Lossless Data Embedding with File Size Preservation

Jessica Fridrich*, Miroslav Goljan, Qing Chen, and Vivek Pathak
Department of Electrical and Computer Engineering
SUNY Binghamton, Binghamton, NY 13902-6000, USA

ABSTRACT

In lossless watermarking, it is possible to completely remove the embedding distortion from the watermarked image and recover an exact copy of the original unwatermarked image. Lossless watermarks found applications in fragile authentication, integrity protection, and metadata embedding. It is especially important for medical and military images. Frequently, lossless embedding disproportionably increases the file size for image formats that contain lossless compression (RLE BMP, GIF, JPEG, PNG, etc…). This partially negates the advantage of embedding information as opposed to appending it. In this paper, we introduce lossless watermarking techniques that preserve the file size. The formats addressed are RLE encoded bitmaps and sequentially encoded JPEG images. The lossless embedding for the RLE BMP format is designed in such a manner to guarantee that the message extraction and original image reconstruction is insensitive to different RLE encoders, image palette reshuffling, as well as to removing or adding duplicate palette colors. The performance of both methods is demonstrated on test images by showing the capacity, distortion, and embedding rate. The proposed methods are the first examples of lossless embedding methods that preserve the file size for image formats that use lossless compression.

Keywords: Embedding, lossless, erasable, invertible, removable, distortion, file size preservation, RLE, JPEG

1. INTRODUCTION

Lossless embedding is a term for a class of data hiding techniques that are capable of restoring the embedded image to its original state without accessing any side information. One can say that the embedding distortion can be erased or removed from the embedded image. This is why some researchers refer to this type of embedding as erasable, removable, invertible, or distortion-free.

The idea of lossless embedding was for the first time proposed by Honsinger[1] in 1999. This technique, originally designed for lossless authentication, suffered from visible distortion (for some images) and limited capacity. Fridrich et al.[2] introduced a general methodology for lossless embedding in digital images that is based on lossless compression of image features. In this method, one first selects a subset X of image features that is losslessly compressible and that can be randomized without causing visible degradation to the image. The lossless embedding proceeds by compressing X to $C(X)$ and replacing X with $C(X)$ & $\{m_j\}_{j=1}^{L}$, where m_j are the message bits and '&' denotes concatenation. This way, one can losslessly embed up to $|X|-|C(X)|$ bits. This embedding paradigm is very general and many schemes can be designed by selecting different image features[2,3].

Alternative approaches to lossless embedding were later proposed by Macq[4], Tian[5], and Kalker[6]. Researchers have focused on different aspects of lossless embedding schemes. Increasing the lossless embedding capacity has recently been the principle motivation[3,5,6]. Celik[3] described a lossless authentication method with localization. Kalker et al.[7] proposed an approach that minimizes distortion per embedded bit. The first lossless embedding scheme for audio signals has been described by Kalker[6].

* fridrich@binghamton.edu; phone: 1 607 777-2577; fax: 1607 777-4464; http://www.ws.binghamton.edu/fridrich; SUNY Binghamton; Watson School of Engineering, Dept. of Electrical and Computer Engineering, Binghamton, NY USA 13902-6000

So far, little attention has been paid to the increase of the file size introduced by lossless embedding. In lossless embedding schemes designed for image formats that use some form of lossless compression, the increase in the file size could be many times larger than the actual number of embedded bits L. This inefficiency partially outweighs the advantage of embedding the data as opposed to appending it to the cover image. In fact, the sponsors of this research* have expressed the need for lossless embedding schemes that preserve the file size.

The act of lossless embedding of a random message stream increases the entropy $E(I)$ of the cover image I to $E(Y)=E(I)+L$, where Y is the embedded image. Fortunately, any specific lossless compression algorithm does not compress Y to the ideal $E(Y)$ bits but to $|C(Y)|$ bits, where $|C(Y)|>E(Y)$ and $C(Y)$ is the compressed embedded image. Consequently, one can theoretically at most $|C(Y)|-E(Y)$ bits losslessly and still preserve the file size+. To design such a scheme, however, one will likely have to tailor it to the specific compression scheme as well as the image format.

For example, if the cover image is an RGB encoded BMP file, the embedding does not increase its file size because the RGB BMP format does not incorporate any compression. However, the run length encoded (RLE) BMP, GIF, and JPEG contain lossless compression[8] (runlength, LZ77, and Huffman, respectively). Thus, the embedded file has a different, usually larger, size than the original.

This paper is the first step to developing lossless embedding techniques that preserve the file size. We have chosen two of the most common formats – the RLE encoded BMP image format and the ubiquitous JPEG format. In the next section, we describe the RLE compression algorithm and then, in Section 3, the RS lossless embedding scheme with file size preservation is introduced. Experimental results are presented in Section 4. In Section 5, we describe the relevant details of the JPEG format and the lossless file-size preserving technique. The algorithm performance is discussed in Section 6. Conclusions and future research are included in Section 7.

2. RLE COMPRESSION

Run length encoding (RLE) is a simple lossless compression that assigns short codes to long runs of identical symbols. It is used in the BMP format for images with up to 256 colors. The RLE format decoding rules are simple:

n B	decode as byte B repeated n-times, $n\geq 1$,
0 0	EOL; end of row,
0 1	EOB; end of bitmap,
0 2 x y	Delta; move x pixels to the right and y pixels down,
0 n $A_1 \ldots A_n$ (0)	$n\geq 3$, decode as $A_1 \ldots A_n$, zero is padded when n is odd.

The last decoding rule is called the "absolute mode". An important observation is that although the decoded image is always unique, the encoding can be done in many different ways. For example, some RLE implementations never use the code "nB" for n=1 but use the absolute mode instead. Therefore, different RLE encoders may generate files with slightly different sizes.

3. LOSSLESS EMBEDDING WITH FILE SIZE PRESERVATION FOR RLE BMPs

3.1 Problem statement

Because there exist many different RLE encoders, the embedding scheme must also guarantee that the message and the original image can be extracted from the embedded and encoded image independently of the RLE encoder.

---

* The Air Force Office of Scientific Research and the Air Force Research Laboratory in Rome, NY.

+ In practice, however, we are not likely to achieve this capacity because the embedded image must be perceptually equivalent to the original image.

We have decided to use the RS lossless data embedding method[2] as our starting point for the design of a lossless file-size preserving method. This method seemed to be the most amenable to modifications that would enable us such construction.

Lossless embedding with file size preservation for RLE compressed images (LE4RLE) should satisfy the following requirements:

(R1) The file size of the original and the embedded images must be equal after RLE compression using virtually any RLE compressor.
(R2) The original image can be retrieved from the embedded image exactly.
(R3) Any image processing that does not modify image content (image renaming, palette reordering, removing or introducing duplicate entries in the palette, image lossless compression and/or decompression of any kind) must not lead to message extraction failure.
(R4) The message and the original image can be retrieved from both RLE compressed and decompressed images.
(R5) Embedded images should be perceptually equivalent to their originals, keeping the embedding distortion as low as possible.

3.2 Defining concepts

In this section, we briefly introduce the concepts needed for the description of the RS embedding method[2] and its LE4RLE modification that preserves file size (in Section 3.4).

First, all palette colors are divided into disjoint (unordered) pairs $\{c_i, c_j\}$ of perceptually similar colors (some colors may be paired to themselves). The set of all color pairs is denoted as P. Furthermore, for each color c_i, we define its flipped color as $\overline{c}_i = c_j$, where $\{c_i, c_j\}$ is a color pair from P.

Next, we extend the flipping operation to a group of k pixels with colors $(c_1, c_2, \ldots, c_k)$ and a binary mask $M \in \{0,1\}^k$: $\overline{G}^M = (c'_1, c'_2, \ldots, c'_k)$, where $G = (c_1, c_2, \ldots, c_k)$ and

$$c_i' = \begin{cases} \overline{c}_i & M_i = 1 \\ c_i & M_i = 0 \end{cases}, \quad i = 1, \ldots, k.$$

The mask M can be the same for all groups (as it is the case in the original RS embedding) or be individually defined for each group (in this paper). We further define the discrimination function $f(G)$

$$f(G) = f(c_1, c_2, \ldots, c_k) = \sum_{i=1}^{k-1} d(c_i, c_{i+1}), \tag{1}$$

where d is the distance between two colors. The selection of color pairs and the distance d is detailed in Section 3.5. Finally, we describe a function that assigns one bit $b(G)$ to each group G:

$$b(G) = \begin{cases} 0, & f(\overline{G}) - f(G) > T \\ 1, & f(\overline{G}) - f(G) < -T \\ \text{undefined}, & \left| f(\overline{G}) - f(G) \right| \le T. \end{cases} \tag{2}$$

The threshold T can be used to achieve different capacity-distortion rate (see Section 4). Note that for natural images the flipped group $\overline{G}$ will be "noisier" than G and thus $Prob\{f(\overline{G}) > f(G)\} > 1/2$. Consequently, $b(G)$ will have more 0's than 1's.

Because $\overline{G} = H \Leftrightarrow G = \overline{H}$, we have $b(\overline{G}) = 1 - b(G)$, whenever $b(G)$ is defined. Also, $b(G)$ is defined if and only if $b(\overline{G})$ is defined.

3.3 RS lossless embedding

Following the original method, the RS lossless embedding starts by dividing the original image X into disjoint groups of the same size and shape (e.g., 2×2 blocks). Let G_i, i=1, 2,…, N be all the groups for which $b_i = b(G_i)$ is defined. The RS algorithm flips some of the groups G_i to G_i' so that their associated bits $b_i' = b(G_i')$ encode the message *and* the (compressed) original bits $C(\{b_i\}_{i=1}^{N}) \,\&\, \{m_j\}_{j=1}^{L}$, where $C(\{b_i\})$ is the losslessly compressed* bit-stream $\{b_i\}$ needed for reconstruction of the original image. Note that because the bit-stream $\{b_i\}$ contains more 0's than 1's, it will be losslessly compressible. As a result, this method can embed up to $N-|C(\{b_i\})|$ message bits m_j.

At the decoder, the compressed bit-stream $C(\{b_i\})$ and the message bits $\{m_j\}$ are extracted. Then, the groups G_i' are flipped as needed to match their associated bits b_i with the extracted and decompressed bit-stream $\{b_i\}$ thus obtaining an exact copy of the original image.

Next, we explain how this scheme can be modified to guarantee file size preservation for RLE encoded BMP images.

3.4 RS lossless embedding with file size preservation

In the RLE BMP format, the image data X is represented by indices x_i to the image palette, which can have up to 256 entries. Let $c(x_i)$ denote the color of the pixel x_i. During embedding, each pixel x_i can either stay unmodified or be changed to $\bar{x}_i$, where $\bar{x}_i$ is the index to the color $\overline{c(x_i)}$.

We start with a simple observation that the size of the RLE compressed image will not be changed by embedding if the length of *all* runs (along image rows) is not changed. This means that any sequence of pixels '*yxx…xz*' can be changed to '*yww…wz*' by replacing x with w, $w \neq y$ and $w \neq z$.

Given the image $X=\{x_i\}$, i=1, 2,…, N_p, represented as a row vector (pixels arranged by rows), we define the invariant image $R=\{r_i\}$ as

$$r_i = \min\{x_i, \bar{x}_i\}\,,\ i=1, 2,\ldots, N_p\,.$$

Thus, the image R does not distinguish between the colors in the pair $\{c(x_i), c(\bar{x}_i)\} \in P$.

When scanning a row of pixels in R, transform this image using the RLE code "nB" only, whenever the index B is repeated n times, and as "00" for End of Line. The sequence of numbers n determines the length of row segments that the embedding algorithm must leave unmodified or modify simultaneously to $n\overline{\mathrm{B}}$. Because each segment will carry the same amount of hidden information regardless of its length, to keep the distortion low, it is better to limit the length of each segment to a small number. In this paper, we use segments consisting of exactly one pixel. The set of all pixels that belong to such segments of length 1 will be denoted as Q

$$x_i \in Q \Leftrightarrow (r_i \neq r_{i-1} \text{ and } r_i \neq r_{i+1}).$$

If the embedding algorithm flips only the pixels in Q, the file size of the embedded image will be preserved under any RLE encoder. Thus, the lossless method with file size preservation proceeds in the same way as the original RS method with one difference – the mask M for each group G is determined by pixels from Q. This mask will reflect which pixels can be modified and which cannot.

To obtain the individual masks, we define a binary matrix $E=(e_i)$, of the same size as the image, that captures which pixels may (1) and must not (0) be modified

$$e_i = \begin{cases} 1 & x_i \in Q \\ 0 & x_i \notin Q. \end{cases}$$

---

* In RS method[2], adaptive arithmetic coding was used to compress the bit-stream $b(G_i)$.

Each group of k pixels $(x_{i_1}, x_{i_2}, ..., x_{i_k})$ with colors $G = (c(x_{i_1}), \ldots, c(x_{i_k}))$ will have its own embedding mask M $M = (e_{i_1}, e_{i_2}, ..., e_{i_k})$. Thus, $\overline{G}^M = (c_1', c_2', .., c_k')$, where

$$c_j' = \begin{cases} \overline{c(x_{i_j})} & x_{i_j} \in Q \\ c(x_{i_j}) & x_{i_j} \notin Q \end{cases}.$$

Note that the embedding mask can be uniquely determined from both the original and embedded images.

Now, let us summarize all steps of lossless embedding with file size preservation.

Encoder

1. Determine pairs of close colors P (see Section 3.5)
2. Calculate the set Q of modifiable pixels.
3. Divide the image into disjoint groups G. Calculate $b_i = b(G_i)$ for all groups of pixels whenever they are defined. Use a pseudo-random order for index i of G_i.
4. Start compressing the bit sequence $\{b_i\}$ to $C\{b_i\}$. Stop the compression at b_k as soon as the inequality $k \geq l + L + length(C\{b_i\}_{i=1}^{k})$ is satisfied (l is the number of bits that encodes message length).
5. Form the composite message *Message_length* & *Message_bits* & $C\{b_i\}$ spanning l, L, and V bits.
6. For each i, if ($b_i \neq i$-th bit of $C\{b_i\} \& \{m_j\}$) then flip G_i to $\overline{G_i}$.

Decoder

1.–2. *The same as in Encoder.*

3. Divide the image into disjoint groups G. Calculate $b'_i = b(G_i)$ for all groups of pixels whenever they are defined. Use the same pseudo-random order for index i of G_i as during embedding.
4. Read $b_1'\, b_2' \ldots\, b_l'$ and message bits $b'_{l+1}\, b'_{l+2} \ldots\, b'_{l+L}$.
5. Set j=1. Decompress the segment $b'_{l+L+1} \ldots\, b'_{l+L+j}$ and denote the length of the decompressed segment V.
6. If $V < l+L+j$ then Go to 5, else Stop. The decompressed bits are $b_1 b_2 \ldots b_{l+L+V}$.
7. For i =1, …, V, if ($b_i \neq b'_{l+L+i}$), flip G_i to $\overline{G_i}$.

Because both the encoder and decoder start from the image decompressed to the spatial domain, the method is insensitive to differences between RLE encoders. Also, presorting the palette to a fixed order (e.g., alphabetically) before determining color pairs P will make the system work after palette reshuffling. The problem of removing or adding duplicate palette entries can be addressed by unifying duplicate entries in the palette to the lowest one from all duplicate indices before embedding and returning the occurrences of the duplicate colors after embedding. In particular, let c_1, c_2, …, c_k are different palette entries corresponding to one RGB color, $c_1 < c_2 < \ldots < c_k$. Before embedding, we modify all pixels with colors c_2, …, c_k to c_1. After embedding, the pixels with colors c_1 are changed back to c_j if they were equal to c_j in the original image, for all j = 2, …, k. The first step guarantees that it will not matter whether or not the duplicate entries are removed and the last step guarantees that the file size will not decrease during data insertion.

3.5 Color pairing and distance *d*

Let $D = \{d_{ij}\}$, i, j =1, …, n, $n \leq 256$, be the matrix of distances between colors i and j after presorting the colors that appear in the image. These distances can be measured in any color space, such as RGB, YUV, or CIELAB. After subjectively evaluating results of experiments with different spaces, we selected the square of the weighted Euclidean RGB distance

$$d = w_r^2 (r_i - r_j)^2 + w_g^2 (g_i - g_j)^2 + w_b^2 (b_i - b_j)^2,$$

where $w_r = 0.35$, $w_g = 0.4$, and $w_b = 0.25$, and r_i, r_j, g_i, g_j, b_i, and b_j are integers in the range from 0 to 255.

Once an appropriate distance measure d in the RGB color space is established, one can attempt to determine the color pairing P that minimizes the distortion with a lower bound on the capacity or maximize the capacity with an upper bound on the distortion.

Such problems can be formulated as an optimal incomplete matching problem[10] in a weighted graph with three weights at each edge and with one constraint. Since the optimal matching can be incomplete, some colors can be paired to themselves. There is no known algorithm for solving this task efficiently in a polynomial time. We skip the details here due to a limited space. Below, we describe a heuristic algorithm that strives to provide good capacity-distortion trade off.

3.5.1 Top-down color matching algorithm

In our previous work[2], we have introduced the color matching algorithm called "Top-down". In this paper, we use an improved version of this approach that gives higher embedding capacity with lower distortion.

The top-down matching algorithm starts with the most isolated color and finds its closest neighbor among all remaining colors. If the distance between these two colors is below a fixed threshold t, both colors are removed from the set of all colors and declared as a "close pair". If the distance is larger than the threshold, the most isolated color is paired with itself and removed. This step is repeated until all colors are paired. The parameter t regulates the trade-off between image distortion and lossless capacity.

Next, we describe a modification of this algorithm that has achieved better results in our experiments.

3.5.2 Improved top-down color matching algorithm

Instead of the most isolated color, this algorithm finds colors with the least number of colors within the distance t. Among them, the pair with its distance closest to t is chosen and removed from the set. This step is repeated until no such pair can be found. The pseudo-code for this algorithm follows.

$k = 1$;

$$b_{ij} = \begin{cases} 0, \text{ when } d_{ij} = 0 \text{ or } d_{ij} > t \\ 1, \text{ otherwise } (0 < d_{ij} \le t) \end{cases}$$

$s_i = \sum_j b_{ij}$, $n_i = \min_{j,\, 0<d_{ij} \le t} \{d_{ij}\}$; for all i

`Repeat`

$m = \min_{s_i > 0} \{s_i\}$;

$a = \arg\max_{i,\, s_i = m} \{n_i\}$; $b = \arg\min_{j,\, b_{aj} = 1} \{d_{aj}\}$;

Set (a, b) as the k-th pair, k=k+1 ;

$s_a = 0$; $s_b = 0$;

For all $s_i > 0$ set $s_i = s_i - b_{ia} - b_{ib}$;

$b_{ia} = 0$; $b_{ib} = 0$; $i = 1, \ldots, n$;

`until` ($s_i = 0$ for all i)

4. EXPERIMENTAL RESULTS

In our experiments, we have used "column" groups G consisting of 4×1 pixels. Because the RLE compression proceeds by rows, such groups provide better capacity than groups of 2×2 or 1×4 pixels. In the original RS method, the threshold T in (2) was set to 0. For color palette images, however, it is advantageous to make T proportional to t to improve the capacity without increasing the embedding distortion. A linear dependence $T = 0.7t$ gave us good results for a wide range $0 < t \le 50$. We note that this relation depends on our choice of the distance measure d.

For brevity, we demonstrate the performance of the proposed method for only 5 test images. Four images were 640×480 true color images (Lenna was 512×512) converted to 256 colors using PaintShop Pro 4.12 with the optimized palette and nearest color options. For all test images, we calculated the capacity (in bits per pixel or bpp), distortion (PSNR), and rate in bits per unit distortion measured in the L_1 norm. Table 1 shows the results for the LE4RLE method that preserves the file size and the original RS method. In the last column, the table also shows the file size increase Δ (in bytes) produced by the original RS method when embedding the maximal length message (the compression was done in the same version of PaintShop Pro). This file size increase bears almost no relationship to the message size (surprisingly) and is mostly influenced by the image content and the efficiency of the RLE compression. The file size increase Δ may become very large if the act of embedding makes the image significantly less compressible using RLE (for Image No. 4, Δ is more than 20 times larger than the message length). Of course, the new LE4RLE method did not lead to any file size increase.

Inspecting the rate in Table 1, we can see that the new method achieves slightly better rate than the original RS method. The absolute capacity decreases by about 20% but, but this is fairly insignificant because the capacity can be adjusted with the parameter t that controls the distortion-capacity trade off (Table 2). Overall, the capacity of the new method is large enough for applications, such as authentication or annotation embedding – the main applications for which the concept of lossless data embedding was originally introduced.

Image	LE4RLE			RS			Δ
	bpp (%)	PSNR	Rate	bpp (%)	PSNR	Rate	
Im_1	3.08	33.3	.0076	4.28	31.4	.0065	9172
Im_2	1.90	33.4	.0050	1.89	32.0	.0032	7634
Im_3	4.63	34.7	.0127	5.51	33.3	.0108	2286
Im_4	4.21	34.6	.0125	5.41	32.1	.0103	33440
Lenna	6.48	30.3	.0148	7.70	27.3	.0088	3186

Table 1. Capacity, distortion, and rate for 5 test images for LE4RLE and the original RS method, $t = 20$. Δ (Byte) is the difference in file size in the original RS embedding.

Image	Capacity (% of bpp)			PSNR (dB)		
	t=10	t=30	t=40	t=10	t=30	t=40
Im_1	1.44	3.98	4.01	37.02	31.35	29.50
Im_2	0.81	2.18	4.12	38.20	33.15	31.56
Im_3	1.68	6.05	6.91	39.30	31.25	30.01
Im_4	1.82	6.25	7.52	40.26	31.97	30.53
Lenna	3.97	8.92	10.58	33.14	31.16	29.04

Table 2. Capacity and distortion as functions of the parameter t for several test images.


Im_1


Im_2


Im_3


Im_4


Lenna

5. LOSSLESS EMBEDDING WITH FILE SIZE PRESERVATION FOR THE JPEG FORMAT

The JPEG format[9] is the most popular image format in current use. This is because it offers a convenient trade off between the file size and the perceptual quality of the encoded image. Our previously developed lossless embedding schemes[2] for JPEG do not preserve the JPEG file size and in some cases the file size increase can be quite disproportional to the embedded message size. This partially negates the advantages of embedding data rather than appending. In this section, we address this issue and describe a lossless embedding technique for sequentially encoded JPEG images that preserves their file size (within a few bytes).

The JPEG encoder consists of three fundamental components (see Fig. 1): Forward Discrete Cosine Transform (FDCT), a scalar quantizer, and an entropy-encoder. After the DCT is applied to an 8×8 block of pixels transforming the block from spatial domain to the frequency domain, DCT coefficients are quantized using the quantization table. The quantized coefficients are arranged in a zigzag order and pre-compressed using the Differential Pulse Code Modulation (DPCM) on DC coefficients and RLE on AC coefficients. Finally, the symbol string is Huffman-coded to obtain the final compressed bit-stream. After pre-pending the header, the final JPEG file is obtained.

Our lossless embedding scheme with file size preservation works with the Huffman-decompressed stream of intermediate symbols. This bit-stream is modified in a careful manner to make sure that the final file size after Huffman compression stays the same within a few bytes. To understand the embedding principles, we need to describe the lossless part of JPEG compression in more detail.

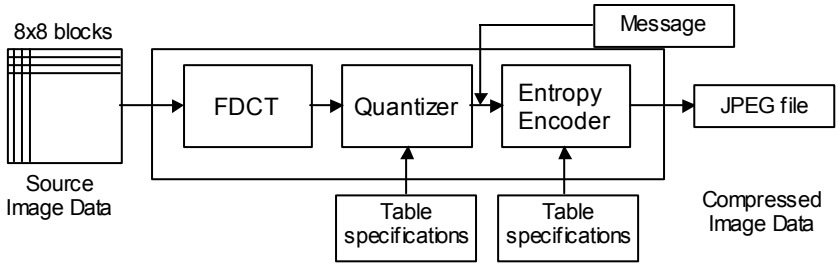

Fig. 1. Lossless embedding in JPEG files.

5.1 The JPEG entropy coder

The entropy coder consists of two steps: (1) DPCM encoding of the DC term and runlength encoding of the AC coefficients into a sequence of intermediate symbols and (2) Huffman coding. The purpose of the DPCM is to decorrelate the DC term because DC coefficients from neighboring blocks still exhibit significant local correlations. The AC coefficients, on the other hand, contain long runs of zeros due to the quantization. Thus, AC coefficients are conveniently encoded using the runlength encoding. The DPCM coding of DC coefficients and the runlength coding of AC coefficients produce a sequence of intermediate symbols, which is finally entropy coded (Huffman) to a data stream in which the symbols no longer have externally identifiable boundaries.

Our embedding technique works with the sequence of intermediate symbols. We ignore the DC coefficients because their modifications usually lead to visible artifacts. To explain how we modify the runlength encoded AC coefficients, we need to describe the runlength coding algorithm in more detail.

5.2 Run length encoding of AC coefficients

Run length encoding (RLE) is a simple lossless compression that assigns short codes to long runs of identical symbols. As mentioned above, majority of AC coefficients in each block are usually zeros. To efficiently utilize this fact, the AC coefficients are coded in a special RLE format as pairs of intermediate symbols (S1, S2). The codeword S1 represents both the number of zeros before the next nonzero DCT coefficient and the *category* (number of bits required to represent its amplitude). The S2 symbol defines the amplitude and sign of the nonzero coefficient. The symbol S1, S1=(Run/Category), is a composite 8-bit value of the form S1 = binary 'RRRRCCCC'. The 4 least significant bits, 'CCCC', define a category for the amplitude of the next non-zero coefficient in the block. The 4 most significant bits, 'RRRR', give the position of the coefficient in the block relative to the previous non-zero coefficient (i.e., the run-length of zero coefficients between non-zero coefficients):

- Run (RRRR): the length of the consecutive zero-valued AC coefficients preceding the next nonzero AC coefficient, 0≤Run≤15.
- Category (CCCC): the number of bits needed to represent the amplitude of the next nonzero AC coefficient, 0≤Category≤15.
- S2 (amplitude): S2 represents the amplitude of the next nonzero AC coefficient by a signed integer.

Once the quantized coefficient data from each 8×8 block is represented in the intermediate symbol sequence described above, variable-length codes are assigned. Each S1 (Run/Category) is encoded with a variable-length code (VLC) from a Huffman table. Each S2 (amplitude) is encoded with a "variable-length integer" (VLI) code, which is an index into the amplitude value field whose length in bits is given in the second column of Table 3.

Both VLCs (S1) and VLIs (S2) are codes with variable lengths, but VLIs are not Huffman coded. They are *appended* to the Huffman coded S1 to form the final JPEG bit-stream. So, we can change a particular VLI as long as the modified value is from the same category (has the same length) without changing the JPEG file size. Consequently, if all the embedding changes have this property, the JPEG file size will be preserved.

Amplitude value field	Category	AC size
0	0	N/A
–1, 1	1	1
–3, –2, 2, 3	2	2
–7,…,–4, 4,…, 7	3	3
–15,…,–8, 8,…, 15	4	4
–31,…, –16, 16,…, 31	5	5
–63,…,–32, 32,…, 63	6	6
–127,…, –64, 64,…,127	7	7
–255,…,–128, 128,…, 255	8	8
–511,…,–256, 256,…, 511	9	9
–1023,…,–512, 512,…,1023	10	A
–2047,…,–1024, 1024,…, 2047	11	B
–4095,…,–2048, 2048, …, 4095	12	C
–8191,…,–4096, 4096,…, 8191	13	D
–16383,…,–8192, 8192, 16383	14	E
–32767,…,–16384, 16384, 32767	15	N/A

Table 3. Runlength coding category and amplitude of AC coefficients.

5.3 Lossless embedding with file size preservation

We build our lossless file size preserving method around the lossless embedding scheme for JPEGs[2]. As explained in the previous paragraph, in order to preserve the file size, a given DCT coefficient d from category C can only be changed to another coefficient d' from the same category C. To minimize the embedding distortion, we want this change to be as small as possible. Also, because changes to DC coefficients usually introduce visible distortion, we confine the embedding modifications to AC coefficients, only.

Considering the requirements above, we further limit the embedding changes to the same category, swapping values of AC DCT coefficients within the following pairs: (–2,–3), (2,3) from category 2, (–7,–6), (–5,–4), (4,5), (6,7) from category 3, etc. During embedding, one value from the pair may be changed to the other value from the same pair. The value pairs are called embedding pairs.

If we assign parity 0 to all even valued coefficients and parity 1 to odd valued coefficients, then the parities of the DCT coefficients that participate in embedding pairs in the original JPEG file is a binary sequence T that is losslessly compressible. This is because in natural images the distribution of DCT coefficients is generalized Gaussian centered at 0 and thus the sequence T contains more 0's in T than 1's.

The rest of the embedding follows the RS method for JPEGs[2]. We first losslessly compress the sequence T, obtaining the compressed bit-stream $C(T)$, $|C(T)|<|T|$, append the message bits M to the compressed bit-stream, $C(T)\&M$, and embed this composite message as the parities of DCT coefficients participating in embedding pairs (the capacity of this scheme is $|T|-|C(T)|$). In our implementation, we used a context-free arithmetic compression[8].

Due to the generalized Gaussian distribution of DCT coefficients, the coefficients occur with highly uneven probabilities. Thus, to obtain a more efficient lossless compression of the sequence T, we divide T into several subsequences (each subsequence corresponding to one category) and perform the arithmetic compression for coefficients from each category separately.

Encoder

1. Huffman-decode the original JPEG file.

2. Either sequentially, or along a key-dependent path, read all DCT coefficients d_i belonging to embedding pairs from all Huffman-decoded data. Form the sequence $T=\{t_i\}$, t_i=parity(d_i).
3. Compress T using the arithmetic encoder as described above to obtain the compressed bit-stream $C(T)$.
4. Concatenate m message bits M, $m<|T|-|C(T)|$, to the compressed bit-stream, obtaining $T'=C(T)\&M$.
5. For each i, if $t_i \neq t_i'$, modify d_i to d_i', where (d_i, d_i') is an embedding pair.
6. Using the same Huffman code table, re-encode the modified Huffman-decoded data to obtain the embedded JPEG file.

Decoder

1. Huffman-decode the JPEG file.
2. Either sequentially, or along a key-dependent path, read all DCT coefficients d_i' belonging to embedding pairs from all Huffman-decoded data. Form the sequence $T'=\{t_i'\}$, t_i'=parity(d_i').
3. Read the message M from T' and decompress $C(T)$.
4. Either sequentially, or along a key-dependent path, modify all DCT coefficients d_i' belonging to embedding pairs so that their parities match the decompressed sequence T: parity(d_i)=t_i. After re-encoding the modified Huffman-decompressed coefficients, the original JPEG file is obtained.

6. EXPERIMENTAL RESULTS

We have tested 50 images to see how the capacity, embedding distortion (measured as PSNR), and file size change with different images, JPEG quality factors, and categories. We note that including categories higher than 4 does not necessarily lead to higher capacity because the bias between coefficient parities for higher categories is very small. The overall best performance was obtained for embedding pairs from categories 2, 3, and 4.

Image ($M\times N$)	JPEG Quality	Capacity (bits)		PSNR		File size (bytes)		
		2	2,3,4	2	2,3,4	Original	2	2,3,4
Balcony (578×891)	98	5978	7520	50.3	49.2	314180	314238	314238
Banoch (1024×768)	90	3265	3333	44.5	43.3	258830	258864	258864
beacon60 (568×512)	40	1303	1378	40.4	39.4	36787	36787	36787
Bloom (960×1280)	97	11797	13360	49.5	48.4	586566	586723	586727
boat80 (512×768)	20	1025	1103	37.9	37.2	27887	27891	27891
Bridge (1280×960)	90	8290	8564	41.6	40.5	604367	604477	604494
brook50 (512×768)	50	2132	2485	37.0	35.8	75353	75370	75367
Bu (960×1280)	97	14110	15892	50.6	49.5	512078	512241	512261
drift30 (512×768)	70	1867	2214	40.7	39.2	76847	76881	76882
Falls (960×1280)	93	5905	6396	51.4	50.3	307041	307133	307138
flight20 (512×768)	80	1084	1207	44.9	43.9	51930	51939	51935
Fog (960×1280)	93	8938	9391	49.5	48.5	267218	267309	267305
girl1 (678×512)	98	4946	6119	50.1	48.8	271388	271422	271432
parrot20 (512×768)	80	1078	1162	46.1	44.9	48757	48772	48777
Tree (960×1280)	93	7278	8751	44.4	43.1	558636	558737	558764
windows40 (512×768)	60	2016	2270	39.3	38.2	70556	70581	70579

Table 4. Capacity for various test images with different JPEG quality factor for amplitude categories 2, 3, and 4.

In Table 4, the capacity is shown for one category 2 and then for three categories 2, 3, and 4. For most images, the capacity difference between one and three categories is about 10%. For some images, however, the gain in using three categories as opposed to just one could be quite substantial (e.g., for image "girl1"). The capacity is strongly influenced by the image content and size. Quite understandably, higher quality factors lead to higher capacities than lower factors. Since the target application of lossless embedding is authentication, possibly combined with metadata embedding, the capacities seem to be adequate for this purpose.

The embedding (removable) distortion is measured for the maximal message length using PSNR. The results indicate that the embedding distortion is low (and removable, in any case). The last three columns show the file size for the original image and the embedded image (using one category and using three categories). The file size changes are quite small, usually limited to a few bytes. The difference between the original and embedded file size is not caused by the

entropy-code change but by JPEG intrinsic byte-alignment structure and zero padding in order to distinguish marker segments and entropy-coded segments[9]. This difference can be eliminated if we read entropy-coded data from the JPEG bit-stream and skip the entropy-coded data that may cause byte-alignment and zero padding change.

7. CONCLUSIONS

Virtually all lossless embedding techniques increase the file size of the embedded image. The increase in the file size is, however, quite often disproportional to the length of the embedded data. This issue has been brought up to our attention by the sponsors of this research (AFOSR and ARFL at Rome, NY). The file size increase is a negative property that potentially negates the advantages of lossless embedding schemes. To our best knowledge, this paper is the first one that presents lossless data embedding techniques that preserve the size of the watermarked file. We describe lossless embedding methods that preserve the file size of images in the RLE encoded BMP and the JPEG formats. The performance of the new schemes is evaluated in terms of their capacity, embedding rate, and embedding distortion.

The method for RLE BMPs is a modification of the RS embedding method[2]. The embedded message and the exact copy of the original image can be obtained even after the RLE encoded BMP embedded image is decompressed to the RGB BMP format, re-encoded using a different RLE encoder, or when its palette is permuted or colors added to it (or when duplicate colors are removed from the palette). By using a more sophisticated color-matching algorithm, the capacity of this new lossless file size preserving method is in fact slightly higher than for the original method lossless method[2] for palette images.

In the second part of this paper, we introduce a lossless embedding technique with file size preservation for sequentially encoded JPEG images. Again, previously proposed lossless techniques for JPEGs could increase the file size of the embedded JPEG files by an amount that was much larger than the size of the embedded message itself. Our new method uses a different embedding principle while making sure that the embedded JPEG file size stays the same. We work with the sequence of intermediate symbols (after Huffman decompression) and modify the amplitude of certain DCT coefficients by at most one. Because the amplitude category is not Huffman coded and because the modifications are always confined to the same amplitude category, the embedded file size stays the same.

Lossless data embedding with file size preservation is a useful new technology that emphasizes the advantage of lossless embedding of data as opposed to appending. We envision image authentication, image integrity protection, and metadata embedding as the main application areas for the new embedding technology.

Future research will be directed towards development of lossless embedding techniques with file size preservation for other image formats that include lossless compression, such as GIF, PNG, or JPEG2000. Also, obtaining theoretical upper bounds on capacity given the compression method and properties of typical images is an open and interesting question that deserves further study.

ACKNOWLEDGEMENTS

The work on this paper was partially supported by Air Force Research Laboratory, Air Force Material Command, USAF, under a research grant number F30602-02-2-0093 and partially by the AFOSR grant F49620-01-1-0123. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation the on. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Air Force Research Laboratory, the Air Force Office of Scientific Research, or the U. S. Government.

REFERENCES

1. C.W. Honsinger, P. Jones, M. Rabbani, and J.C. Stoffel, “Lossless Recovery of an Original Image Containing Embedded Data”, US Patent application, Docket No: 77102/E–D, 1999.
2. J. Fridrich and M. Goljan, “Lossless Data Embedding for all Image Formats”, *Proc. of EI SPIE*, *Security and Watermarking of Multimedia Contents IV,* vol. 4675, San Jose, pp. 572–583, 2002.

3. M. Celik, G. Sharma, A.M. Tekalp, and E. Saber, "Localized Lossless Authentication Watermark (LAW)", *Proc. of EI SPIE, Security and Watermarking of Multimedia Contents V,* vol. 5020, Santa Clara, pp. 689–698, 2003.
4. C. De Vleeschouwer, J.-F. Delaigle, and B. Macq, "Circular Interpretation of Histogram for Reversible Watermarking"*, IEEE 4th Workshop on Multimedia Signal Processing,* pp. 345–350, 2001.
5. Y. Tian, "Wavelet-Based Reversible Watermarking for Authentication", *Proc. of EI SPIE*, *Security and Watermarking of Multimedia Contents V*, vol. 4675, San Jose, pp. 679–690, 2002.
6. T. Kalker and F.M. Willems, "Capacity Bounds and Code Constructions for Reversible Data-Hiding", *Proc. of EI SPIE*, *Security and Watermarking of Multimedia Contents V,* vol. 5020, Santa Clara, pp. 604–611, 2003.
7. D. Maas, T. Kalker, and F.M. Willems, "A Code Construction for Recursive Reversible Data-Hiding", *Proc. Multimedia and Security Workshop at ACM Multimedia*, Juan-les-Pins, France, December 6, 2002.
8. K. Sayood, *Introduction to Data Compression*, Morgan Kaufmann Publishers, San Francisco, California, pp. 87–94, 1996.
9. ITU/CCITT, "Information technology-digital compression and coding of continuous-tone still images-requirements and guidelines".
10. L. Lovasz, M.D. Plummer, *Matching Theory*, Ann. Disc. Math. 29, North-Holland, Amsterdam, 1986, ISBN: 0-444-87916-1, pp. 369–382.