the randomized code produced from Sec. 3.2 can be embedded in the pixels of the cover image.

Notably, in the above, about whether M_E or M_S should be used, the decision rule is according to the variance rather than the pixel value itself. Below we

(a) (b)

Fig. 4. The stego-images obtained by embedding a 256×512 important image Jet [Fig. 6(a)] in a 512×512 Lena (Fig. 5). (a) The stego-image obtained by the pixel-based method[13] (PSNR = 31.05 dB); (b) the stego-image by using our variance-based method (PSNR = 34.76 dB).

explain why. Figure 4 is the stego-image obtained when our embedding process is replaced by the embedding method introduced in Ref. 13, which are quite similar to ours, except that they used pixel value directly (rather than using variance) in the decision rule. To obtain Fig. 4(a), the 256×512 important image "Jet" shown in Fig. 6(a) is embedded in Fig. 5(a). The threshold V (see Sec. 2.2) is set to 160, and two modulus bases are set to $M_U = 32$ and $M_L = 16$, the same as suggested in Ref. 13. In other words, when the gray value of a cover image pixel is larger than 160, five bits of the hidden data are embedded in this pixel. On the other hand, when the gray value is not more than 160, then only four bits of the hidden data are embedded in this pixel. Obviously, there are some pockmarks on the shoulder and face of Lena in Fig. 4. Artificial distortion appears on the shoulder and forehead of Lena where pixel values vary smoothly and yet the gray values are often larger than 160. Hence, the pixel value might not be a suitable criterion to estimate the hiding capacity of a pixel in the cover image.

3.4. *The extraction procedure for the decoding*

The extraction steps for revealing the important image are as follows.

(a) For a given stego-image, we first use Eq. (8) to compute the var_{ij} value of each pixel (i, j) of the stego-image, and then the modulus base M_O of each pixel can be determined by comparing var_{ij} with the threshold V_{cut}.
(b) Extract the randomized MSOC code (a binary stream) from the stego-image. This is achieved by sequentially processing each pixel of the stego-image, with

the raster scan order, and the extraction equation

$$x = \hat{y}_{ij} \pmod{M_O} \tag{11}$$

to obtain the hidden data x from pixel (i, j). Of course, convert each x ($0 \leq x < M_O$) to its binary equivalent before processing the next pixel of the stego-image.

(c) Obtain the same series of pseudo-random fraction numbers, which are used in the pseudo-random permutation phase (Sec. 3.2), by running the MT pseudo-random number generator[9] with the same seed. Then, do inverse permutation to obtain the nonrandomized MSOC code stream.

(d) Fetch a bit sequentially from the (remaining) MSOC code stream; it is an indicator bit in our MSOC scheme.

(e) If the indicator bit obtained in step (d) is 0, fetch N bits from the remaining MSOC code stream. Then, according to these N bits and our MSOC predetermined search path, locate the corresponding search position in the partially reconstructed important image. Use the pixel value at the pointed pixel position to paint the gray value of the current pixel position in the reconstructed image.

(f) If the flag bit obtained in step (d) is 1, fetch eight bits from the remaining MSOC code stream, and directly assign these eight bits to the pixel value of the current position in the reconstructed image.

(g) Repeat steps (d) through (f) until all of the data in the MSOC code stream are processed.

(h) The generated image is the revealed important image.

4. Experimental Results

All images in the experiments are eight-bit gray-scaled. In each experiment, one of the important images is embedded in the cover image. As suggested in Ref. 5, we set $N = 2$, i.e. we also use two-bit string to record each MSOC position. As for the value of our threshold V_{cut} and the values of the two modulus bases M_E and M_S, they are all dynamic according to the total size of MSOC code. (Notably, the total size of MSOC code is affected by the threshold value T in turn.)

In the MSOC encoding algorithm, set T to 1 if the extracted important image is required to be error-free. On the other hand, use a larger value of T if the cover image is not much bigger than the important image; for example, when both images are of 512×512.

In the first experiment we hide an important image in a cover image of the same size. Figure 5 shows an example of the hiding result in this experiment, where Fig. 5(a) is the cover image "Lena" of size 512×512, and Fig. 5(b) is the important image "Jet" of size 512×512. The obtained stego-images using thresholds $T = 7$ and 9 are as shown in Figs. 5(c) and 5(d), respectively. The corresponding lossy version of the important images extracted from Figs. 5(c) and 5(d) are shown in Figs. 5(e) and 5(f), respectively. Finally, the settings of the experimental

Fig. 5. The first experiment. (a) The cover image "Lena" with 512×512 pixels; (b) the important image "Jet" with 512×512 pixels; (c) and (d) the stego-images obtained by embedding Fig. 5(b) in Fig. 5(a) with threshold values $T = 7$ and 9, respectively; (e) and (f) the extracted important images from Figs. 5(c) and 5(d), respectively.

Table 1. The parameters used in our experiments to embed an important image of size 512×512. The PSNR values (marked as "stego") are between the stego-image and cover image, and the PSNR values (marked as "extracted") are between the extracted important image and original important image.

Cover Image	Important Image	Parameters and PSNR	$(M_E = 16, M_S = 8)$				
			$T = 7$	$T = 8$	$T = 9$	$T = 10$	$T = 11$
Lena	Jet (512×512)	Threshold (V_{cut})	14	16	20	24	32
		PSNR (stego)	37.12	37.56	38.05	38.31	38.70
		PSNR (extracted)	41.77	40.55	39.50	38.77	37.87
	Tiffany (512×512)	Threshold (V_{cut})	29	42	55	70	98
		PSNR (stego)	38.62	38.96	39.40	39.63	39.83
		PSNR (extracted)	41.43	40.15	39.36	38.66	37.66

parameters, and the PSNR values of the stego-images and the extracted versions of the important image, are summarized in Table 1. From Table 1, we can see that the qualities of the stego-images are acceptable (the PSNRs between the stego-images and the original cover image are all greater than 36.0 dB). It is also hard to distinguish between Fig. 5(b) and those in Figs. 5(e) and 5(f) using naked eyes, and the PSNRs between the extracted lossy versions and the original version of the important image are all greater than 37.0 dB. Note that the extraction is lossy because we try to embed a 512×512 image in another 512×512 image of the same size, which is just impossible for us to set $T = 1$.

In the second experiment we hide an important image whose size is half of the cover image. Figure 6 shows some examples of the hiding result in this experiment. The settings of the experimental parameters, and the PSNRs of the stego-images and the extracted important images, are listed in Table 2. From the PSNRs in Table 2, we can see that the qualities of the stego-images are good, for the PSNRs between the stego-images and the original cover image are all between 40.0 dB and 46.8 dB. The qualities of the extracted important images are also high, for the PSNRs between the extracted important images and the original important image are between lossless (∞) and 44.4 dB. Notably, the recovered important image is error-free when we use the most strict value $T = 1$. As a remark, when $T = 3$, the M_E and M_S computed by Step (3) of Sec 5.1 are ($M_E = 8$, $M_S = 4$) for image Jet, different from the ($M_E = 4$, $M_S = 2$) for image Tiffany; this should be of no surprise because the total number of bits in the produced MSOC codes are different for these two images.

To know the performance of our scheme, the PSNRs of the stego-image obtained in our scheme are compared in Table 3 with those elegant methods reported in literature. Note that in Ref. 6, the important image is first compressed using the VQ technique, and then the VQ indices and the encoded codebook are embedded in a cover image by the LSB substitution method. Therefore, in the comparison with Ref. 6, we also use the VQ technique to compress the important image before

Fig. 6. The second experiment about hiding the middle-size (256×512 pixels) important image. (a) The important image "Jet"; (b) the important image "Tiffany"; (c) and (d) the stego-images obtained by embedding Figs. 6(a) and 6(b) in Fig. 5(a) with threshold values $T = 1$ and 3, respectively; (e) and (f) the extracted important images from Figs. 5(c) and 5(d), respectively. Note that (e) is identical to Fig. 6(a) without any loss.

hiding, and then encode the VQ index file using the MSOC algorithm with the threshold $T = 1$ before using the pseudo-random permutation or the modulus embedding procedure. Of course, both the MSOC output code and the encoded codebook are randomized and embedded in the cover image. In Table 3, the symbol α means that the experiment used this special VQ approach.

Table 2. The parameters used in our experiments to embed the important image of size 256×512. The PSNR values (marked as "stego") are between the stego-image and cover image, and the PSNR values (marked as "extracted") are between the extracted important image and original important image.

Cover Image	Important Image	Parameters and PSNR	The Tolerance Parameter T				
			$T = 1$	$T = 2$	$T = 3$	$T = 4$	$T = 5$
Lena	Jet (256×512)	M_E/M_S	$M_E = 16$, $M_S = 8$	$M_E = 8$,	$M_S = 4$	$M_E = 4$,	$M_S = 2$
		Threshold (V_{cut})	90	12	130	0.6	0.8
		PSNR (stego)	39.80	43.68	45.75	46.61	46.83
		PSNR (extracted)	Error-free	52.83	49.12	46.22	44.40
	Tiffany (256×512)	M_E/M_S	$M_E = 16$, $M_S = 8$	$M_E = 8$, $M_S = 4$	$M_E = 4$, $M_S = 2$		
		Threshold (V_{cut})	16	11	0.2	1	1.5
		PSNR (stego)	37.54	43.57	46.43	46.94	47.18
		PSNR (extracted)	Error-free	52.18	48.40	45.68	43.80

Notably, Refs. 1, 12–14, 16 did not process any important image of size as big as the 512×512 cover image, so we only list in Table 3 the experimental data copied from Refs. 3 and 6 when the important image is 512×512.

In order to make a fair comparison, we show the results of our two versions in Table 3. One version is without MSOC compression, the other version is with MSOC compression. The MSOC version has a PSNR much better than the PSNRs of all other listed methods; and this should be of no surprise because MSOC reduces the size of the important image. Below we focus on the without-MSOC version.

Since Ref. 1 outperformed the remaining reported methods listed in Table 3, we compare our without-MSOC version with Ref. 1, as follows. When the bpp (bits per pixel, here interpreted as the number of bits in the important image over the number of pixels in the cover image) is an integer, then the PSNR of the stego-image in our without-MSOC version is less than, but very close to, that of Ref. 1. The phenomenon can be seen in Table 3. For instance, when we embed the 256×256 Tiffany (or 256×512 Tiffany) in a 512×512 Lena, the bpp $256 \times 256 \times 8/512 \times 512 = 2$ (or $256 \times 512 \times 8/512 \times 512 = 4$) is an integer; then the stego-image's PSNR is 46.33 dB (or 34.74 dB) in our method, and 46.37 dB (or 34.84 dB) in Ref. 1. [The difference is only 0.04 dB (0.1 dB).]

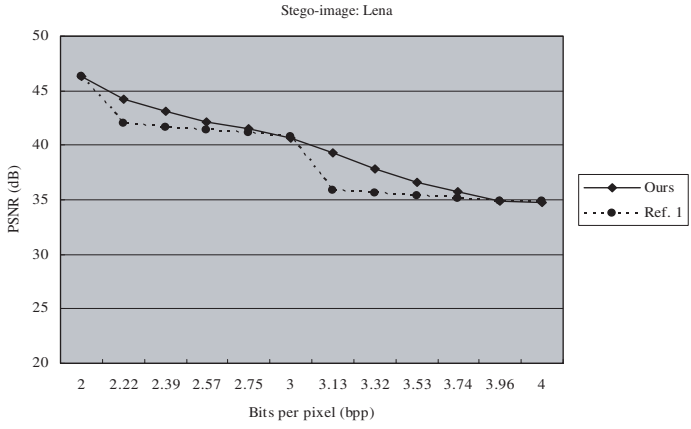
However, when the bpp is not an integer, then our stego-image's PSNR (the without-MSOC version) is higher than that of Ref. 1. To show this, we embed the important image Jet of various sizes into the 512×512 cover image Lena [Fig. 5(a)], without using MSOC. The experimental results are shown in Fig. 7(a). One such example is to embed a 320×320 Jet into the 512×512 Lena, which means the bpp is $320 \times 320 \times 8/(512 \times 512) = 3.125$; and it is found that our 39.32 dB stego-image is better than the 35.90 dB stego-image of Ref. 1. Figure 7(b) is the

Table 3. A comparison between some published methods and our scheme.

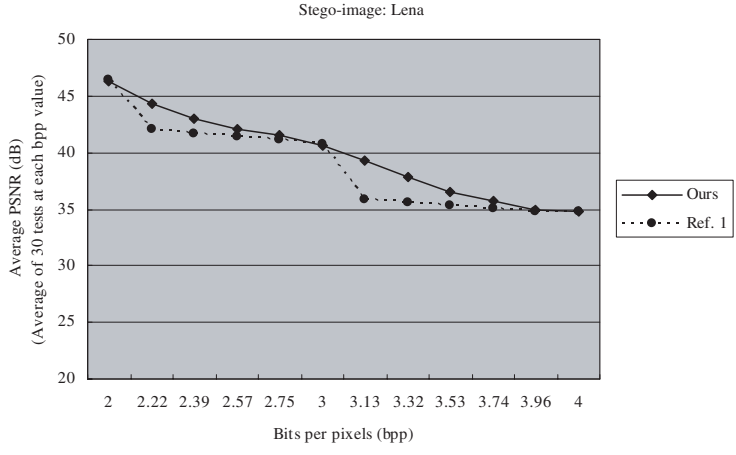
Cover Image	Scheme	Size of the Important Image	PSNR of Extracted	PSNR of Stego-Image
Lena (Baboon)	[3]	256 × 256 Tiffany	Error-free	44.90*(44.90*)
	[3]	256 × 256 Jet	Error-free	44.53* (44.54*)
	[4]	256 × 256 Tiffany	Error-free	46.37* (46.38*)
	[9]	256 × 256 Jet	Error-free	38.85*
	[12]	256 × 256 Jet	Error-free	44.08* (44.03*)
	ours without MSOC	256 × 256 Tiffany	*Error-free*	*46.33 (46.32)*
	ours without MSOC	256 × 256 Jet	*Error-free*	*46.33 (46.31)*
	ours	256 × 256 Tiffany	*Error-free*	*47.52 (47.58)*
	ours	256 × 256 Jet	*Error-free*	*47.76 (47.78)*
	[3]	256 × 512 Tiffany	Error-free	32.90* (32.95*)
	[3]	256 × 512 Jet	Error-free	32.71* (32.79*)
	[4]	256 × 512 Tiffany	Error-free	34.84* (34.79*)
	[11]	256 × 512 Tiffany	Error-free	34.80*
	[11]	256 × 512 Jet	Error-free	34.76*
	[12]	256 × 512 Jet	Error-free	31.05* (31.09*)
	ours without MSOC	256 × 512 Tiffany	*Error-free*	*34.74 (34.73)*
	ours without MSOC	256 × 512 Jet	*Error-free*	*34.76 (34.73)*
	ours	256 × 512 Tiffany	*Error-free*	*37.54 (37.81)*
	ours	256 × 512 Jet	*Error-free*	*39.80 (39.79)*
	[6]	512 × 512 Jet	30.01*	32.50*
	[7]	512 × 512 Tiffany	*32.02**	44.42*
	ours$^{\alpha}$ without MSOC	512 × 512 Tiffany	*32.02$^{\alpha}$*	*51.68$^{\alpha}$ (51.69$^{\alpha}$)*
	ours$^{\alpha}$ without MSOC	512 × 512 Jet	*31.43$^{\alpha}$*	*51.69$^{\alpha}$ (51.69$^{\alpha}$)*
	ours	512 × 512 Tiffany	*32.02$^{\alpha}$*	*53.37$^{\alpha}$ (53.37$^{\alpha}$)*
	ours	512 × 512 Jet	*40.55*	*37.69 (37.81)*

("*" means "quoted from the reported papers", and "α" means "the input of the MSOC is the VQ index file of the important image, rather than the important image itself")

average of 30 tests (30 tests for each bpp value), in which each test uses one the 30 important images {Lena, Jet, Tiffany, Baboon, Barbara, Boat, Bridge, Couple, Elaine, Family, Gold, House, Milk, Painting, Pepper, Scene, Tank, Toys, Woman, Zelda, Girl, Cameraman, Beach, Car, Iran, Logo, Map, Mickey, Satellite image, X-ray image} of a specified size. In general, as shown in Fig. 7, if the bpp is not an integer, then our stego-images' PSNR are better than the stego-images' PSNR of Ref. 1. This is because Ref. 1 was originally designed to use t-bits LSB in a finer manner, and their t were integers; as a result, when the number of bits in the important image was, for example, 3.125 times larger than the number of pixels in the cover image, then they could not use $t = 3$ bits LSB; instead, they needed to



(a)



(b)

Fig. 7. The comparison of stego-image's quality between Ref. 1 and our no-MSOC version, when the cover (stego) image is Lena. (a) Jet is used as the important image; (b) the average of testing 30 important images (so, 30 tests for each bpp value).

use $t = 4$. Therefore, their distortion curve has a sudden jump (while our PSNR curve has no such jump).

In conclusion, as shown in Fig. 7, when the number of "bits" in the important image is an integer multiple of the number of "pixels" in the cover image, i.e. when the bpp is an integer, then use Ref. 1, for their PSNR is better than ours (they lead us by an amount between 0.04 dB and 0.1 dB in Table 3). However, when the bpp is not an integer, then use ours.

As for the experiment done by Wu and Tsai,[15] they used a Word-format file, rather than an image, as the hidden important data. To compare with Ref. 15, we

try to embed a Word-format file with 50,960 bytes in an extra experiment. If the cover image is the 512×512 Lena, the PSNR of the stego-image Lena obtained by Ref. 15 is 41.79 dB. On the other hand, our stego-image's PSNR is 47.65 dB even if we do not use the MSOC compression (the PSNR would have been better than 47.65 dB if MSOC had been used). Analogously, when trying to embed a Word-format file of 25,940 bytes, their PSNR is 48.43 dB and ours is 51.89 dB when MSOC is not used. In summary, our method can also compete with Ref. 15.

About the (net) embedding rate of our scheme, the MSOC reduced the 256×256 Jet (and 256×256 Tiffany) from $256 \times 256 = 65{,}536$ bytes to 51,817 bytes (and 53,467 bytes) with threshold $T = 1$ before embedding. Similarly, the MSOC reduced the 256×512 Jet and 256×512 Tiffany from $256 \times 512 = 131{,}072$ bytes to 100,989 bytes (and 109,054 bytes) with threshold $T = 1$ before embedding. So, the (pure) embedding rate for the 512×512 cover image Lena mentioned in the experiments and Table 3 is $51{,}817 \times 8/(512 \times 512) = 1.58$ bits per pixel (bpp) for the compressed code of 256×256 Jet (and $53{,}467 \times 8/(512 \times 512) = 1.63$ bpp for 256×256 Tiffany), and $100{,}989 \times 8/(512 \times 512) = 3.08$ bpp for the compressed code of 256×512 Jet (and $109{,}054 \times 8/(512 \times 512) = 3.33$ bpp for 256×512 Tiffany).

In summary, to hide Jet's MSOC code, according to Table 3, we obtained 47.76 dB stego-image Lena when the (net) hiding rate is 1.58 bits per pixel; and obtained 39.80 dB stego-image Lena when the (net) hiding rate is 3.08 bits per pixel. Analogously, to hide the source data of image Jet directly without using compression, we obtained 46.33 dB stego-image Lena when the hiding rate is $256 \times 256 \times 8/(512 \times 512) = 2$ bits per pixel; and obtained 34.76 dB stego-image Lena when the hiding rate is $256 \times 512 \times 8/(512 \times 512) = 4$ bits per pixel.

To ensure that the proposed method can work well for most of the images, we test 30 important images {Lena, Jet, Tiffany, Baboon, Barbara, Boat, Bridge, Couple, Elaine, Family, Gold, House, Milk, Painting, Pepper, Scene, Tank, Toys, Woman, Zelda, Girl, Cameraman, Beach, Car, Iran, Logo, Map, Mickey, Satellite image, X-ray image}. Each image has three kinds of size: 256×256, 256×512, and 512×512. So there are in fact $30 \times 3 = 90$ images. Each time one of these 90 important images is hidden in the 512×512 cover image Lena shown in Fig. 5(a). The results are shown in Table 4, from where we can see that, to hide a 256×512 important image using the threshold $T = 3$, the PSNR of the extracted image is about 48.89 dB and the PSNR of the stego-image is about 43.97 dB. So both the qualities of the extracted and stego-image are acceptable. Analogously, when we use $T = 8$ to hide one of the 30 important images whose sizes are all as large as the 512×512 cover image, the PSNR of the extracted image is about 39.88 dB in average, and the PSNR of the stego-image is about 37.24 dB in average. Therefore, the proposed method can still work for most of the large images.

We also try another 30 tests; and the cover image in each test is one of the thirty 512×512 images {Lena, Jet, Tiffany, Baboon,..., Mickey, Satellite image, X-ray image} mentioned above. Table 5 summarizes these 30 tests. From Table 5, we can

Table 4. The results of testing 30 important images.

Size of each Important Image	Versions	30 Extracted Images		30 Stego-Images	
		PSNR (Range)	SNR (Average)	PSNR (Range)	PSNR (Average)
256×256	No MSOC	Error-free	Error-free	46.31–46.35	46.335
	$T = 1$ in MSOC	Error-free	Error-free	46.08–51.40	47.480
256×512	$T = 3$ in MSOC	47.75–52.29	48.886	38.16–47.84	43.968
512×512	$T = 8$ in MSOC	38.64–43.66	39.881	30.60–40.56	37.242

Table 5. The results of testing 30 cover images.

Cover Images	Scheme	Size of the Important Image	PSNR (extracted)	PSNR (stego)
30 tests using 30 cover images	*ours without MSOC*	256×256 Tiffany	*Error-free*	*46.28–46.34*
	ours without MSOC	256×256 Jet	*Error-free*	*46.27–46.35*
	ours	256×256 Tiffany	*Error-free*	*47.51–47.61*
	ours	256×256 Jet	*Error-free*	*47.62–47.81*
	ours without MSOC	256×512 Tiffany	*Error-free*	*34.71–34.78*
	ours without MSOC	256×512 Jet	*Error-free*	*34.70–34.79*
	ours	256×512 Tiffany	*Error-free*	*37.50–37.86*
	ours	256×512 Jet	*Error-free*	*39.76–39.83*
	ours$^{\alpha}$ *without MSOC*	512×512 Tiffany	*32.02*$^{\alpha}$	*51.65*$^{\alpha}$*–51.72*$^{\alpha}$
	ours$^{\alpha}$ *without MSOC*	512×512 Jet	*31.43*$^{\alpha}$	*51.66*$^{\alpha}$*–51.72*$^{\alpha}$
	ours$^{\alpha}$	512×512 Tiffany	*32.02*$^{\alpha}$	*53.32*$^{\alpha}$*–53.44*$^{\alpha}$
	ours$^{\alpha}$	512×512 Jet	*31.43*$^{\alpha}$	*53.45*$^{\alpha}$*–53.58*$^{\alpha}$
	ours	512×512 Tiffany	*40.15*	*38.74–39.21*
	ours	512×512 Jet	*40.55*	*37.36–37.83*

"α" means "the input of the MSOC is the VQ index file of the important image, rather than the important image itself".

see that the proposed method is stable (has a narrow range) for using different images as the cover image.

As for the processing time, the average encoding/decoding time of our method are shown in Tables 6 and 7. From Tables 6 and 7, we can see that the larger the size of the important image, the higher is the encoding/decoding time. Note that, all programs in the current paper were implemented by using the Borland C++ Builder

Table 6. The encoding time of the separate procedure in our method (unit: second).

Size of Important Image	MSOC	Permutation	Embedding	Total
256×256	0.203	0.406	0.078	0.687
256×512	0.375	0.483	0.085	0.943
512×512	0.723	0.667	0.093	1.483

Table 7. The decoding time of our method (unit: second).

Size of Important Image	Decoding
256×256	0.599
256×512	0.734
512×512	1.197

6.0, and ran on a notebook with Intel Pentium Processor M-740 (1.73 GHz) and 512 MB RAM under the operation system of Microsoft Windows XP Professional.

5. Discussions

5.1. *Parameter setting*

Because the extracted image is the compression result of the important image with the threshold T, the quality of the extracted image is determined by the threshold T. (The larger the threshold T, the higher is the compression ratio; i.e. the larger the threshold T, the lower is the quality of the extracted "important image".)

On the other hand, once the MSOC code (the compression result) is generated, the MSOC code is fixed (so the quality of the important image recovered in the future is also fixed). To hide this fixed MSOC file, we determine the suitable V_{cut} value so that we can avoid causing too much distortion to the cover image (as long as the "whole" MSOC code file can be hidden in the cover image completely). So, the V_{cut} value determines the quality of the "stego-image". In general, the larger the V_{cut} value, the better is the quality of the stego-image. However, the V_{cut} value cannot be too large; otherwise, there will not be enough space in the cover image to hide the whole MSOC code file.

Therefore, about parameters setting, we proceed as follows

(1) We first determine a value of T according to the size of the important image. (For example, Table 1 uses a larger T (between 7–11) because its important image size 512×512 is larger than the important image size 256×512 in Table 2, which uses a smaller T (between 1–5).
(2) Then get the MSOC code corresponding to this T.
(3) Since we want to hide the MSOC code in the cover image, the bits per pixel (bpp) used to measure the (net) embedding rate is

$$\text{bpp} = \frac{\text{The total number of bits in the MSOC code}}{\text{The total pixels of the cover image}}. \tag{12}$$

Then, let $M_S = 2^{\lfloor \text{bpp} \rfloor}$ and $M_E = 2^{\lceil \text{bpp} \rceil}$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are the ceiling and floor functions to get two closest integers of bpp.
(4) Then, to avoid damaging the cover image too much, we start from a large value of V_{cut}.

(5) If the MSOC code cannot be embedded completely in the cover image using this V_{cut} value, then try to hide again using a smaller V_{cut} value. Repeat reducing the V_{cut} value if necessary. □

As a final remark of this Sec. 5.1, note that in our method, since the input to the embedding algorithm (Sec 3.3), is in fact the MSOC code, and the MSOC code can be embedded in cover image and extracted from cover image without any error; therefore, BER = 0 always holds (BER stands for bit-error-rate). The fact that BER is always 0 has nothing to do with parameter setting.

5.2. *Using pseudo-random process to increase security level*

As in other image-hiding techniques, our stego-images are utilized to reduce the attention of hackers. However, because the parameters V_{cut}, M_S, and M_E are not necessarily constant, we transmit the values of these parameters along with the stego-image to the receiver. If a hacker happens to monitor the transmission of a stego-image, and if he also knows the number of bits (N) used to record MSOC, then the MSOC code of the important image might be extracted completely, and the important image is then exposed. (For example, if the hacker has got a MSOC code $(\underline{\mathbf{1}}01110110\underline{\mathbf{0}}00\underline{\mathbf{1}}01111001)_2$, he can decode the code according to the MSOC algorithm described in Sec. 3.1 to obtain that the three pixels are [OPV(118), MSOC(0), OPV(121)], and then know the three pixel values are [118, 118, 121].) Therefore, the pseudo-random process is needed to prevent the important image from being revealed. The protection is through randomizing the MSOC code before embedding. To randomize a binary string of length S, there are $C(S,K) = \frac{S!}{(S-K)!K!}$ possible combinations where K is the number of 1 in the binary string. The possibility of guessing the right solution is only $\frac{1}{C(S,K)}$. In the above example, the possibility is $\frac{1}{C(21,12)} = \frac{1}{293{,}930}$. In our experiment, which embeds 256×256 Jet into 512×512 Lena with threshold $T = 3$, the size of the MSOC code is 285,762, and K is 106,090, so the possibility is $1/C(285{,}762,\ 106{,}090)$. Obviously, there is a very small chance for the hacker to guess the right combination to obtain the important image. Hence, the pseudo-random process is utilized to prevent the important image from being divulged. Note that the seed of the pseudo-random number generator[9] can be a private key known to the sender and the receiver. The hacker can hardly know the important image without the private key, even if he monitors the whole process of transmission, and picks the right stego-images from a bunch of coy images sent along with these stego-images.

6. Summary

In this paper, a steganography scheme is proposed. The scheme contains three parts: (1) MSOC encoding; (2) pseudo-random permutation; and (3) an embedding process using variance-based modulus function. The MSOC method makes the

important image more compact and hence more suitable for the embedding procedure later. A user-specified non-negative integer threshold T is introduced to control the length of the MSOC code, which in turn affects later the quality of the extracted important image. If the size of the important image is not small, then we might need to use a larger value of T to handle the case. However, if the size of the important image is small, then the readers may just use $T = 1$ so that the extracted important image is error-free. Therefore, the use of T provides more flexibility for practical applications. To increase the security level, a pseudo-random permutation algorithm has been utilized by applying the MT pseudo-random number generator.[9] In the embedding part, we use a variance-based criterion to estimate the hiding capacity of a pixel in the cover image. The criterion is based on human visual system (HVS): more bits can be hidden in a pixel of busy area. From the experimental results, it can be seen that the variance-based estimation is more suitable than its counterpart in Ref. 13 which uses the pixel value to estimate the hiding capacity of a pixel.

Experimental results show that the quality of both the stego-images and extracted important images are competitive to those obtained in many existing steganograpy methods reported recently. From Table 3, it can be seen that the proposed method can create low-profile stego-images to protect the important image (because stego-images are with competitive qualities), and yet preserve the fidelity of the important image. The cover images are not necessarily bigger than the important images in our approach. In the future, we might try other related topics, such as digital watermarking[7,8] or corresponding applications.

Acknowledgments

This work was supported by National Science Council, Republic of China, under grant NSC962221-E-009-039. The authors would like to thank the editor and the three reviewers for their valuable suggestions.

References

1. C. K. Chan and L. M. Cheng, Hiding data in images by simple LSB substitution, *Patt. Recogn.* **37**(3) (2004) 469–474.
2. C. C. Chang, M. H. Lin and Y. C. Hu, A fast and secure image hiding scheme based on LSB substitution, *Int. J. Patt. Recogn. Artif. Intell.* **16**(4) (2002) 399–416.
3. K. L. Chung, C. H. Shen and L. C. Chang, A novel SVD- and VQ-based image hiding scheme, *Patt. Recogn. Lett.* **22** (2001) 1051–1058.
4. R. M. Davis, The data encryption standard in perspective, *Computer Security and the Data Encryption Standard*, National Bureau of Standards Special Publication (February, 1978).
5. C. H. Hsieh and J. C. Tsai, Lossless compression of VQ index with search-order coding, *IEEE Trans. Imag. Process.* **5**(11) (1996) 1579–1582.
6. Y. C. Hu and M. H. Lin, Secure image hiding scheme based upon vector quantization, *Int. J. Patt. Recogn. Artif. Intell.* **18**(6) (2004) 1111–1130.

7. Y. C. Hu and M. H. Lin, Fast watermark detection scheme from camera-captured images on mobile phones, *Int. J. Patt. Recogn. Artif. Intell.* **20**(4) (2006) 543–564.
8. D. C. Lou, J. M. Shieh and H. K. Tso, A robust buyer-seller watermarking scheme based on DWT, *Int. J. Patt. Recogn. Artif. Intell.* **20**(1) (2006) 79–90.
9. M. Matsumoto and T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator, *ACM Trans. Model. Comput. Simul.* **8**(1) (1998) 3–30.
10. F. A. P. Petitcolas, R. J. Anderson and M. G. Kuhn, Information hiding — a survey, *Proc. IEEE* **87**(7) (1999) 1062–1078.
11. M. Y. Rhee, *Cryptography and Secure Communication* (McGraw-Hill Book Co, Singapore, 1994).
12. C. C. Thien and J. C. Lin, A simple and high-hiding capacity method for hiding digit-by-digit data in images based on modulus function, *Patt. Recogn.* **36**(12) (2003) 2875–2881.
13. S. J. Wang, Steganography of capacity required using modulo operator for embedding secret image, *Appl. Math. Comput.* **164** (2005) 99–116.
14. R. Z. Wang, C. F. Lin and J. C. Lin, Image hiding by optimal LSB substitution and genetic algorithm, *Patt. Recogn.* **34**(3) (2001) 671–683.
15. D. C. Wu and W. H. Tsai, A steganographic method for images by pixel-value differencing, *Patt. Recogn. Lett.* **24** (2003) 1613–1626.
16. H. C. Wu, N. I. Wu, C. S. Tsai and M. S. Hwang, Image steganographic scheme based on pixel-value differencing and LSB replacement methods, *IEE Proc. Vis. Imag. Sign. Process.* **152**(5) (2005) 611–615.

Yu-Jie Chang received the B.S. degree in computer science and information engineering in 1999 from National Central University, Taiwan. In 2001, he received his M.S. degree in computer and information science from National Chiao Tung University. He is now a Ph.D. candidate in the Computer Science Department of National Chiao Tung University.

His research interests include digital watermarking, image processing, and pattern recognition

Ran-Zan Wang received the B.S. degree in computer engineering and science in 1994 and M.S. degree in electrical engineering and computer science in 1996, both from Yuan-Ze University, Taiwan, R.O.C. In 2001, he received his Ph.D. degree in computer and information science from National Chiao Tung University, Taiwan. He is currently an Associate Professor in the Department of Computer Engineering and Science at Yuan Ze University, Taiwan.

Dr. Wang is a member of the Phi-Tau-Phi Scholastic Honor Society.

His recent research interests include media security, image processing, and pattern recognition.

Ja-Chen Lin received his B.S. degree in computer science in 1977 and M.S. degree in applied mathematics in 1979, both from National Chiao Tung University (NCTU), Taiwan. In 1988, he received his Ph.D. degree in mathematics from Purdue University, USA. During 1981 1982, he was an instructor at NCTU. From 1984 to 1988, he was a graduate instructor at Purdue University. He joined the Department of Computer and Information Science at NCTU in August 1988, and became a professor there.

Dr. Lin is a member of the Phi-Tau-Phi Scholastic Honor Society.

His research interests include pattern recognition and image processing.