Reversible Steganography Based on Side Match and Hit Pattern for VQ-Compressed Images

Cheng-Hsing Yang  
Department of Computer Science  
National Pingtung University of Education  
Pingtung, Taiwan 900  
chyang@mail.npue.edu.tw

Cheng-Ta Huang  
Department of Computer Science  
National Pingtung University of Education  
Pingtung, Taiwan 900  
bm096118@mail.npue.edu.tw

Shiuh-Jeng Wang  
Department of Information Management  
Central Police University  
Taoyuan, Taiwan 333  
sjwang@mail.cpu.edu.tw

Abstract—Data hiding is a technique which embeds secret data into cover media. It is important to the multimedia security and has been widely studied. Recently, some researchers paid attention to reversible data hiding methods. These methods can reconstruct the original image from the stego-image when embedded data are extracted. In this paper, we propose a new reversible steganographic method for vector quantization (VQ) compressed images. The new method uses the hit pattern to achieve reversibility, and the hit pattern strategy successfully reduces the overhead. The codebook is partitioned by the extension method which makes the stego-image have good visual features. Also, a look-up table is used to speed up the partitioning operation. Compared to Chang et al.’s method, the experimental results show that the proposed method has higher capacity, better visual quality, and lower running time.

Keywords—Data hiding, Reversible data hiding, Steganography, Vector quantization, Side match vector quantization.

I. INTRODUCTION

Data hiding is one of the most important technologies for the multimedia security. For digital images, a lot of data hiding techniques have been proposed for different purposes, such as secret communication, copyright protection, and tampering detection [1]. These techniques embed secret messages into original images and create stego-images. Digital images are usually stored in compressed formats, such as JPEG, vector quantization (VQ), and side match vector quantization (SMVQ). Some hiding schemes based on VQ or SMVQ have been proposed [2].

Reversible data hiding techniques can recover the original image after the embedded message is extracted [3, 4]. In these developed reversible hiding methods for original images, the difference expansion [3] and the histogram [4] are two mainly technologies. On the other hand, some reversible hiding methods for VQ-compressed images have been proposed [5-12]. In those developed reversible hiding approaches based on VQ or SMVQ, they can be classified into three categories according to their outputs: images as outputs [5, 6], legitimate VQ codes or SMVQ codes as outputs [7, 8], and VQ codes or SMVQ codes with additional control messages as outputs [9-12].

In this paper, we propose a new reversible hiding scheme based on the SMVQ concept for VQ-compressed images. Both input and output of our embedding method are legitimate VQ compressed codes. In our method, a hit pattern is used to deal with conflict cases. Our approach not only reduces the overhead of the hit pattern, but also increases the embedding capacity. The remainder of this paper is organized as follows. In Section II, Chang et al.’s method is introduced. Our proposed method is presented in Section III, and the experimental results are shown in Section IV. Finally, we draw our conclusions in Section V.

II. LITERATURE REVIEW

In this section, Chang et al.’s reversible embedding method is introduced [7]. The input of Chang et al.’s method is a VQ-compressed image (i.e., a VQ index table). For convenience of description, referring to a block as “X” means that the block is reconstructed from the corresponding VQ index. In their method, blocks of the first column and the first row are unprocessed. Then, from left to right and from top to down, each block X is processed by the side-match concept. For each block X, three state codebooks G0, G1, and G2 with the same size are created. Fig. 1 shows an example of constructing G0, G1, and G2, where the size of G0, G1, and G2 is 4 and the size of the super codebook is 16. In the initial stage, the codewords of the super codebook are clustered as shown in Fig. 1. G0 is constructed by the side-match prediction which uses X’s left block L and upper block U to predict and select codewords from the super codebook. Then, each codeword cw0 of G0 successively finds its closest codeword in the same cluster as the corresponding codeword of G1. If a codeword of G0 cannot find its closest codeword, the corresponding position of G1 is set to null. The finished corresponding state codebook G1 should satisfy G1 ∩ G0 = Ø and ∀ i: j cwj ∈ G1, cwj ≠ cw0. For example, as shown in Fig. 1, codeword cw0 of G0 finds the closest codeword cw0 as the corresponding codeword of G1. Codeword cw0 of G0 cannot find the closest codeword from its cluster. Then, the corresponding position of G1 is set to null.
After \( G_0 \) and \( G_1 \) are constructed, they can be used to perform the embedding process. If \( X \) is equal to the \( i \)th codeword of \( G_0 \) and the \( i \)th codeword of \( G_1 \) is available, one secret bit is embedded in \( X \). If the secret bit is 0, \( X \) is unchanged; otherwise, \( X \) is replaced with the \( i \)th codeword of \( G_1 \). If the \( i \)th codeword of \( G_1 \) is null, \( X \) cannot embed any secret bit. However, if \( X \) is in \( G_1 \), \( X \) must be replaced with the corresponding codeword of \( G_2 \). In order to recover \( X \) correctly, their method creates state codebook \( G_2 \) for \( X \) and guarantees that \( X \) is not in \( G_2 \). A hit map records which codeword is possible or is not possible to be equal to \( X \). For example, as shown in Fig. 1, codeword \( cw_1 \) of \( G_1 \) finds its closest codeword \( cw_3 \) from the codewords with hit rates of zero. If a position of \( G_1 \) is null, the corresponding position of \( G_2 \) is set to null. The third state codebook \( G_2 \) should also satisfy the following conditions: \( G_0 \cap G_1 \cap G_2 = \emptyset \) and \( \forall i, j \), \( cw_i \neq cw_j \). The hit map is created by training each block \( X \). The sorted codebook consists of the codewords sorted by the side-match prediction results. In order to enlarge the number of positions with hit value 0, 12 hit maps are proposed and separated according to the smoothness in the neighborhood of each block \( X \).

### III. OUR PROPOSED METHOD

In this section, we propose a new reversible hiding scheme. An ingenious hit pattern, which is small and is easy to be embedded together during the embedding process, is proposed.

#### A. Reversible hiding scheme

For each block \( X \), firstly, codebook \( C \) is partitioned into \( m \) state codebooks, \( S_0, S_1, ..., S_{m-1} \) with the same size. \( S_0 \) is generated based on the SMVQ method, which uses \( X \)'s left block \( L \) and upper block \( U \) to predict and select codewords from codebook \( C \). \( S_i \) is generated by using codewords in \( S_{i-1} \) to find their closest codewords from remained codewords in \( C \), where \( i = 1, 2, ..., m-1 \). This extension method can reduce the distance between each pair of corresponding codewords in \( S_{i-1} \) and \( S_i \). Now, \( X \) will fall into one of the three following cases.

Case (a): If block \( X \) is equal to the \( i \)th codeword of \( S_0 \), the embedding process is invoked. If the to-be-embedded bit is 0, \( X \) remains unchanged; on the other hand, if the to-be-embedded bit is 1, \( X \) is transformed into the \( i \)th codeword of \( S_1 \).

Case (b): If block \( X \) is equal to the \( i \)th codeword of state codebook \( S_j \) for \( j = 1, 2, ..., m-2 \), \( X \) is transformed into the \( i \)th codeword of \( S_{j+1} \). If the original \( X \) is in \( S_{m-2} \), a hit bit 0 is outputted.

Case (c): If block \( X \) is in state codebook \( S_{m-1} \), a hit bit 1 is outputted and \( X \) remains unchanged.

#### B. Embedding algorithm

The algorithm of our reversible embedding method is presented below. Both input and output of our algorithm are index tables, which are legitimate VQ compressed codes.

**Algorithm Embedding**

**Input:** Index table \( I \), codebook \( C \), secret \( W \)

**Output:** Embedded index table \( I' \), hit pattern \( H \)

1. Indexes belonging to the first row or the first column of \( I \) do not participate in the embedding process. For each index \( I_i \) in the first row or the first column, set \( I'_i = I_i \).
2. From left to right and from top to down, process each remaining index \( I \) by executing Step 3 ~ Step 6.
3. For index \( I \), construct state codebooks \( S_0, S_1, ..., S_{m-1} \) from codebook \( C \).
4. If \( I \in S_0 \), one bit is embedded. If the to-be-embedded bit in \( W \) is 0, set \( I'_i = I_i \). If the to-be-embedded bit in \( W \) is 1, set \( I'_i \) to the corresponding index in \( S_1 \).
5. If \( I \in S_j \) and \( j = 1, 2, ..., m-2 \), no embedding. Set index \( I'_i \) to the corresponding index in \( S_{j+1} \). If \( I \in S_{m-2} \), output a hit bit 0 to hit pattern \( H \).
6. If \( I \in S_{m-1} \), no embedding. Set \( I'_i = I_i \), and output a hit bit 1 to hit pattern \( H \).

Figure 2 shows an example to describe the embedding process, where 32 state codebooks are used. Indexes located in the first row or the first column of the index table remain unchanged. Then, the other indexes are performed by executing Step 3 to Step 6 from left to right and from top to down. Each index needs to construct its own state codebooks \( S_0, S_1, ..., S_{m-1} \). However, in Fig. 2, state codebooks are fixed for the purpose of ease explanation. The first index 13 belongs to \( S_2 \). Therefore, Step 5 is executed. There is no embedding and \( I'_i \) is set to the corresponding index 15 in \( S_3 \). Next, index 44 is processed and it belongs to \( S_6 \). Therefore, Step 4 is executed. Because the to-be-embedded secret bit is 0, \( I'_i \) is set to 44. The third index 58 belongs to \( S_0 \) and the to-be-embedded secret bit is 1. Therefore, \( I'_i \) is set to the corresponding index 69 in \( S_1 \). The fourth index 17 is in \( S_{17} \). Therefore, Step 6 is executed. \( I'_i \) is set to 17 and a hit bit 1 is outputted. The fifth index 73 is in \( S_{30} \). Therefore, Step 5 is
executed. \( I'_i \) is set to the corresponding index 85 in \( S_{31} \) and a hit bit 0 is outputted.

![Index table](image1)

![Embedded index table](image2)

**Figure 2.** An example of our embedding method.

**C. Improving run time**

In our approach, each block \( X \) needs to construct its own state codebooks \( S_0, S_1, \ldots, S_{m-1} \). This process consumes a lot of time. In order to reduce the running time of constructing state codebooks dramatically, a lookup table keeping the distance between each pair of codewords is pre-constructed. Moreover, the distance values of the row for each codeword are pre-sorted.

**IV. PREPARE YOUR PAPER BEFORE STYLING**

In this section, some experiments were performed on an Intel Core 2 T5500 1.66 GHz NB with 2 GB RAM and the Microsoft XP operating system. Our methods are implemented by C language. In the experiments, cover images are 512×512 gray images. Figure 3 shows the VQ cover images Lena, Sailboat, Baboon, and Peppers. The secret is a 64×64 binary image as shown in Fig. 4. The codebook has size 512 and is created by the Linde-Buzo-Gray (LBG) algorithm, and the size of a state codebook is 16.

![VQ cover images](image3)

**Figure 3.** Four VQ cover images. (a) Lena with a PSNR of 32.24 dB. (b) Sailboat with a PSNR of 29.25 dB. (c) Baboon with a PSNR of 24.70 dB. (d) Peppers with a PSNR of 31.40 dB.

**Table 1 shows the results of our method and Chang et al.’s method [7]. Our payloads are larger than Chang et al.’s payloads. On average, it increases about 56.25%. Our hit pattern size is much smaller than their hit map size. Chang et al.’s multiple hit maps need 512×12 = 6144 bits, where 512 is the codebook size and 12 is the number of hit maps. On average, our hit pattern size is about 1.5% of theirs. Moreover, the hit patterns of all case shown in Table 1 can be successfully embedded during their embedding processes. Although the PSNR values of our method are smaller than that of their method, we show that the visual quality of our embedded results is better than that of their results later.

**Table 2 shows the running time of our methods with and without pre-constructing the lookup table and Chang et al.’s method. The time of pre-constructing the lookup table is about two seconds, which is included into the time shown in Table 2. With the lookup table, our running time is dramatically reduced and is faster than Chang et al.’s running time.

Figure 5 shows stego-images of Chang et al.’s and our methods. In Chang et al.’s stego-image, it is easy to find many spots. For example, their stego-image Lena’s forehead has obvious white spots and cheek next to the hair has obvious black spots. This is because that their \( G_2 \) is constructed using the codewords with the hit bit 0. The distance between the corresponding codewords in \( G_1 \) and \( G_2 \) will be very far. But in our method, codewords in a state codebook find out their closest codewords to construct the next state codebook. So, in visual, our stego-image is better than Chang et al.’s stego-image in spite of the fact that our PSNR value is lower than theirs.

**V. CONCLUSIONS**

In this paper, a novel reversible data hiding scheme based on SMVQ prediction is proposed to embed secret data into a VQ compressed image. Our proposed method uses an extension method to generate state codebooks. This method makes the corresponding codewords in the current state codebook and the next state codebook close. Therefore, it avoids generating spots in stego-images. Compared to Chang et al.’s method, our proposed method has larger capacity and more efficient running time.
reduces the total size of hit maps. Moreover, in visual, our stego-images are better than Chang et al.’s stego-images.

ACKNOWLEDGMENT

This research was partially supported by the National Science Council of the Republic of China under the Grants NCS 97-2221-E-153-001.

REFERENCES


Figure 5. Stego-image comparisons: (a) Chang et al.’s method (PSNR = 30.23) (b) our proposed (PSNR = 27.49).