Soft Comput (2009) 13:321–331
DOI 10.1007/s00500-008-0332-x

FOCUS

A data embedding scheme for color images based on genetic algorithm and absolute moment block truncation coding

Chin-Chen Chang · Yi-Hui Chen · Chia-Chen Lin

Published online: 27 May 2008

Abstract Recently, embedding a large amount of secret data into gray-level and color images with low distortion has become an important research issue in steganography. In this paper, we propose a data embedding scheme by using a well-known genetic algorithm, block truncation code and modification direction techniques to embed secret data into compression codes of color images to expand the variety of cover media. In the scheme, the common bitmap generation procedure of GA-AMBTC has been modified to speed up the hiding procedure. Two embedding strategies are proposed to hide secret data into the common bitmap and the quantization values in each block of the cover image. Experimental results confirm that the proposed scheme can provide high data capacity with acceptable image quality of the stego-images. Moreover, the compression ratio of the scheme is exactly the same as that of GA-AMBTC so that attackers cannot detect any trace of hidden data from the size of the modified compressed result.

Keywords Data embedding · Steganography · Genetic algorithm · Block truncation code · Modification direction

C.-C. Chang (✉)
Department of Information Engineering and Computer Science, Feng Chia University, 100 Wenhwa Rd., Seatwen, Taichung 40724, Taiwan, R.O.C.
e-mail: ccc@cs.ccu.edu.tw

C.-C. Chang · Y.-H. Chen
Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi 621, Taiwan, R.O.C.
e-mail: chenyh@cs.ccu.edu.tw

C.-C. Lin
Department of Computer Science and Information Management, Providence University, Taichung 43301, Taiwan, R.O.C.
e-mail: mhlin3@pu.edu.tw

1 Introduction

Due to the emergence and flourishing of network technologies, the Internet has become a very popular means of communication. Although data can be transmitted over the Internet without geographic restriction, the transmitted data are at risk. While legitimate senders and receivers can easily transmit information, malicious attackers can easily grab or tamper transmitted data from the Internet, especially valuable data, such as military maps and circuit diagrams. Therefore, security has become an increasingly important issue.

The two kinds of techniques that are generally used to protect data are cryptography and data hiding. In the former technique, the digital data or secret data are encrypted by the sender to become ciphers using a public key cryptography system such as RSA (Rivest et al. 1978). Then, the sender transmits the ciphers to the receiver side over the Internet. Later, the receiver can decrypt the ciphers by using a corresponding secret key to obtain the correct data. However, the ciphers are usually meaningless, so attackers may further test these data because they appear suspicious.

In the data hiding technique, the senders embed the secret data into a digital media called the cover image, such as the "Mona Lisa," to become a stego-image. Then the senders send this stego-image to the receiver side. Later, receivers can retrieve the secret data from the digital media. Because the stego-image is often imperceptible and statistically undetectable to attackers, the embedded secret information can be transmitted securely over the Internet.

Once data embedding is used for secret transmissions, two crucial requirements must be satisfied. One is minimizing the degradation in image quality after secret data is embedded. The other is making the hiding capacity of cover media as large as possible. Basically, the image quality of stego-image and hiding capacity are tradeoffs, so that higher hiding

capacity often causes greater distortion in the cover medium, and vice versa. There is currently many literatures on not only embedding large amounts of secret data into cover images but also on obtaining stego-images of reasonable image quality (Bender et al. 1996; Chan and Cheng 2004; Chang et al. 2003, 2006; Chuang and Chang 2006; Cox et al. 1997; Du and Hsu 2003; In et al. 1999; Konstantinides et al. 1999; Lin and Tai 1998; Lu et al. 2000; Lu and Sun 2000; Nasrabadi and King 1988; Pai and Ruan 2006; Pan et al. 2004; Pfitzman 1996; Rivest et al. 1978; Shie et al. 2006; Shieh et al. 2004; Tai et al. 1998; Zhang and Wang 2006; Lee and Chen 2000).

Up to now, data embedding techniques can be roughly divided into three categories: the spatial-domain manner (Bender et al. 1996; Chan and Cheng 2004; Chang et al. 2003; Walker 1996), the frequency-domain manner (Du and Hsu 2003; In et al. 1999; Nasrabadi and King 1988) and the compression-domain manner (Cox et al. 1997; Lu et al. 2000; Rivest et al. 1978). For the spatial-domain manner, secret data are mixed directly into the distributed pixels. The least significant bit (LSB) (Bender et al. 1996; Chan and Cheng 2004; Chang et al. 2003; Walker 1996; Wang et al. 2001) is the general approach to hide the secret information into the LSBs of each pixel of a cover image directly owing to their lower distortion in image quality. For the frequency-domain manner, the cover image must first be transformed into frequency coefficients by using a frequency-oriented mechanism such as discrete Fourier transformation (DFT) (Walker 1996), discrete cosine transformation (DCT) (Chen et al. 1999), discrete wavelet transformation (DWT) (Munteanu et al. 1999), and so on. Later, the secret data are combined with the relative coefficients in the frequency-form image. Because the human eye is more sensitive to the lower-frequency band, secret data are often embedded in the higher-frequency band to obtain reasonable degradation (Du and Hsu 2003; In et al. 1999; Nasrabadi and King 1988). For the compression-domain manner, the secret data are embedded into the compression codes. Recently, the compression codes generated by well-known image compressions such as JPEG (Hu and Chang 1999; Kim 1992; Kamstra and Henk 2005), vector quantization (VQ) (Cox et al. 1997; Lu et al. 2000; Rivest et al. 1978), block truncation coding (BTC) (Chang and Hu 1999; Chen and Tai 1999; Hsu and Wu 1999) and similar methods are used for data embedding to extend the variety of cover images.

Lin and Wang (1999) embedded secret data into VQ compressed images. In their approach, a codeword in a codebook can find its similar codeword to become a similar pair, and therefore a codebook can be broken down into several similar pairs. Then these similar pairs are assigned to two sub-codebooks titled the "0-sub-codebook" and "1-sub-codebook", respectively. In this manner, each codeword in a sub-codebook can always find one codeword from the other sub-codebook that is the most similar and vice versa. Later, the codeword obtained by computing the least Euclidean distance from "0-sub-codebook" is selected to encode the current block when the secret bit is 0. Conversely, the current block is encoded by using the other codeword which is in the same pair but belongs to "1-sub-codebook" when the secret bit is 1.

Due to the fact that Lin et al.'s scheme only hides one bit per block, Lu and Sun (2000) extended Lin et al.'s method to partition the codebook into 2^k sub-codebooks to embed k bits into a single index. Du and Hsu (2003) proposed an adaptive data hiding scheme that varied their hiding capacity with the amount of secret data. The advantages of Lu and Sun's scheme include both an increase in capacity and reduction of cover image distortion; the weakness of their scheme is that their embedding procedure is very time-consuming (Shie et al. 2006), especially because the size of a secret bit stream can be up to 64 kb.

To deal with this drawback, Shie et al. (2006) applied side match vector quantization (SMVQ) (Kim 1992; Lin and Shie 2000) with the concept of prediction to propose an adaptive data hiding scheme. To maintain the visual quality of the stego-images, Shie et al. used two thresholds, TH_{var} and TH_{smd}, to adaptively select enough qualified smooth blocks from the cover image to hide secret data. However, the image quality of the stego-image in their scheme depends to a large extent on the number of smooth blocks in a cover image. Although larger thresholds allow a larger number of secret data to be hidden, they also result in larger degradation of a cover image. Chang et al. (2006) presented a reversible data hiding technique for SMVQ. To achieve reversibility, Chang et al. sorted the codebook and used adjacent indices to approximate codewords for conveying secret messages.

In addition to the VQ compression domain, Chuang and Chang (2006) proposed a data embedding scheme based on BTC for gray scale images in 2006. In their scheme, they first used a 4×4 BTC to compress a gray scale cover image directly. Therefore, the output of BTC-encoded block contains two quantization values and one bitmap. Then they predefined a threshold to classify the type of each BTC-encoded block as smooth or complex. Subsequently, they embedded the secret data into the bitmap of the selected BTC-encoded blocks. Although they proposed a new embedding scheme for BTC compressed images, it was only designed for gray scale images and the compression ratio was still approximately 0.25 bpp, which is the same as that of the original BTC but less than those of BTC's variants.

Because there are already several existing variants of BTC such as AMBTC (Chen and Tai 1999) and GA-AMBTC (Tai et al. 1998) which aim at improving the compression ratio of BTC, in this paper, we propose a data hiding scheme for color images based on GA-AMBTC to expand the variety of cover media. To provide sufficient background knowledge of the embedding scheme, the details of BTC, AMBTC,

and GA-AMBTC are described in Sect. 2. In addition to GA-AMBTC, the concept of modification direction (Zhang and Wang 2006) is applied to enhance hiding capacity of the proposed scheme. The reason we chose GA-AMBTC is for its effectiveness in improving the compression ratio of BTC. The major reason modification direction is chosen is for its low distortion after pixel modifications. Detailed descriptions of the embedding and extracting phases appear in Sects. 3.1 and 3.2, respectively. Experimental results confirm that the proposed scheme can provide good image quality, hiding capacity and compression ratio.

The rest of this paper is organized as follows. In Sect. 2, the BTC, AMBTC and GA-AMBTC data compression schemes are described. The proposed data embedding scheme with high hiding capacity, good image quality and high compression ratio is presented in Sect. 3. Several experimental results are illustrated and discussed in Sect. 4. Finally, concluding remarks as well as some suggestions for future work appear in Sect. 5.

2 Related works

Block truncation coding (BTC) (Chuang and Chang 2006) is a technique of block-by-block image compression. To encode a block, BTC computes the average for the current block first and then uses this average to produce a bitmap for the current block. In the bitmap, the indicator 1 is notated when the pixel value of this block is larger than the block's average. Otherwise, the indicator is denoted as 0. Subsequently, BTC derives two quantization values, $\bar{x}_0$ and $\bar{x}_1$, by averaging the pixels whose indicators are 0 and 1 in the bitmap, respectively. Generally, for each block, the sender sends a bitmap and two quantization values to the receiver. Later, the receiver can decode a pixel with $\bar{x}_0$ when its corresponding indicator is denoted as 0. Otherwise, the pixel is reconstructed by $\bar{x}_1$.

As an example, a block of the image is shown in Fig. 1a. The average of this block $\bar{x}$ is 124 and the corresponding bitmap for this block is shown in Fig. 1b. The sender calculates two quantization values as $\bar{x}_0$ and $\bar{x}_1$ when the corresponding indicators of pixels are 0 and 1, respectively. In this example,

(a)

146	97	89	84
150	122	92	90
150	142	135	99
147	156	145	143

Original image block

(b)

1	0	0	0
1	0	0	0
1	1	1	0
1	1	1	1

Bitmap

(c)

146	96	96	96
146	96	96	96
146	146	146	96
146	146	146	146

Reconstructed image block

Fig. 1 An example of BTC encoding and decoding: **a** Original image block, **b** Bitmap, **c** Reconstructed image block

the value of $\bar{x}_0$ is 96 and $\bar{x}_1$ is 146. Then the sender sends the bitmap, $\bar{x}_0$ and $\bar{x}_1$, to the sender side. Later, the receiver can decode a block with $\bar{x}_0$ and $\bar{x}_1$, as shown in Fig. 1c, when the indicators in this bitmap are 0 and 1, respectively.

In BTC encoding, eight bits are required for both $\bar{x}_0$ and $\bar{x}_1$ and 16 bits are requires for a bitmap so that a total of 32 bits is required to represent an original image block. Therefore, the compression ratio of BTC for each block is $\frac{32}{16\times 8} = 0.25$ bpp. To further improve the compression ratio of BTC by reducing the bits of $\bar{x}_0$ and $\bar{x}_1$, an absolute moment block truncation coding (AMBTC) (Bender et al. 1996) was proposed. In AMBTC, the values of $\bar{x}_0$ and $\bar{x}_1$ can be composed by Eqs. (1) and (2), respectively.

$$\bar{x}_0 = \bar{x} - \frac{m\alpha}{2(m-q)}, \tag{1}$$

$$\bar{x}_1 = \bar{x} + \frac{m\alpha}{2q}. \tag{2}$$

Here, m is the amount of pixels in each block; q is the amount of the pixels when the indicator denoted as 1; $\bar{x}$ is the average of all pixels in the block; and the values of $\bar{x}$ and α are defined as below, respectively.

$$\bar{x} = \frac{\sum_{i=1}^{m} x_i}{m}, \tag{3}$$

$$\alpha = \frac{\sum_{i=1}^{m} |x_i - \bar{x}|}{m}, \tag{4}$$

where x_i is the pixel value located at position i in the block. In AMBTC, $\bar{x}$ and α only require 6 and 4 bits, respectively. A bitmap needs 16 bits. Therefore, if two quantization values $\bar{x}_0$ and $\bar{x}_1$ used in BTC are replaced with $\bar{x}$ and α, the compression ratio of AMBTC can be successfully reduced, as shown in the formula $\frac{26}{16\times 8} = 0.203$. With this result, the compression ratio of AMBTC is better than that of BTC.

If a color image is compressed by AMBTC, each pixel in a color image can be broken into R, G and B planes. Each plane needs two quantization values; therefore, three quantization pairs depicted as $\{\bar{R}x, R\alpha\}$, $\{\bar{G}x, G\alpha\}$ and $\{\bar{B}x, B\alpha\}$ are generated for the R, G and B planes, respectively. Here, $\bar{R}x$, $\bar{G}x$ and $\bar{B}x$ are averages of planes R, G and B, respectively. The three corresponding coefficients of $\bar{R}x$, $\bar{G}x$ and $\bar{B}x$ are $R\alpha$, $G\alpha$ and $B\alpha$, respectively. Later, $\{\bar{R}x_0, \bar{R}x_1\}$, $\{\bar{G}x_0, \bar{G}x_1\}$, and $\{\bar{B}x_0, \bar{B}x_1\}$ can be computed according to $\{\bar{R}x, R\alpha\}$, $\{\bar{G}x, G\alpha\}$ and $\{\bar{B}x, B\alpha\}$ by using Eqs. (1) and (2), respectively.

Because a pair of quantization values such as $\bar{R}x$ and $\bar{R}\alpha$ needs 10 bits and each bitmap for either R, G or B plane requires 16 bits, the required total number of bits for encoding a block of a color image is $78 (= 3\times 10 + 3\times 16)$ bits. The compression ratio can be computed as $\frac{30+48}{16\times 8\times 3} \cong 0.203$ bpp. If three bitmaps for different planes for a block can be further condensed into a common plane, the compression ratio can be effectively reduced to $\frac{30+16}{16\times 8\times 3} \cong 0.120$ bpp.

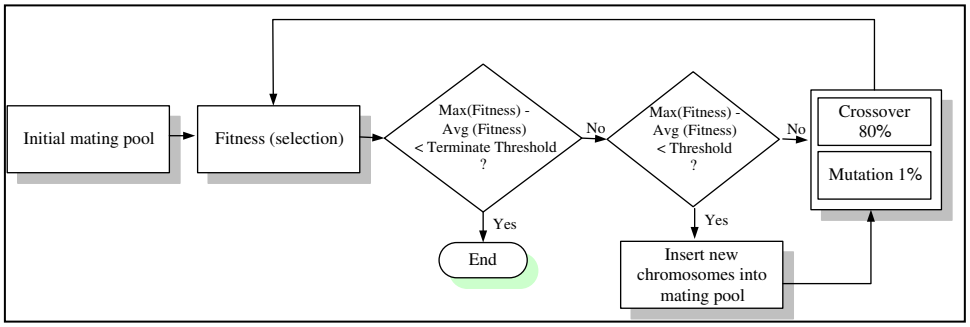

Fig. 2 The flowchart of GA-AMBTC

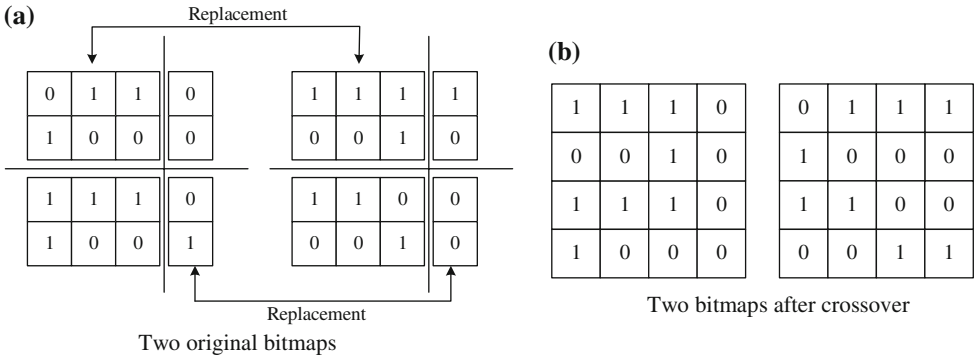

Fig. 3 Example of crossover: **a** Two original bitmaps, **b** Two bitmaps after crossover

To achieve this objective, an exhaustive search is usually adopted to try all possible values of a condensed bitmap and then to provide the least distortion. However, an exhaustive search is time-consuming. For example, if a block is sized as 4×4, then the total number of combinations of its bitmap is 2^{16}. In addition, the generation of a common bitmap for each block is independent. That means the exhaustive search has to be performed repeatedly until all blocks in an image are encoded. To speed up the generation of a condensed bitmap for each block, Tai et al. (1998) proposed an improved version called GA-AMBTC by using the genetic algorithm (GA). GA is a kind of best searching algorithm that simulates biological evolution to produce a similar optimal solution. GA is widely used in various fields such as pattern recognition, decision support and the nearest optimization problem.

In organism evolution, organisms with defective genes are weeded out so that a species of organisms preserves its beneficial genes for its descendants. Generally, better chromosomes will be produced for propagation after crossover or mutation. In GA-AMBTC, the three bitmaps R, G and B of a block can be reduced as a condensed bitmap to a block which can be seen as the best chromosome after selection in evolution. The flowchart of GA-AMBTC for a given block is shown in Fig. 2.

Initially, four copies of the R, $G$and B planes are put into a mating pool. Then two fitness functions f and P_i are defined to calculate the fitness values for better chromosome selection as shown in Eqs. (5) and (6). These fitness values can determine whether a chromosome is preserved or not.

$$f = \frac{1}{\mathrm{MSE}_B}, \tag{5}$$

$$P_i = \frac{f(G_i)_k}{\sum_{j=1}^{N} f(G_i)k}. \tag{6}$$

Here, MSE_B is the distortion computed as Eq. (7); k is the kth iteration; G_i is the preserved gene; and N is the amount of preserved chromosomes in mating pool. A larger error occurs while f is small but a larger P_i leads to a smaller error. Therefore, a large P_i should be preserved.

$$\mathrm{MSE}_B = \frac{\sum_{b_i=0} (\bar{x}_0 - x_j)^2 + \sum_{b_i=1} (\bar{x}_1 - x_j)^2}{m}, \tag{7}$$

where $B = \{b_i | b_i \in \{0, 1\}, i = 1, 2, \ldots, m\}$, m is the block size; the two quantization values are $\bar{x}_0$ and $\bar{x}_1$ while indicators in bitmap are 0 and 1, respectively; x_j is the pixel value located at position j of this block.

The average of all the f values and the maximum f value in the mating pool are depicted as Avg(f) and Max(f), respectively. Basically, GA-AMBTC will be terminated when the difference between Max(f) and Avg(f)is less than a predefined terminating threshold. Conversely, the contents of the bitmaps are similar when the difference between Max(f) and Avg(f)is larger than the terminating threshold but less than the pre-defined threshold $\mathrm{th}_{\mathrm{sim}}$. If this is the situation, GA-AMBTC generates new chromosomes and

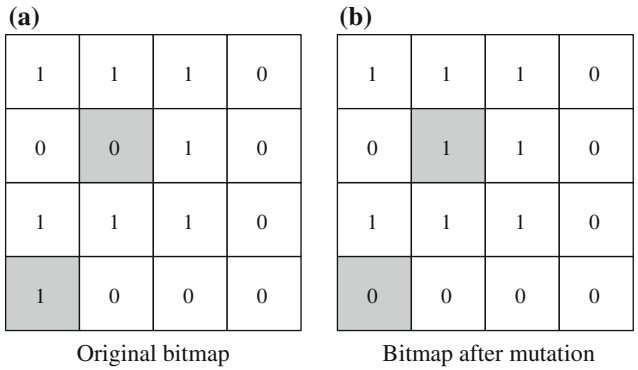

Fig. 4 Example of the bitmap mutation: **a** Original bitmap, **b** Bitmap after mutation

replaces 30% of the original chromosomes with new ones. Otherwise, the chromosomes must cross over and mutate within the mating pool.

During crossover, two bitmaps (chromosomes) as shown in Fig. 3a are used to produce another two bitmaps as shown in Fig. 3b. In Fig. 3a, 80% of the bits in the two bitmaps are replaced with bits from each other to perform the crossover. The production after crossover is shown in Fig. 3b.

During mutation, GA-AMBTC randomly selects 1% of the bits in the original bitmap and changes the original value of 1 with 0. Otherwise, GA-AMBTC changes them to 1. In the example in Fig. 4, two marked bits are changed; the original bitmap and the changed bitmap are shown in Fig. 4a, b, respectively. If the difference between Max(f) and Avg(f) is less than a pre-determined terminating threshold, the evolution procedure is converged; the optimal bitmap will be selected and outputted as the common bitmap for the R, G and B planes.

3 Proposed scheme

In this section, a simple data embedding and extracting scheme is presented based on GA-AMBTC to hide secret data in GA-AMBTC compressed color images. To protect secret bits, senders do not directly hide secret data *sc* in color images. The secret data *sc* must be transformed into temporary secret data *tc* first by using an *XOR* operator before data embedding. Subsequently, the encoder embeds the generated *tc* into a GA-AMBTC color compressed image. In the transformation, the encoder first uses a private key to produce a random bit stream *rc*. Next, the temporary secret data *tc* is generated by *sc XOR rc*. Note that the lengths of *rc* and *tc* are the same as *sc*. The private key can be shared between senders and receivers before transmitting other secret data over the Internet.

Later, after receivers extract the temporary secret data *tc* from compression codes by using the proposed extracting strategy, they can use the same private key to generate the random bit streams *rc*. Then the secret data *sc* can be restored with *tc XOR rc*. The proposed scheme consists of two phases: data embedding and extraction. For clarity, examples are demonstrated in Sect. 3.3 to give detailed explanations of the embedding and extracting procedures.

3.1 Embedding phase

On the sender side, a color image I serving as the cover media is first divided into several non-overlapping blocks. Then the encoder will scan all blocks of the color image I in a raster-scanning order. The diagram of the embedding procedure is shown in Fig. 5. The encoder repeats the embedding procedure until all blocks in a color image are processed.

As mentioned in the previous section, the compression results of a block of a color image using GA-AMBTC are the three quantization pairs $\{\bar{R}x, R\alpha\}$, $\{\bar{G}x, G\alpha\}$, and $\{\bar{B}x, B\alpha\}$ for the R, G and B planes, respectively, and a common bitmap. In the embedding procedure, the proposed scheme can derive the three block pairs $\{\bar{R}x_0, \bar{R}x_1\}$, $\{\bar{G}x_0, \bar{G}x_1\}$ and $\{\bar{B}x_0, \bar{B}x_1\}$ for the R, G, and B planes by using Eqs. (1) and (2). To hide temporary secret data *tc* in three block pairs and a common bitmap, the embedding procedure is divided into two phases (phase-1 and phase-2) with different embedding strategies. In phase-1, the temporary secret data *tc* are embedded into the common bitmap. In phase-2, the temporary secret data *tc* are embedded into three block pairs by exploiting their modification directions (Chang et al. 2003).

All details of each process are stated below to clearly describe the proposed methods. To further speed up the common bitmap generation of the R, $G$and B planes for each block, the traditional generation procedure of GA-AMBTC

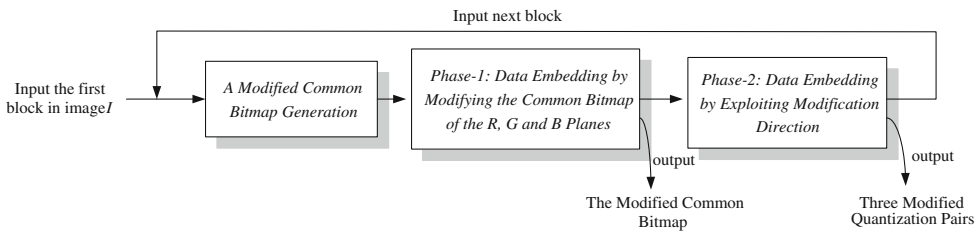

Fig. 5 The diagram of the proposed embedding procedure

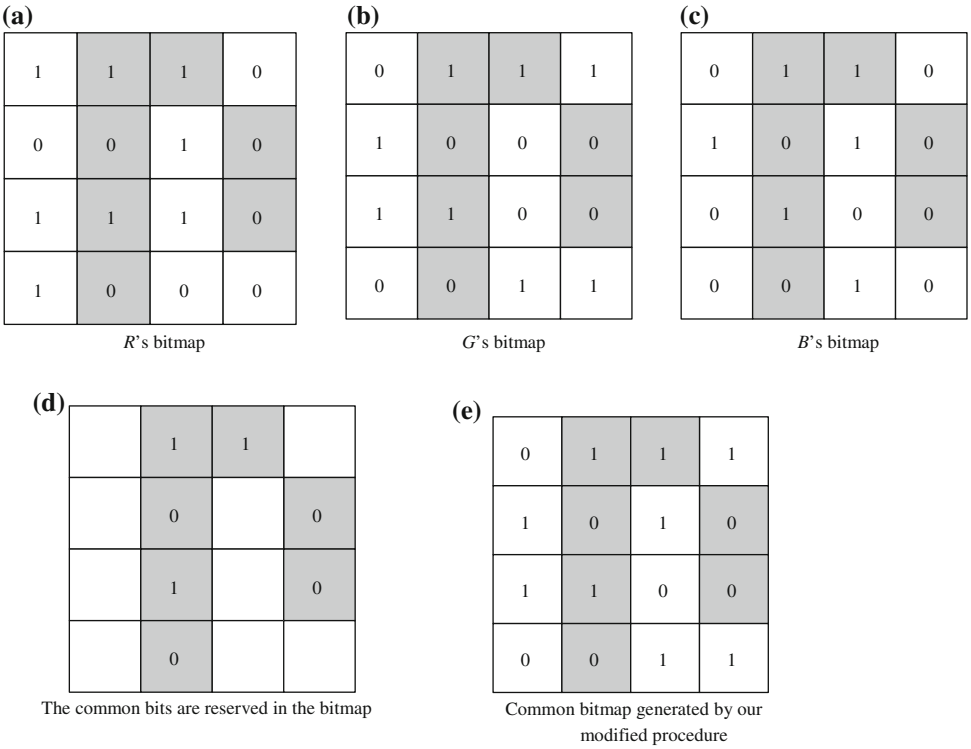

Fig. 6 Example of GA training to generate a common bitmap for a block, **a** *R*'s bitmap, **b** *G*'s bitmap, **c** *B*'s bitmap, **d** The common bits are reserved in the bitmap, **e** Common bitmap generated by our modified procedure

has been modified in our scheme. The modified common bitmap generation procedure is described in the following paragraph.

3.1.1 A modified common bitmap generation

For an inputted block, the encoder can first apply AMBTC to generate three bitmaps for the *R*, *G* and *B* planes. To enhance compression performance, these three bitmaps have to be condensed into a common bitmap. The best scenario for condensation is that the three bitmaps are exactly the same so that a common bitmap can be generated without any effort. Unfortunately, the best scenario does not occur frequently. Mostly, the three bitmaps are partially similar. Therefore, in traditional GA-AMBTC, all bits of these bitmaps must be processed through selection, crossover and mutation to produce a single bitmap. In other words, even if some bits of the three original bitmaps of the *R*, *G*, and *B* planes are the same, GA-AMBTC still expands effort in processing them to produce the common bitmap.

To speed up the generation of a common bitmap in GA-AMBTC, in our modified procedure, bits which are exactly the same in the three bitmaps are simply reserved in the common bitmap, and then GA is used to train the bits, which are different in the three bitmaps of the *R*, *G* and *B* planes. Examples of the bitmaps for the *R*, *G* and *B* planes are shown in Fig. 6a, b, c, respectively. The marked bits, whose values are the same, are located at the *R*, *G* and *B* planes. These bits are reserved as shown in Fig. 6d. Then the residual bits, which are in the white area, will be filled up by GA. The final common bitmap of the *R*, *G*and *B* planes for a block is shown in Fig. 6e.

3.1.2 Phase-1: data embedding by modifying the common bitmap of the R, G and B planes

In this phase, the encoder retrieves a temporary bit from *tc* to hide it in the common bitmap, which is generated by the modified common bitmap procedure for the current block. According to the temporary secret bit value, the encoder modifies some bits to satisfy the embedding strategy. The embedding strategy is straightforward. If the secret bit is 0, the encoder has to make sure that the number of 0bits in the common bitmap is more than the number of 1 bits by modifying some bits, which leads to less distortion. If the secret bit is 1, the number of 1 bits in the common bitmap must be more than the number of 0 bits.

Based on the proposed embedding principle, there are two cases in which the common bitmap does not need to modify any bits. The first case is when the embedded bit is 0 and the number of 0 bits in the common bitmap is more than the number of 1 bits. The second case is when the embedded bit is 1 and the number of 1 bits in the common bitmap is more than the number of 0 bits. For the remaining scenarios, the modification of the common bitmap is required according to the proposed embedding strategy. Usually, modification is

unavoidable. To maintain the image quality of a stego-image after modifying the common bitmap, the caused distortion measure of modification is measured in Eq. (8) and depicted as D_{amt}.

$$D_{\text{amt}} = \sum_{i=0}^{1} \left\{ (Rx_{\text{amt}} - \bar{R}x_i)^2 + (Gx_{\text{amt}} - \bar{G}x_i)^2 + (Bx_{\text{amt}} - \bar{B}x_i)^2 \right\}, \quad (8)$$

where $1 \leqq \text{amt} \leqq m$; m is the amount of pixels in a block. After measurement, the encoder sorts all D_{amt}'s of all possible modifications in an ascendant order. A smaller D_{amt} means that the caused distortion is small when the bitmap is modified. The modification which leads to the smallest distortion will be chosen by the encoder to modify the common bitmap and embed one secret bit. For example, a secret bit 0 is assumed to be embedded into the bitmap as shown in Fig. 6e. The number of 1 bits in the bitmap is 9, which is more than the number of 0 bits. In order to embed 0 into this block, two bits of 1 must be changed to 0 is in this bitmap.

3.1.3 Phase-2: data embedding by exploiting modification direction

The encoder retrieves two temporary secret bits from *tc* and then transforms these two bits into a secret decimal digit d. Next, the encoder uses a block pair such as $\{\bar{R}x_0, \bar{R}x_1\}$ to compute F value by using Eq. (9).

$$F(g_1, g_2) = \left[\sum_{j=1}^{2} g_j \times j \right] \bmod 5, \quad (9)$$

Here, g_1 and g_2 are the values of a block pair such as $\bar{R}x_0$ and $\bar{R}x_1$, respectively. No modification is needed if a secret digit d is equal to the calculated value F. If $d \neq F$, the encoder calculates $s = d - F \bmod 5$. If s is less than or equal to 2, the value of g_s is added by 1; otherwise, the value is g_{5-s} minus 1.

Because each block pair can hide two secret bits, three block pairs such as $\{\bar{R}x_0, \bar{R}x_1\}$, $\{\bar{G}x_0, \bar{G}x_1\}$ and $\{\bar{B}x_0, \bar{B}x_1\}$ for the R, G and B planes can hide six bits in total. In addition, by using Eqs. (3) and (4), the modified values of block pairs can generate three quantization pairs as $\{\bar{R}x, R\alpha\}$, $\{\bar{G}x, G\alpha\}$ and $\{\bar{B}x, B\alpha\}$ after data embedding. For example, by using Eqs. (3) and (4), unknown $\{\bar{R}x, R\alpha\}$ can be computed by two modified values of block pairs $\{\bar{R}x_0, \bar{R}x_1\}$. Subsequently, these three new quantization pairs can be transmitted to the receiver side for decoding later.

3.2 Extracting phase

In our extracting phase, receivers are allowed to apply extracting strategies to extract the secret data from the received data and decode the image at the same time. The extracting phase is broken into two phases, extracting phase-1 and extracting phase-2, to extract the temporary secret data that were embedded by embedding phase-1 and embedding phase-2, respectively. The details of each phase are described below.

3.2.1 Phase-1: extracting hidden data from the common bitmap of the R, G and B planes

In the phase-1 embedding strategy, senders embed the secret data into the common bitmap of the R, G, and B planes by justifying the amount of 0 bits and 1 bits in the common bitmap. In this phase, the hidden bit can be easily determined according to the amounts of 1 bits and 0 bits; that is, the hidden data is judged as 0 when the number of 0 bits is larger than that of the number of 1 bits. Otherwise, the hidden data is determined to be 1.

3.2.2 Phase-2: extracting hidden data from three quantization pairs

In phase-2 data extracting phase, hidden data can be extracted from three modified quantization pairs according to our extracting strategies. First, receivers use the three received quantization pairs $\{\bar{R}x, R\alpha\}$, $\{\bar{G}x, G\alpha\}$, and $\{\bar{B}x, B\alpha\}$ to reconstruct the three block pairs $\{\bar{R}x_0, \bar{R}x_1\}$, $\{\bar{G}x_0, \bar{G}x_1\}$and $\{\bar{B}x_0, \bar{B}x_1\}$ for each block by using Eqs. (1) and (2). Next, the hidden data is extracted from a block pair by using Eq. (9). Finally, the extracted secret data can be transformed into a serial of binary numbers.

By repeating the phase-1 and phase-2 extracting phases until all temporary secret data *tc* are retrieved and all blocks are decoded, the receiver can use the same private key as the sender to generate a serial number bit streams *rc*. Then, the real secret data *sc* can be correctly extracted by calculating *tc XOR rc*.

3.3 Examples of the phase-2 data embedding and phase-2 extracting phases

In our phase-1 data embedding phase, the sender can easily embed a secret bit into the common bitmap by justifying the amounts of 1 bits and 0 bits. Moreover, the receiver can simply judge the hidden bit by the amounts of 1 bits and 0 bits in the common bitmap in phase-1 data extracting phase. Therefore, in this section, we only present a simple example to demonstrate the phase-2 data embedding and phase -2 data extracting phases. It is assumed that the bitmap generated by our modified common bitmap procedure appears as shown in Fig. 7a and the values of the three block pairs $(\bar{R}x_0, \bar{R}x_1)$, $(\bar{G}x_0, \bar{G}x_1)$ and $(\bar{B}x_0, \bar{B}x_1)$ are 20, 50, 60, 80,

(a)

0	1	1	1
1	0	1	0
1	1	0	0
0	0	1	1

The common bitmap

(b)

	$\bar{R}x_0$	$\bar{R}x_1$	$\bar{G}x_0$	$\bar{G}x_1$	$\bar{B}x_0$	$\bar{B}x_1$
values	20	50	60	80	30	65

Three block pairs

(c)

Secret digits	$\bar{R}x_0$	$\bar{R}x_1$	$\bar{G}x_0$	$\bar{G}x_1$	$\bar{B}x_0$	$\bar{B}x_1$
0	20	50	60	80	30	65
1	21	50	61	80	31	65
2	20	51	60	81	30	66
3	20	49	60	79	30	64
4	19	50	59	80	29	65

The modified block pairs for all potential secret digits such as 0,1,2, 3 and 4

Fig. 7 Example of data embedding and extracting: **a** The common bitmap, **b** Three block pairs, **c** The modified block pairs for all potential secret digits such as 0, 1, 2, 3 and 4

30 and 65, respectively, as shown in Fig. 7b. In Fig. 7, it is assumed that a secret digit is embedded into the block pair $(\bar{R}x_0, \bar{R}x_1)$. In Eq. (9), $\bar{R}x_0$and $\bar{R}x_1$ can be treated as g_1 and g_2, respectively. According to the phase-2 data embedding strategy, the sender first computes the F value by using Eq. (9) and calculates $(20 \times 1 + 50 \times 2) \bmod 5 = 0$. If the secret digit is 0 and it is equal to the F value, the pair $(\bar{R}x_0, \bar{R}x_1)$ does not need any modification. If the secret digit is 2, in pair $(\bar{R}x_0, \bar{R}x_1)$, the $\bar{R}x_0$ does not need any change but the $\bar{R}x_1$ will be changed to 51 by computing $g_2 = g_2 + 1$ to hide the secret data 2. When the secret digit is 3, in $(\bar{R}x_0, \bar{R}x_1)$ pair, only $\bar{R}x_0$ needs to be modified to 49 by computing $g_{5-s} - 1 = g_{5-3} - 1 = g_2 - 1 = 49$.

For clarification, Fig. 7c lists all possible embedding results for these three block pairs using PHase-2 embedding strategies. It is assumed six secret bits have been transformed into three secret decimal digits 0, 3, 2, which are embedded into the R, G and B planes, respectively. Therefore, the values of $\bar{R}x_0$, $\bar{R}x_1$, $\bar{G}x_0$, $\bar{G}x_1$, $\bar{B}x_0$ and $\bar{B}x_1$ are changed to 20, 50, 60, 79, 30 and 66, respectively, as shown in Fig. 7c. Subsequently, by using Eqs. (1) and (2), the three corresponding quantization pairs $(\bar{R}x, R_\alpha)$, $(\bar{G}x, G_\alpha)$and $(\bar{B}x, B_\alpha)$ are generated.

In the phase-2 data extracting phase, the decoder can compute the three block-pairs $(\bar{R}x_0, \bar{R}x_1)$, $(\bar{G}x_0, \bar{G}x_1)$ and $(\bar{B}x_0, \bar{B}x_1)$ according to the values of $(\bar{R}x, R_\alpha)$, $(\bar{G}x, G_\alpha)$ and $(\bar{B}x, B_\alpha)$ by Eqs. (1) and (2). Subsequently, the decoder can simply retrieve the secret data by using Eq. (9). The extracted data will be the same as the secret data listed in Fig. 7c. Finally, the image is decoded according to the bitmap and three block-pairs $(\bar{R}x_0, \bar{R}x_1)$, $(\bar{G}x_0, \bar{G}x_1)$and $(\bar{B}x_0, \bar{B}x_1)$ as shown in Figs. 8a, b, c for the R, G and B planes, respectively.

(a)

20	50	50	50
50	20	50	20
50	50	20	20
20	20	50	50

R's plane

(b)

60	79	79	79
79	60	79	60
79	79	60	60
60	60	79	79

G's plane

(c)

30	66	66	66
66	30	66	30
66	66	30	30
30	30	66	66

B's plane

Fig. 8 The decoded results for the R, G and B planes, **a** R's plane, **b** G's plane, **c** B's plane

4 Experimental results

In the following experiments, six images entitled "Lena", "Zelda", "Baboon", "F16", "Boat" and "GoldHills" serve as test images, as shown in Fig. 9. These test images are 512×512 color images. For portability, the proposed scheme is implemented using Java. The simulation platform is Microsoft Windows XP, Pentium III with 1 GHz memory. Three performance matrices are used to measure the performance of the proposed hiding schemes: hiding capacity, image quality and compression ratio. The hiding capacity denotes the amount of secret bits embedded in the image. The compression ratio indicates the bit rates after data compression. The peak signal-to-noise ratio (PSNR) (Chang et al. 2006) defined in Eq. (10) is used to evaluate image quality of a stego-image generated by the proposed scheme.

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\text{MSE}} \text{dB}. \tag{10}$$

Here, 255 represents the maximum value of each pixel and the MSE (mean square error) for an image is defined in Eq. (11).

$$\text{MSE} = \left(\frac{1}{H \times W}\right) \sum_{i}^{H} \sum_{j}^{W} (x_{ij} - x'_{ij})^2. \tag{11}$$

Here, the notations H and W represent the height and width of an image, respectively; x_{ij} is the pixel value of the coordinate (x, y) in an original image, and x'_{ij} is the pixel value after embedding processing. Because all test images are color image and a color image is composed of three planes, the PSNRs of these three planes are averaged to obtain the PSNR of a color image in the following experiments.

The image quality derived by AMBTC is shown in Table 1 for later comparison with the proposed scheme after data embedding.

Generally, the higher the PSNR value of an image implies the better the image quality. Conversely, the worse the image quality of an image implies the lower its PSNR. In addition, a high hiding capacity generally leads to low image quality. However, the distortion of the stego-image caused by hidden data may not be sensitive to the human visual system when

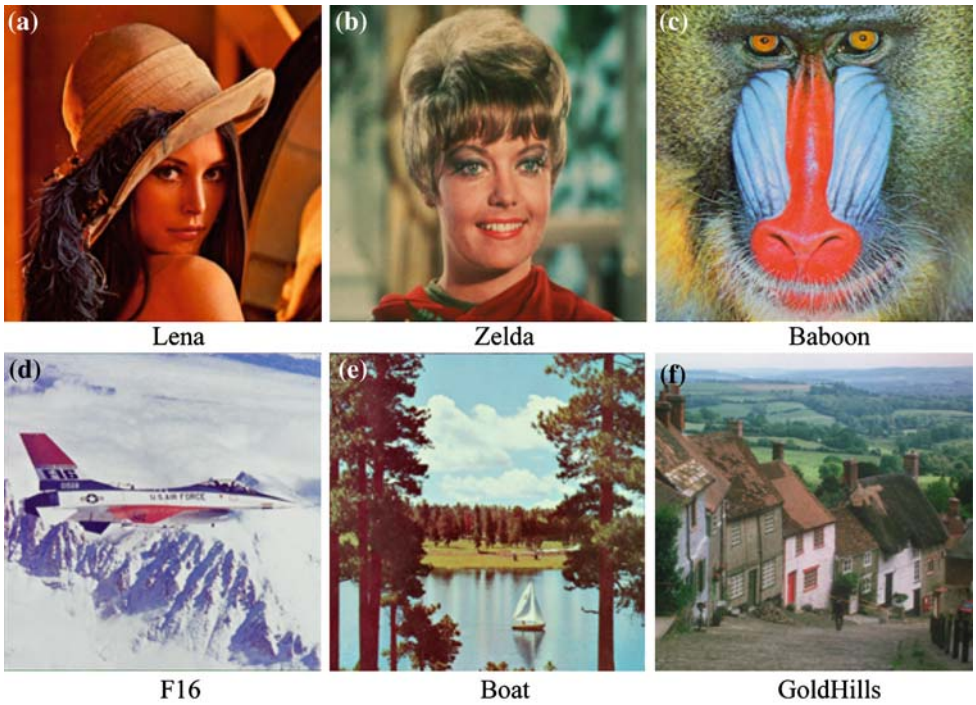

Fig. 9 Six test color images with size in 512 × 512, **a** Lena, **b** Zelda, **c** Baboon, **d** F16, **e** Boat, **f** GoldHills

Table 1 The image quality (PSNRs) of AMBTC encoding

Images	*R*'s plane	*G*'s plane	*B*'s plane	Avg PSNR
Lena	35.57	34.01	32.5	34.02
Zelda	30.98	31.05	30.99	31.00
Baboon	25.94	25.99	27.32	26.41
F16	34.78	31.03	32.23	32.68
Boat	28.57	27.95	32.02	29.51
GoldHill	28.13	28.74	28.70	28.52

the PSNR value of the stego-image is larger than or equal to 30 dB.

In the first experiment, the performance of the scheme is evaluated according to image quality, hiding capacity and compression ratio. The related experimental results are shown in Table 2.

From Table 2, it can be noted the compression ratio provided by the proposed scheme is the same as that of GA-AMBTC. The highest hiding capacity of the proposed scheme with a color cover image size of 512 × 512 is 112 kb. Although some PSNRs of the proposed scheme are less than 30 dB for "Zelda", "Baboon", "Boat" and "GoldHills," the average difference of PSNR between the ours and AMBTC is still maintained at 3 dB. Moreover, the visual qualities of the stego-images of "Lena", "Zelda", "Baboon", "F16", "Boat" and "GoldHills" shown in Fig. 10a–e are very similar to the original images, and the differences among them are not discernable to the human visual system.

To further evaluate the performance of the phase-2 embedding strategy on image quality, the second experiment compares PSNR values with and without performing phase-2 embedding. The related experimental results are listed in Table 3. From Table 3, it can be seen that the largest distortion caused by performing phase-2 data embedding is 0.07 dB, which is very slight. For some special cases, such as "Lena," the PSNR of the *R* plane is 0.01 higher than it would have been without performing phase-2 data embedding.

Table 2 The performance of the proposed scheme

Images	*R*'s plane	*G*'s plane	*B*'s plane	Avg PSNR	Hiding capacity (kb)	Compression ratio (bpp)
Lena	33.39	32.33	30.70	32.14	112	0.120
Zelda	27.98	29.09	28.00	28.35	112	0.120
Baboon	23.52	24.08	24.34	23.98	112	0.120
F16	31.71	28.77	29.64	30.04	112	0.120
Boat	24.56	25.47	27.81	25.95	112	0.120
GoldHills	25.31	26.40	25.29	25.66	112	0.120

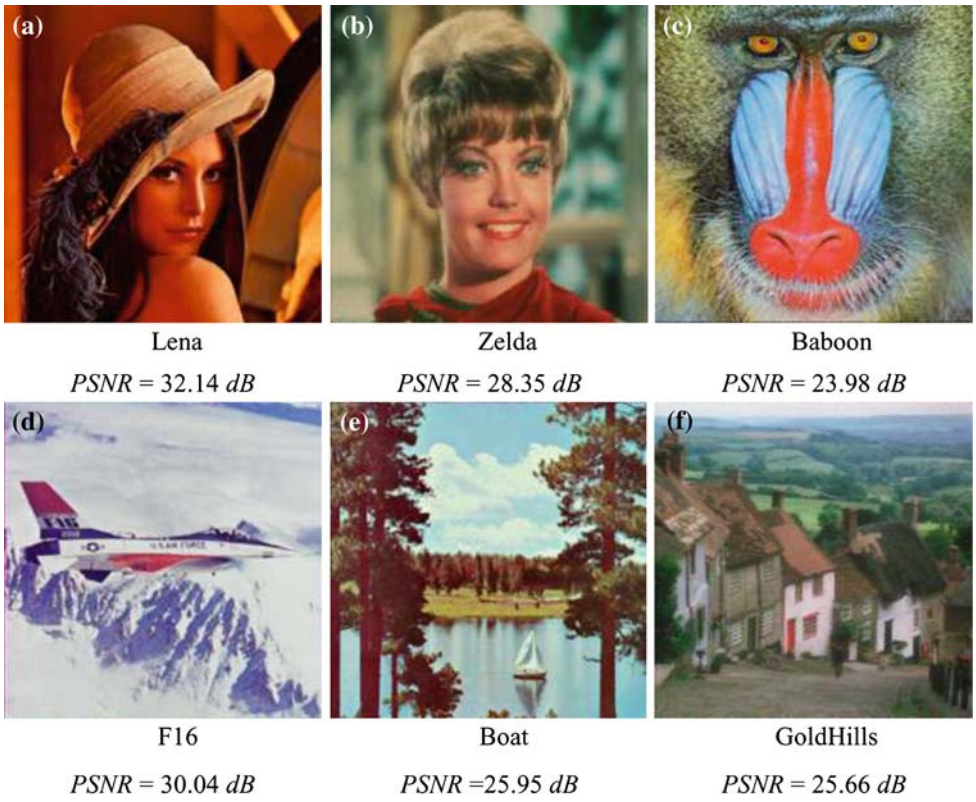

Fig. 10 Six stego-images, **a** Lena PSNR = 32.14 dB, **b** Zelda PSNR = 28.35 dB, **c** Baboon PSNR = 23.98 dB, **d** F16 PSNR = 30.04 dB, **e** Boat PSNR = 25.95 dB, **f** GoldHills PSNR = 25.66 dB

Table 3 PSNR comparisons among setgo-images with and without performing phase-2 embedding

Images	After phase-2 data embedding			Without phase-2 data embedding		
	R's plane	*G*'s plane	*B*'s plane	*R*'s plane	*G*'s plane	*B*'s plane
Lena	33.39	32.33	30.70	33.40	32.32	30.66
Zelda	27.98	29.09	28.00	27.97	29.09	28.01
Baboon	23.52	24.08	24.34	23.52	24.05	24.32
F16	31.71	28.77	29.64	31.66	28.70	29.65
Boat	24.56	25.47	27.81	24.51	25.45	27.72
GoldHills	25.31	26.40	25.29	25.66	26.37	25.28

The third experiment explores how much computation time can be saved by using the modified common bitmap generation procedure. The efficiency ratio can be calculated as following formula.

$$\text{Efficiency ratio} = (\text{reserved bits}) / (\text{the image size}). \quad (12)$$

The results indicate more efficiency with more reserved bits. Table 4 lists the reserved bits, processing bits and the efficiency ratio, respectively. As can be seen, for the best scenario such as "Lena," there are 183,459 bits of *R*, *G* and *B* planes are the same. If these bits are skipped and the residual bits are processed, up to 69% of the computation time can be saved. Therefore, more time is saved with more reserved bits.

Table 4 The efficiency of our modified common bitmap generation

Images	Reserved bits	Processing bits	Efficiency ratio
Lena	183459	78685	0.699
Zelda	176586	85558	0.674
Baboon	151281	110863	0.577
F16	165616	96528	0.631
Boat	132723	129421	0.506
GoldHills	170867	91277	0.651

5 Conclusions

In this paper, a scheme is proposed based on GA-AMBTC and modification direction to design a simple data embedding

scheme for color images. The sender hides the secret data into GA-AMBTC compressed result and transmits the modified compressed result to the receiver side. Receivers can later directly extract secret data and decode images by using simple extracting procedures. In general, the proposed scheme has three advantages. First, even when secret data are embedded into the GA-AMBTC's compressed result, the modified compressed result is the same size as it would have been without modification. Therefore, the modified compressed result does not appear suspicious to potential attackers. Second, by performing a GA-AMBTC decoding procedure, the modified compressed result can still be used to reconstruct an image so that attackers cannot determine it contains secret data. Third, the data extraction procedure is efficient because the hidden data can be extracted directly from the modified compressed result without prior decompression. As an additional benefit, the experimental results illustrate that the proposed scheme maintains a satisfying image quality at a high embedding capacity using a high compression ratio.

References

Bender W, Gruhl D, Morimoto N, Lu A (1996) Techniques for data hiding. IBM Syst J 35(3&4):313–336

Chan CK, Cheng LM (2004) Hiding data in images by simple LSB substitution. Pattern Recognit 37(3):469–474

Chang CC, Hsiao JY, Chan CS (2003) Finding optimal least-significant-bit substitution in image hiding by dynamic programming strategy. Pattern Recognit 36(7):1583–1593

Chang CC, Hu YC (1999) Hybrid image compression methods based on vector quantization and block truncation coding. Opt Eng 38(4):591–598

Chang CC, Lin CY, Wang YZ (2006) New image steganographic methods using run-length approach. Inform Sci 176(22):3393–3408

Chang CC, Tai WL, Lin CC (2006) A reversible data hiding scheme based on side match vector quantization. IEEE Trans Circuits Syst Video Technol 16(10):1301–1308

Chen B, Latifi S, Kanai J (1999) Edge enhancement of remote image data in the DCT domain. Image Vis Comput 17(12):913–921

Chen WJ, Tai SC (1999) A genetic algorithm approach to multilevel absolute moment block truncation coding. IEICE Trans Fundam Electron Commun Comput Sci E82-A(8):1456–1462

Chuang JC, Chang CC (2006) Using a simple and fast image compression algorithm to hide secret information. Int J Comput Appl 28:1735–1743

Cox IJ, Kilian J, Leigiton T, Shamoon T (1997) Secure spread spectrum watermarking for multimedia. IEEE Trans Image Process 6(12):1673–1687

Du WC, Hsu WJ (2003) Adaptive data hiding based on VQ compressed images. IEE Proc Vis Image Signal Process 150(4):233–238

Hsu CT, Wu JL (1999) Hidden digital watermarks in images. IEEE Trans Image Process 8(1):58–68

Hu YC, Chang CC (1999) Quadtree-segmented image coding schemes using vector quantization and block truncation coding. Opt Eng 39(2):464–471

In J, Hsiarani S, Kossentini F (1999) On RD optimized progressive image coding using JPEG. IEEE Trans Image Process 8(11):1630–1638

Kamstra LH, Henk JAM (2005) Reversible data embedding into images using Wavelet t-techniques and sorting. IEEE Trans Image Process 14(12):2082–2090

Kim T (1992) Side match and overlap match vector quantizers for images. IEEE Trans Image Process 1(2):170–185

Konstantinides K, Bhaskaran V, Beretta G (1999) Image sharpening in the JPEG domain. IEEE Trans Image Process 8(6):874–878

Lee YK, Chen LH (2000) High capacity image steganographic model. IEE Proc Vis Image Signal Process 147(3):288–294

Lin SD, Shie SC (2000) Side-match finite-state vector quantization with adaptive block classification for image compression. IEICE Trans Inform Syst E83-D(8):1671–1678

Lin YC, Tai SC (1998) A fast linde-buzo-gray algorithm in image vector quantization. IEEE Trans Circuits Syst II: Analog Digit Signal Process 45(3):432–435

Lu CS, Huang SK, Sze CJ, Liao HYM (2000) A new watermarking technique for multimedia protection. In: Ling G, Jan L, Kung SY (eds) Multimedia image and video processing, Chap. 18. CRC Press, Boca Raton, pp 507–530

Lu ZM, Sun SH (2000) Digital image watermarking technique based on vector quantization. Electron Lett 36(4):303–305

Lin YC, Wang CC (1999) Digital images watermarking by vector quantization. In: Proceedings of 9th National Computer Symposium, Taichung, Taiwan, pp 76–87

Munteanu A, Cornelis J, Auwera GVD, Cristea P (1999) Wavelet image compression—the quadtree coding approach. IEEE Trans Technol Biomed 3(3):176–185

Nasrabadi NM, King RA (1988) Image coding using vector quantization: a review. IEEE Trans Commun 36(8):957–971

Pai YT, Ruan SJ (2006) Low power block-based watermarking algorithm. IEICE Trans Inform Syst E89-D(4):1507–1514

Pan JS, Huang HC, Jain LC (2004) Intelligent watermarking techniques. World Scientific Publishing Company, Singapore

Pfitzman B (1996) Information hiding terminology, Information Hiding: First International Workshop. Springer Lecture Notes in Computer Science, Cambridge, UK, vol 1174, pp 347–350

Rivest R, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21(2):120–126

Shie SC, Lin SD, Fang CM (2006) Adaptive data hiding based on SMVQ prediction. IEICE Trans Inform Syst E89-D(1):358–362

Shieh CS, Huang HC, Wang FH, Pan JS (2004) Genetic watermarking based on transform domain techniques. Pattern Recognit 37(3):555–565

Tai SC, Chen WJ, Cheng PJ (1998) Genetic algorithm for single bitmap AMBTC coding of color images. Opt Eng 37(9):2483–2490

Walker JS (1996) Fast Fourier transforms, 2nd edn. CRC Press, Boca Raton, FL

Wang RZ, Lin CF, Lin JC (2001) Image hiding by optimal LSB substitution and genetic algorithm. Pattern Recognit 34(3):671–683

Zhang XP, Wang SZ (2006) Efficient steganographic embedding by exploiting modification direction. IEEE Commun Lett 10(11):1–3