A Digital Signal Processing Report on Image Steganography using LSB Matching

By: Himanshu Sharma
Roll Number: 1610110149
Dept. of Electrical Engineering (ECE)

Shiv Nadar University,
Gautam Buddh Nagar, Greater Noida, Uttar Pradesh 201314

Under the guidance of Professor Vijay K. Chakka

1 Paper Comprehension

Image Steganography is the art of hiding information inside a digital image. Steagnography does not restrict to only plain text but it includes text, video, audio and image hiding also. In this report, the author has restricted himself to text data type only. Even when doing text embedding, there are immense possible algorithms to choose from. The most popular among them is the LSB replacement or Least Significant Bit replacement. With this type of algorithm, the least significant bit of each pixel (only one channel out of the RGB pallete) is changed by a bit decided according to the message bit. Therefore, we should not expect much change in the image after data hiding because the least significant bit is changed. For example, lets take a blue channel with value of 145. In binary, it is equivalent of 10010001. If by some technique, the last bit is changed to 0, then the decimal equivalent would become 144, which does not change the image much. This is the power of LSB replacement. It basically relies on the fact that hiding message bits in the LSB of each pixel does not affect the image much, in fact, the image practically remains as it is. There are, however, techniques now in modern signal processing science which can detect that some data is hidden in the image or not, a field called *steganalysis*. A successful hiding algorithm would be that, that would allow high payload and still look similar to the original image.

1.1 LSB Matching

In LSB matching, if the message bit does not match with the pixel's LSB, then ± 1 is randomly added to that pixel value. Unlike LSB replacement, where the pixel's LSB is just replaced by the message bit, here, a set of conditions are used to modify the pixel. LSB Matching could not be detected using the techniques used to detect LSB replacement. However, now it has been proved that LSB matching acts like a low pass filter on the digital images and therefore, this fact is utilized to detect whether LSB matching is applied on an image or not.

Usually, two consecutive pixels are taken alongwith two consecutive message bits. The consecutive pairs are chosen randomly based on the *pseudo-random number generator* (PRNG). If the LSB matching is used as it is, then any pixel could be chosen with every pixel pair having equal chances of being selected by the algorithm. This has a flaw. This type of approach makes it difficult to disguise a changed pixel to the pixel surrounding it. For example, if lot of pixels are changed in a close vicinity, then they could be easily identified if the region is light in color, like sky which is light blue in color. The paper chosen here tries to rectify this problem by chosing those pixels on the image which lie on the edges. Edges are usually sharper than the surrounding regions and therefore its not easy to identify the change in pixel color on the edges. Similar papers have already been published. All of them suggest to use what is called the *pixel-value difference* (PVD).

In PVD, what we do is that when we try to hide a message bit in a pixel's LSB, the pixel value is compared with its neighbouring pixels. If the difference is large, then more bits can be accomodated in that region without them being easily identified. Why? Because if there is a large difference in the pixel values then on changing the pixel value will not generate any significant difference. For example, if a pixel that is to be modified has a value of 45 and it's neighbouring pixel has a value of 145, then the difference $\Delta = 145 - 45 = 100$. Now, if by some technique if this pixel value if changed to 46, then the difference would become $\Delta' = 145 - 46 = 99$. To a human eye, this difference is not accountable. PVD is a good approach, in fact, far more better that PRNG because it utilizes the fact that sharp changes can be used to hide the information.

In both LSB replacement and LSB matching, a travelling order is generated using PRNG which also acts as a key for decoding the stego-image. In both of these algorithms, the LSB of the selected pixel becomes equal to the message bit. According to the algorithm, the if the two consecutive pixels are x_i and x_{i+1} and the consecutive message bits are m_i and m_{i+1}, then the pixels are modified such that x_i becomes $x_i^{'}$ and x_{i+1} becomes $x_{i+1}^{'}$ and the following relation holds.

$$LSB(x_i^{'}) = m_i \text{ and } LSB\Big(\Big\lfloor \frac{x_i^{'}}{2} \Big\rfloor + x_{i+1}^{'}\Big) = m_{i+1}$$

From experiments done by the authors of the paper it has been shown that even if the cover image has rough textures, it will still have smooth regions in every 5×5 non-overlapping blocks. So, if by any chance, a pixel is selected in that region for message hiding then it could be easily identified. This is what the paper aims to solve.

1.2 Bit Planes

We now come to a brief discussion on what is called the bit planes. Bit planes help us visualize the importance of the significant bits used to represent the images. They clearly display the fact that altering the LSB of pixels is

less damaging to the original image than altering the MSB of the same original image. Before we discuss them in great detail, let me show an example. Suppose we have a cover image which is shown below.

Figure 1: Cover Image

Its bit planes are shown below. The LSB plane and the MSB planes are what we display first.

Figure 2: MSB Plane

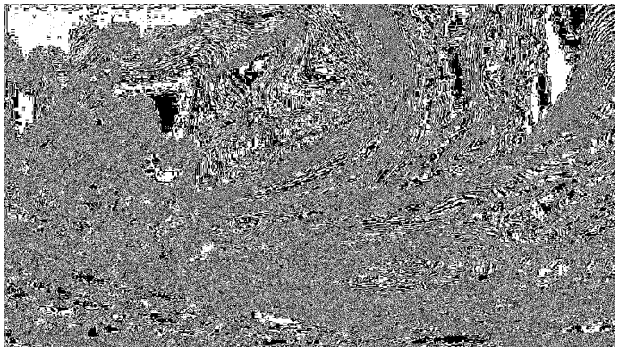

Figure 3: LSB Plane

Changing a pixel value in the MSB plane is more prone to human eye detection as compared to the LSB plane. This is because a MSB plane has more structured black and white regions, and so, changing any pixel value, for example say, that a pixel is changed from white to black in the white region, then it would be easily detected. Whereas, on a LSB plane, the black and white regions are more uniformly aligned, just like *white noise* on a television screen. Inverting any color on the LSB plane won't change the visual effect. Hence, it is always advised to change the LSB plane in stegnography.

The idea is to generate these images using a single values. Since we have 3 values associated with a single pixel *(R, G, B)*, it would be a great idea to convert the image to a grayscale image, so that each pixel is represented by a single value. Now each pixel value, which initially was a three dimensional array, is converted to a one dimensional array having a single value. This matrix of single values (grayscale image) is passed to a function which converts these decimal pixel values to 8 bit binary number. As per the request of the user, the i^{th} bit from each binary pixel is accessed. So, for example, if a pixel has 8-bit binary representation as 11110101 and the user asks for fifth bit plane, then the fifth bit from starting would be accessed and that is 0 for this example. These values are stored in another matrix. So now we have a matrix with only 1 and 0s. We now multiply each and every value of the matrix by 255. The matrix becomes full of 255 and 0s. Zero represent black color and 255 represent white color. When saved, this image is a black and white i^{th} bit plane.

Below, I have shown all the bit planes of the above cover image.

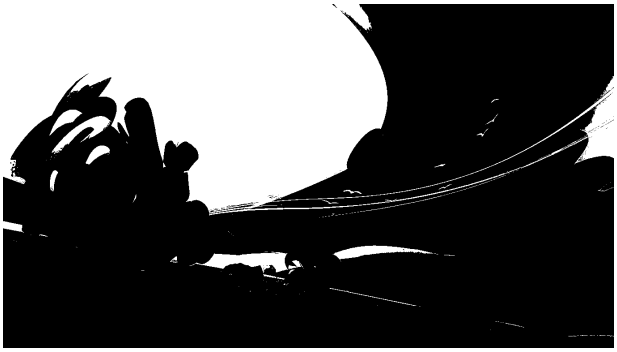

Figure 4: 1^{st} bit plane or the MSB Plane

Figure 5: 2^{nd} bit plane

Figure 6: 3^{rd} bit plane

Figure 7: 4^{th} bit plane

Figure 8: 5^{th} bit plane

Figure 9: 6^{th} bit plane

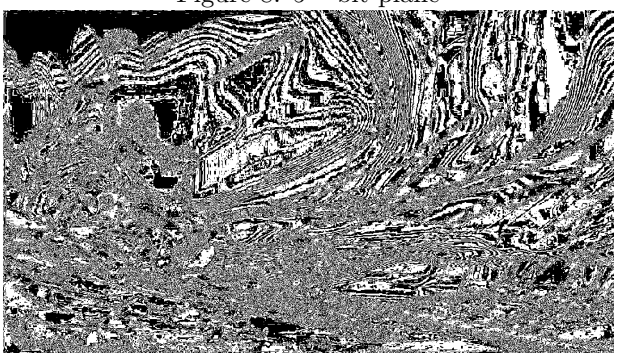

Figure 10: 7^{th} bit plane

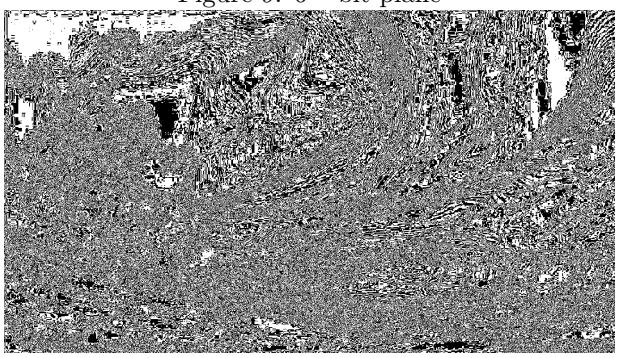

Figure 11: 8^{th} bit plane or the LSB plane

Clearly, as we increase the bit plane number, we are bound to get more better results. Increasing the bit plane number implies less human eye detection in a pixel change. Now let us see the mathematical reasoning behind the bit planes.

Let us define a closed form expression which converts a binary number to its decimal equivalent.

$$d(n) = \sum_{i=1}^{n} g(n+1-i,\ b)2^{n-i} \tag{1}$$

where $g(k, b)$ returns the k^{th} bit from left of the binary number b and n denotes the length of the binary number. Here we are dealing with $n = 8$. Let us say that the j^{th} bit is changed while doing some steganography. The j^{th} bit could be LSB or MSB or any other bit in between, we are not concerned about that for now. So, we know that $1 \leq j \leq n$. Let the original binary number be α and the binary number after the changed bit be denoted by β. Hence we get,

$$d_1(n) = \sum_{i=1}^{n} g(n+1-i,\ \alpha)2^{n-i} \text{ and } d_2(n) = \sum_{i=1}^{n} g(n+1-i,\ \beta)2^{n-i}$$

If we take the difference $\Delta d(n) = (d_1 - d_2)(n)$, the we get,

$$\Delta d(n) = \sum_{i=1}^{n} g(n+1-i,\ \alpha)2^{n-i} - \sum_{i=1}^{n} g(n+1-i,\ \beta)2^{n-i} \tag{2}$$

Assuming all other bits of α and β to be same except for the j^{th} bit, we get,

$$\Delta d(n) = g(n+1-j, \alpha)2^{n-j} - g(n+1-j, \beta)2^{n-j} = 2^{n-j}\Delta g(n, j)$$

where, $\Delta g(n, j) = g(n+1-j, \alpha) - g(n+1-j, \beta)$ and thus we get,

$$\delta(n) = |\Delta d(n)| = 2^{n-j}|g(n, j)| \tag{3}$$

Note that $\Delta d(n)$ is the difference in the decimal values of the binary numbers which can be positive, negative or 0, $\delta(n)$ is always non-negative. Now, $|\Delta g(n, j)|$ can be defined as follows;

$$|\Delta g(n, j)| = \begin{cases} 1, & j^{th} \text{ bit of } \alpha \text{ and } \beta \text{ are different} \\ 0, & \text{otherwise} \end{cases}$$

The *otherwise* case is trivial because if there is no difference in even the j^{th} bit of these binary numbers, then both of them are identical. This means that the number was not changed. But if the number has change, then $|\Delta g(n, j)|$, for sure, is 1. Now, it is in this part we are concerned with the difference. We need to minimize this difference $\delta(n)$ because then only our stego-image will be statistically close to the original image. The value of $\delta(n)$ could be minimum only if $n = j$ (see equation 3), i.e., $\delta(n) = |\Delta g(n, j)|$ which is 1 if we assume that the pixel bit was changed. That is, the difference in decimal value has the minimum magnitude of 1, 0 being the trivial case that the binary numbers were never changed. Since the decimal difference value will decide the grayscale of the stego-image, we want to minimize it. This proves that if we want statistically similar features in the stego-image, we should take $j = n$, i.e., we should consider the LSB only, because that would affect the orignal decimal value by ± 1 only.

If we do consider $j = 1$, then $\delta(n) = 2^{n-1}|\Delta g(n, 1)|$. Again, $\Delta g(n, 1) = \pm 1$ if we consider that the pixel value was changed. However, it is clear that 2^{n-1} will yield the maximum difference, and therefore, $j = 1$ should never be taken, or, MSB should never be changed.

1.3 The Algorithm

The paper provides a scheme to optimize the traditional LSB algorithm. It covers an extra mile by embedding the key information in the stego-image itself. We can also say that first the pixels are changed according to the algorithm and then the key information is embedded into the image. Finally, the image thus obtained is the stego-image.

Let me now explain how the data embedding is done in the image.

Data Embedding

Data embedding in the image starts by a new process which is generally not found in traditional steganography algorithms. First the image is divided into non-overlapping blocks of size $B \times B$ (as per the paper). Then we rotate each and every individual block by some random angle θ, where $\theta \in \{0, 90, 180, 270\}$ in degrees.

Dividing the image into non-overlapping blocks and then rotating each block by a random angle increases the sequrity. That is because if we rotate the image before we embedd message bits in it and then later on rotate back those blocks to form the cover-like image, we have actually gained a great deal of security as compared to the image on which the embedding was done directly. Think about it, since the image is rotated back after embedding, all the embedded bits have been shuffled. Now, if some attacker tries to bruteforce, then even if he guesses the correct key, he won't be able to get the correct message.

Now, the image is raster scanned. Raster scan is like converting an image into a row vector. In language like Python which I am using, this transformation can be achieved by using the statement `img.flatten()`, where `img` is the image as a numpy array. To read about numpy, please visit the docs. After this step, consecutive pixels are selected in pairs and following set is defined.

$$S(t) = \{(x_i, x_{i+1}) || x_i - x_{i+1}| \geq t, \forall (x_i, x_{i+1}) \in V\} \tag{4}$$

where V is the raster scanned image and t is some value in the set $\{0, 1, 2, ..., 31\}$. In other words, $S(t)$ is the set of all those consecutive pixels in the raster scanned image which have a differece greater than or equal to the parameter t. After this, the threshold value T is calculated by the following method.

$$T = \text{argmax}_t \{2 \times n(S(t)) \geq n(M)\} \tag{5}$$

where, $n(S)$ denotes the number of elements in S and M is the message. This equation, that is, equation 5 checks whether the message bits can be embedded in the image or not. Now let us understand what this equation actually means. Suppose a hypothetical case of $S(t) = \{(x_1, x_2), (x_3, x_4), (x_5, x_6)\}$ for some $t = t_1$ and let M be a 5 bit stream. Then, we find that $2 \times 3 \geq 5$ where $n(S(t)) = 3$. We chose some other t now which is $t = t_2 > t_1$. For this, let us say that we get $n(S(t)) = 2$, impling $2 \times 2 \geq 5$ which is false and therefore $T = t_1$. This is what equation 5 tries to say. It gives us that maximum t for which the condition in the equaiton holds true. It is clear why we are multiplying by 2. Thats because each pixel can hold one bit from the message stream and S contains a tuple of such pixels. In total, we have twice the length pixels. A good observation is that if $T = 0 \; \exists \; t$, then that would mean that our equation 5 is satisfied only for max $t = 0$. Hence, our equation 4 would become,

$$S(0) = \{(x_i, x_{i+1}) || x_i - x_{i+1}| \geq 0, \forall (x_i, x_{i+1}) \in V\}$$

This is what a conventional LSB steganography technique does. It just embedds the message stream without thinking about the difference between the consecutive pixels.

We now come to the discusssion of the main pseudocode which actually embedds the data in the image. Below, I have shown the algorithm used for LSB matching.

Algorithm 1 LSB Matching

```
procedure HIDE
    if LSB(x_i) = m_i then
        if f(x_i, x_{i+1}) = m_{i+1} then
            (x_i', x_{i+1}') = (x_i, x_{i+1})
        else
            (x_i', x_{i+1}') = (x_i, x_{i+1} + r)
    if LSB(x_i) ≠ m_i then
        if f(x_i − 1, x_{i+1}) = m_{i+1} then
            (x_i', x_{i+1}') = (x_i − 1, x_{i+1})
        else
            (x_i', x_{i+1}') = (x_i + 1, x_{i+1})
```

This algorithm is used for hiding the text data inside the image. r is any random value in $\{1, -1\}$. Where, $(x_i^{'}, x_{i+1}^{'})$ are the new pixel values after data hiding. After this, the image blocks are rotated back to form the orignal image back and the angle values, threshold T and the blcok size are bundled into a binary file and returned as the key, back to the user.

Data Extraction

Now, to get back the message stream, the image is first divided into blocks of size $B \times B$ again and then each of these blocks is rotated by the angles provided in the key file. Then those pixels are found in which data hiding was actually done using the inequality $|x_{i+1} - x_i| \geq T$. Now, to get back two units of the stream, the formula given in section 1.1 is used (refer section 1.1).

1.4 Conclusions

In the case of data embedding, the lower the value of T, more number of bits from the message can be embedded in the cover image. The following reasoning proves this; if we have two thresholds T_1 and T_2 with $T_1 < T_2$, then this would mean

$$\text{argmax}_t\{2 \times n(S_1(t)) \geq n(M)\} < \text{argmax}_t\{2 \times n(S_2(t)) \geq n(M)\}$$

where,

$$S_1(t) = \{(x_i, x_{i+1}) || x_i - x_{i+1}| > t_1, \forall (x_i, x_{i+1}) \in V\}$$
$$S_2(t) = \{(x_i, x_{i+1}) || x_i - x_{i+1}| > t_2, \forall (x_i, x_{i+1}) \in V\}$$

Because $T_1 < T_2 \implies t_1 < t_2$ and that would mean that S_1 has lesser threshold value when compared to S_2. This would mean that more pixels would be available which satisfies $|x_i - x_{i+1}| > t_1$ than the pixels which satisfy $|x_i - x_{i+1}| > t_2$. Hence, S_1 would contain more elements or tuples than S_2. This is what the very first statement is saying, that if, threshold is small, more pixels could be accomodated.