

Detecting LSB Steganography in Images

Ankit Gupta, Rahul Garg

Abstract

In this paper, we present techniques to detect Least Significant Bit (LSB) steganography for images. Here, images can be hidden inside lower bits of the pixels of the cover image. We use image statistics to detect two cases - when hidden image is stored as one big chunk (Simple Mode) or spread out(Shuffle Mode). The results are presented on a synthetically generated database of around 350 steganographic images. We also analyze the detection rate of a popular steganography tool, StegDetect against a hiding tool, JPHide. Based on this analysis, we present intuitions about what types of images can act as good cover images for this tool.

1 Introduction

Image steganography is the science of hiding data inside cover images for security. Images have a lot of visual redundancy in the sense that our eyes do not usually care about subtle changes in color in an image region. One can use this redundancy to hide text, audio or even image data inside cover images without making significant changes to the visual perception. Image steganography is becoming popular on the internet these days since a steganographic image, which just looks like any other image, attracts a lot less attention than an encrypted text and a secure channel.

In this paper, we look at a special case of image steganography - hiding images inside images. From now on, we will use the word *hidden images* for the former, *cover images* for the latter, and *stegified image* for the cover image containing a hidden image. There are already a bunch of techniques available for this purpose and Least Significant Bit (LSB) Steganography is one of them. LSB steganography is a very simple algorithm where higher bits of the color channels of hidden images are stored in lower few bits of the color channels in the cover image. We examine two variants of this approach - storing the hidden image as one big chunk (Simple Mode) or spreading out the pixels while hiding (Shuffle Mode). We found no downloadable tools on the internet which can detect simple LSB steganography and recover the hidden images. Hence we build the tool and present its detection and recovery results on a synthetically generated database of stegified images.

We also wanted to investigate if there are specific properties of cover images for which steganography is hard to detect. We do the cover image analysis using existing image hiding and recovery tools, JPHide and StegDetect respectively, which work for JPEG images.

The remainder of this paper is organized as follows. Section 2 talks about some existing techniques for image steganography. In Section 3, we talk about the generation of steganographic image database. In Section 4, we explain our algorithm for detecting Simple Mode

steganography. In Section 5, we explain how we extend our algorithm to detect Shuffle Mode steganography. We present the quantitative results and analysis in Section 6 and conclude the paper in Section 7 with a summary and discussion about future goals.

2 Related Work

Images are a popular cover medium for steganography given the high degree of redundancy present in digital representation of an image (despite compression). [3] and [9] provide an overview of concepts and practices related to image steganography. The steganographic algorithms can be classified in a couple of ways:

- Spatial vs Transform: The algorithm may exploit the redundancy present in the spatial domain or the frequency domain.
- Model based vs ad-hoc: The algorithm may model the statistical properties of the medium trying to preserve them or embedding can be done in an ad-hoc manner.

The LSB steganography technique is the most common form of technique in spatial domain. This technique makes use of the fact that the least significant bits in an image could be thought of as random noise and changes to them would not have any effect on the image. In Transform domain, embedding is done by altering DCT (Discrete Cosine Transform) coefficients. One of the constraints in transform domain is that many of the coefficients are zero and altering them changes the compression rate of the image. That is why the information carrying capacity of an image is much lesser in transform domain than in spatial domain. Two popular transform domain steganography algorithm are F5 [13] and OutGuess [7]. F5 has two important features. First, it permutes the DCT coefficients before embedding aiming to distribute the induced changes uniformly over the image. Second, it employs matrix embedding to minimize the amount of change made to DCT coefficients. Outguess is also a two step algorithm. It first identifies redundant DCT coefficients which have minimal effect on the cover image, and then depending on the information obtained in initial steps, chooses bits in which it would embed the message. Both F5 and OutGuess have been successfully attacked. Two other tools available on the internet for LSB steganography in transform domain are JSteg and JPHide. While JSteg modifies all the DCT coefficients, JPHide modifies some predecided set of coefficients and hence is more difficult to detect. An overview of these techniques can be found in [8].

More sophisticated techniques try to model image statistics and try to minimize changes to them. For example, the method in [10], the transformed image coefficients are broken down into two parts and replaces the perceptually insignificant component with the coded message signal.

There is some work on exploring what is a good cover medium. For example, in [5], good cover media are selected given some knowledge about the steganalysis tool.

Steganalysis is the science of detecting steganography. Most methods only aim to detect whether a medium contains hidden data and do not seek to recover the hidden message as well as that is a very hard problem in a general setting. Steganalysis methods can be classified into two categories

- Specific to a particular steganographic algorithm.
- Universal steganalysis

Obviously, success rate with former kind of methods is much higher. Provos et al [8] present a tool StegDetect which is targeted at JSteg, JPHide and OutGuess. They show good detection results and high processing speeds and further use this tool to crawl the internet and find steganographic images.

Most steganalysis techniques in the second category look at how the embedding modifies the statistics of the medium but many of them do not take into account that medium are images which have certain characteristic statistics [2]. However, [1] does try to exploit natural image statistics to some extent. None of the methods make any assumption about the kind of data hidden inside the medium. Universal steganalysis techniques essentially design a classifier based on a training set of cover-objects and stego-objects obtained from a variety of embedding algorithms. Note that none of these techniques allow for recovering the hidden images automatically. In this paper, we are aiming to develop tools which can automatically recover hidden images in LSB steganography scheme.

Kharrazi et al [6] do a comprehensive study of available stegalyzers and study their performance with varying image parameters like size, JPEG compression factors, compression artifacts etc. We are more interested in correlating some image statistics of cover or hidden images with the ability to detect steganography.

3 Database Creation

We have selected 19 images that we use as both cover images and hidden images. These are shown in Appendix 8.1.

3.1 Generating Steganographic images

We could not find any open source tool that could be used for hiding images inside other images using LSB steganography. We have written an application that takes as input a cover image, an image to be hidden (smaller in size than the cover image) and the number of lower order bits to be used for hiding data. The depth of the image to be hidden is reduced depending on the number of bits to be used for hiding data. For example, if we specify that 3 lower order bits are to be used, the depth of the image to be hidden is reduced to 3 bits per pixel per channel. In strict sense, LSB steganography means that only the last bit is to be altered, however we allow multiple lower bits to change (upto 4). To generate images to be hidden we reduce the height and width by a factor of 4. Since the number of lower order bits used vary from 1 to 4, our data injection rate (percentage of hidden data in the cover media) varies between 0.75% to 3%.

Our application supports two modes of hiding the images:

- **Simple Mode:** A pixel is randomly selected in the cover image and the image to be hidden is embedded in the lower bits of the cover image with the upper left corner at

the chosen pixel. For recovering the hidden image, only the location of the selected corner, number of bits used to hide the image and dimensions of the original image are required which can be communicated as a secret key for decryption. An example of hiding images using this mode is shown in figure 1

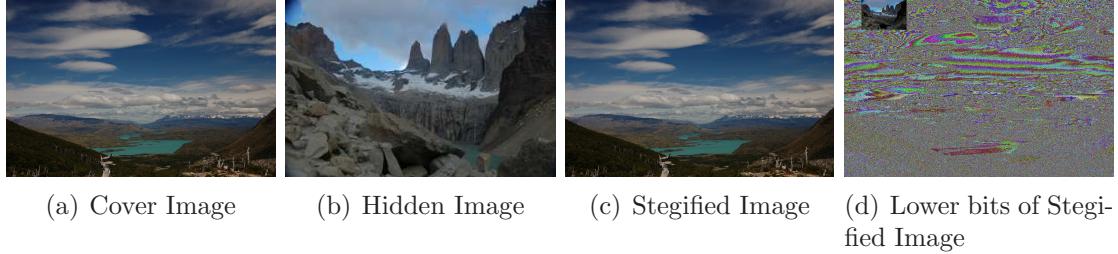


Figure 1: (a) is the cover image while (b) is an image to be hidden. (c) shows the image which contains (b) embedded within. Note that it is not perceptually different from (a). However, (d) shows the lower 3 bits of (c), which clearly reveals the hidden image

- **Shuffle Mode:** In this mode, given a cover image of size $M \times N$, and an image of size $m \times n$ to be hidden, we randomly select m out of M rows and n out of N columns of the cover image and use these selected rows and columns ($m \times n$ pixels in total) to embed the image. the decryption key in this case would consist of the indices of selected rows and columns and number of bits chosen to hide the image. An example of hiding an image using this mode is shown in figure 2.

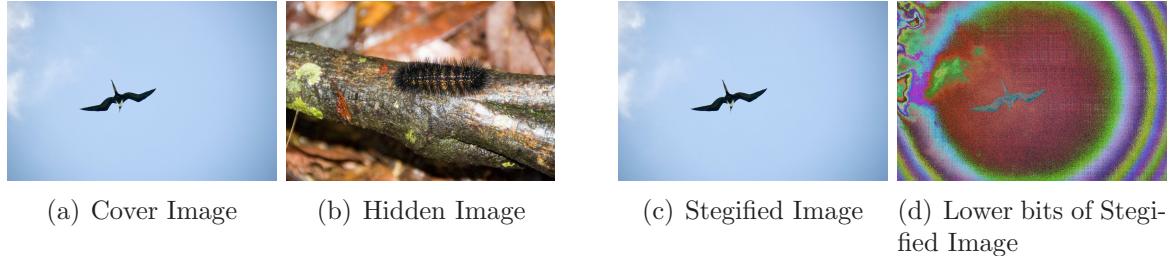


Figure 2: (a) is the cover image while (b) is an image to be hidden. (c) shows the image which contains (b) embedded within. Note that it is not perceptually different from (a). However, (d) shows the lower 4 bits of (c). It is still hard to detect the hidden image but one can see checkered pattern which corresponds to rows and columns used to hide the image

4 Detection of Simple Mode Steganography

In contrast to existing steganalysis tools, we not only want to detect the stegified images, but we want to extract the hidden content as well. This may be compared to recovering the original message from an encrypted piece of text. The high level idea of such a process is to have a brute force attack on the decryption key and check if some decryption key yields a message that can be verified using a dictionary of words. However, such an approach may not be directly applied to the domain of images since there does not exist a *dictionary* of

valid images. However, we claim that we can build such a virtual dictionary and distinguish between a real world image and arbitrary noise using some image statistics that distinguish between the two.

There has been a lot of work on identifying key image statistics that distinguish natural images. Humans are exceptionally well trained to identify these characteristics. A well known property of natural images is that when derivative like filters are applied, the distribution of the filter output is a heavy tailed Gaussian, i.e. a distribution that peaks at 0 but falls off rapidly but with significantly heavier tails than a Gaussian distribution. Such priors on image statistics have been successfully used to do various image processing operations like denoising [12], deblurring [4], super-resolution [11], etc.

In our work, rather than modeling full image statistics, we employ a very simple model where we exploit the fact that output of a gradient filter peaks at 0. This constraint can be stated in a more naive fashion by saying that adjacent pixels tend to be of the same color.

In order to decrypt the images hidden using Simple Mode, we are looking to find a decryption key that consists of the coordinates of the top left corner of the hidden image (X, Y) , dimensions of the hidden image (W, H) , and the number of bits chosen to hide the image i.e k . Hence, we are trying to figure out the 5-tuple (X, Y, W, H, k) given the stegified image. Assuming that $k \in \{1, 2, 3, 4\}$, there are only $O(n^2)$ possible values of the tuple where n is the number of pixels in the image, making a brute force search feasible. All we now need is a scoring function, that measures the likelihood of the decrypted image being a natural image based on the statistics of the decrypted image.

As stated before, we require adjacent pixels in the decrypted image to be similar. Before we give a description of our objective function, let us define some notation. I represents the input image which we are checking for steganography. $I(x, y)$ is the pixel value in I at location (x, y) . $I_m(x, y)$ denotes the lower m bits of the pixel value while $I^m(x, y)$ denotes the upper m bits of the pixel value. We define a binary function $F(x, y, k)$ over I as follows:

$$\begin{aligned} F(x, y, k) = & (I_k(x, y) == I_k(x - 1, y)) \wedge (I_k(x, y) == I_k(x, y - 1)) \\ & \wedge (I^{8-k}(x, y) \neq I^{8-k}(x - 1, y)) \wedge (I^{8-k}(x, y) \neq I^{8-k}(x, y - 1)) \end{aligned}$$

where we assume that pixel value is a 8 bit value. Observe that $F(x, y, k)$ evaluates to 1 iff the lower k bits of pixel at location (x, y) match the lower k bits of the pixel to the left and above while the upper $(8 - k)$ bits are different. Intuitively, such pixels are good candidates for containing hidden data. More discussion over the choice of this function is in the Appendix 8.2. In practice, we are dealing with 3 channel images. In such a case, we compute the function value for the three channels independently and the outputs are added to yield the final $F(x, y, k)$ value. However, we'll talk only about single channel images to keep the description simple. A sample $F(x, y, k)$ output on a stegified image is shown in figure 3.

Now we seek to build a scoring function that takes in a tuple $T = (X, Y, W, H, k)$, and evaluates the score $S(T)$ based on the decrypted image. As mentioned before, number of tuples would be $O(n^2)$ and hence our algorithm will have a complexity of $O(n^2)$ assuming $O(1)$ complexity for computing $S(T)$. We can speed things up by processing the two dimensions independently i.e for a given k we search independently over (X, W) and (Y, H) .

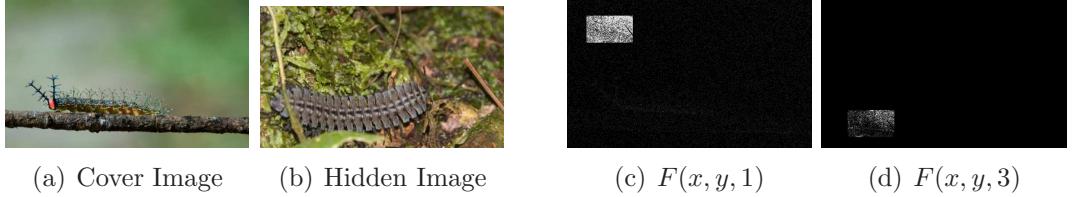


Figure 3: (a) is the cover image while (b) is an image to be hidden. The image is hidden in k lower bits and we vary k from 1 to 4. (c) and (d) show $F(x, y, k)$ values for $k = 1$ and $k = 3$ respectively. Even though it looks like that function is non zero only inside the region considering the hidden image, there is considerable clutter outside the region as well, especially for the case of $k = 1$

Number of such tuples is only $O(n)$ along each dimension. To enable such a searching, we derive two more functions from $F(x, y, k)$:

$$F_x(x, k) = \sum_y F(x, y, k)$$

and

$$F_y(y, k) = \sum_x F(x, y, k)$$

F_x and F_y are simply 1-D histograms of $F(x, y, k)$ computed along horizontal and vertical directions respectively. We make both these histograms zero mean by subtracting the mean from each bin. Now a scoring function $S_x(T')$ for evaluating a tuple $T' = (X, W)$ may be defined as:

$$S_x(T') = \sum_{x \in [X, X+W]} F_x(x, k)$$

and a similar scoring function $S_y(T')$ may also be defined to score the tuples along vertical direction.

Hence for a given value of k , we compute two tuples $T'_1 = \text{argmax}S_x(T')$ and $T'_2 = \text{argmax}S_y(T')$. The overall tuple is then simply $T = (T'_1, T'_2, k)$ with $S(T) = S_x(T'_1) + S_y(T'_2)$. We can iterate over all values of k to select the value which gives the maximal score. However, there is one final observation to be made. It is easy to see that if $F(x, y, k) = 1$, then $F(x, y, k') = 1 \quad \forall k' < k$. Hence, the scoring function is biased towards lower values of k . We fix this by changing $S(T) = 2^k * (S_x(T'_1) + S_y(T'_2))$. We put a threshold on $\frac{S(T)}{W*H}$ to decide whether an image contains a hidden image or not.

To summarize, our detection algorithm is as follows:

```

DETECT(Threshold)
  Best Score = 0;
  Best Tuple = NULL;
  FOR k = 1,2,3,4
    EVALUATE F(x,y,k)
    EVALUATE F_x(x,k) and F_y(y,k)
    FIND OPTIMAL (X,W)
    FIND OPTIMAL (Y,H)
  
```

```

EVALUATE Score of (X,Y,H,W,k) = 2^k * (S_x(X,W) + S_y(Y,H))
IF SCORE > Best Score
    Best Score = SCORE
    Best Tuple = (X,Y,H,W,k)
if Best Score > Threshold
    return image decrypted using Best Tuple
else
    "Clean Image"

```

Finding optimal (X, W) (or (Y, H)) can take $O(W'^2)$ (or $(O(H'^2))$ time where W' and H' are width and height of the original image respectively. However, it is possible to reduce the complexity to linear time using dynamic programming. Evaluating $F(x, y, k)$ is the most expensive step in this computation giving an overall time complexity of $O(n)$ where n is the number of pixels.

It's easy to see from the construction of the algorithm that it is likely to perform poorly in cases when the hidden image has a high frequency content while the cover image is a low frequency one. It becomes difficult to identify candidate pixels from the definition of $F(x, y, k)$ in such a case. More discussion on this appears in the Appendix 8.2.

5 Shuffle Mode Steganography Detection

In case of shuffle mode, the decryption key T consists of a list of rows T_r and a list of columns T_c . Again, we can check for *goodness* of decrypted image by evaluating a metric similar to $F(x, y, k)$. However, there are two key issues that need to be addressed:

- The search space of tuples in this case is large enough to make the brute force search infeasible.
- In the case of Simple Mode Detection, we could precompute $F(x, y, k)$ on the input image and then use this precomputed function to evaluate tuples. However, this is not possible in this case since the value of $F(x, y, k)$ depends upon the tuple in consideration. This is because the pixel to the left and above current pixel is defined by the set of rows and columns in the tuple itself.

To give an intuition of our approach, assume that we know what are the columns containing hidden pixels and we are supposed to find the *hidden rows* i.e the rows containing hidden pixels. Since we know the hidden columns, given a hidden pixel, we know what its predecessor is within the same row. Let the known columns be indexed using a index set C . Let us define a score $S(y, k)$ for each row (indexed by y) as:

$$S(y, k) = \sum_{c \in C} F_s(c, y, k)$$

where $F_s(c, y, k)$ is defined as:

$$F_s(c, y, k) = (I_k(C[c], y) == I_k(C[c - 1], y)) \wedge (I_{8-k}(C[c], y) \neq I_{8-k}(C[c - 1], y))$$

It is almost the same as $F(x, y, k)$ but we are only using pixel to the left of the current pixel and not the pixel above. The rows with high $S(y, k)$ are likely to contain hidden pixels. In our implementation we choose all rows with $S(y, k) > \mu + 0.1\sigma$ where the μ and σ are the mean and standard deviation of $S(y, k)$ calculated over all rows.

We can also create a similar approach to find *hidden* columns given the rows.

So given an initial set of rows (or columns), one can iteratively alternate between finding columns and rows, and hope that the algorithm converges to the correct set of rows and columns. We still have to devise a way to initialize the algorithm.

One way to initialize is to try multiple random initializations and see when does the algorithm converge to a subset of rows and columns with a good score for rows and columns. However, this might be too expensive and we choose a simpler approach in our implementation. More concretely, we consider the $F(x, y, k)$ image as calculated in the simple mode steganography detection, calculate row (column) scores using that function instead of $F_s(c, y, k)$, and initialize a set of rows (columns) using that score. Observe, that for it to work reasonably well, there should a few consecutive hidden columns (rows) which seems like a very strong assumption, especially when the hidden image is small. Hence, better initialization methods are needed. However, Figure 4 shows an example run of the algorithm and shows its capability to converge to a good solution starting from a very bad initialization.

Here also, we have the same problem that the score is biased towards choosing a lower k . However, in this case, we found out that multiplying the score by 2^k biases it towards higher bits. This could be because search space is considerably larger than the case of Simple Mode, and the method is able to lock on to some candidate solution with reasonably good score in many cases. Hence, in this case, we simply fix it by choosing the highest possible k for which we obtain a score greater than the supplied threshold. This should also work in the case of Simple Mode, but we feel that it is more elegant to have that metric in the scoring function itself if possible.

6 Experiments and results

As described in Section 3, we have a set of 361 stegified images which has been created by using 19 cover images and 19 hidden images. We run all our steganography detection experiments on a machine with 3.8GHz processor and 4GB RAM.

6.1 Detecting LSB steganography

6.1.1 Simple mode detection

We run the Simple mode LSB steganography detection algorithm on the stegified set and the set of 19 cover images. We get a 100% detection rate with no false positives or negatives. The algorithm takes around 5 seconds to run on a 6MP image. The algorithm also requires a thresholding parameter which we empirically choose as 0.2. Figure 5 shows some example of stegified images, recovered hidden images from them and ground truth hidden images. We see the color quantization effects in the recovered images because the process of LSB

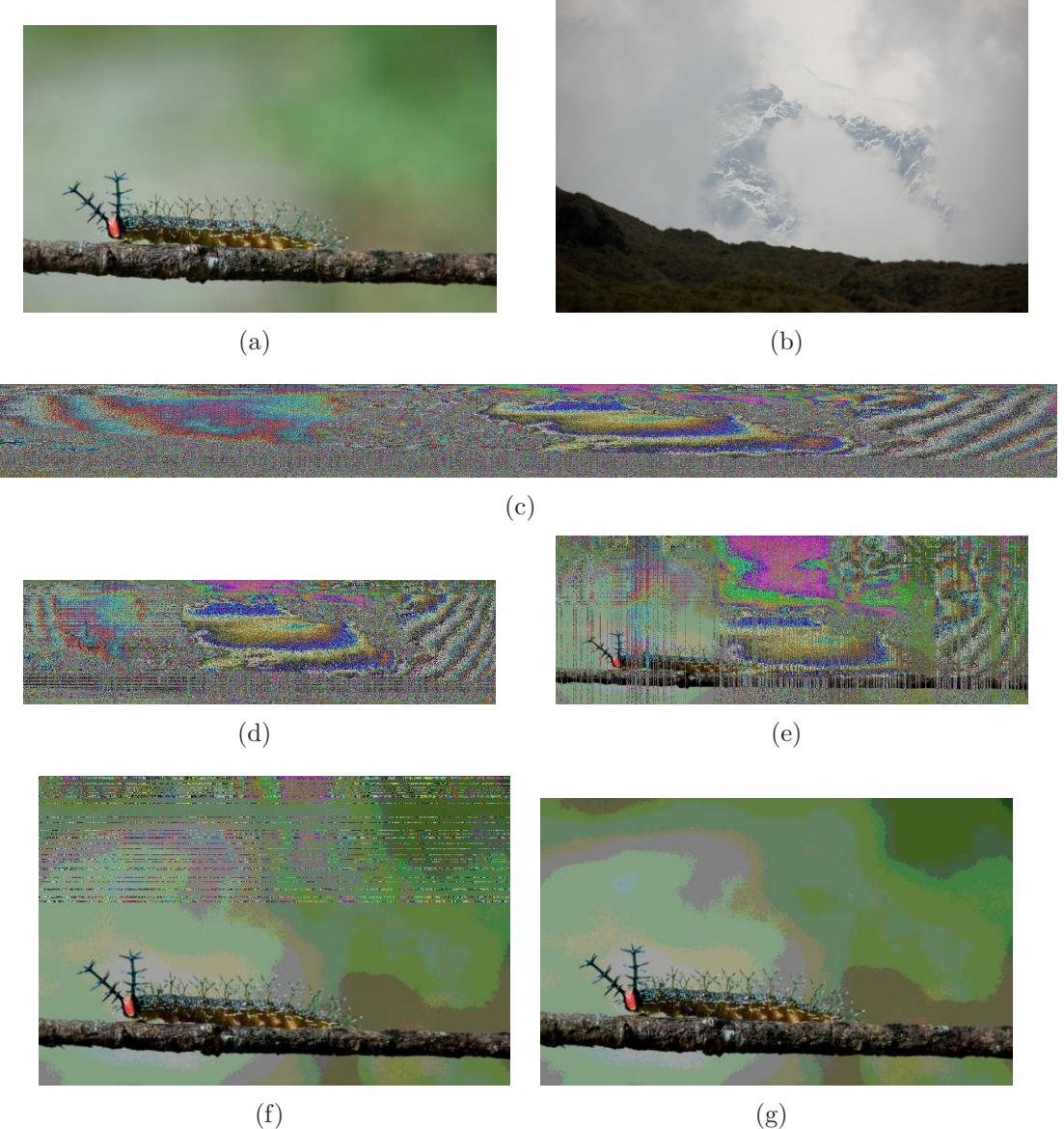


Figure 4: (a) is the hidden image while (b) is the stegified image containing the hidden image in the lower 3 bits in shuffle mode (c) shows the initial set of rows selected by the algorithm. (d) shows the selected columns given the rows selected in (c). We start to see some structure appear especially at the very top of the image but it is still dominated by a lot of spurious rows and columns. (e), (f) and (g) show subsequent iterations. It shows that the algorithm tends to converge very fast given only a few good rows in the initialization

steganography hides the higher bits of pixels from hidden image in lower bits of pixels from cover image. These results indicate that image statistics enforce a very strong prior and may be useful beyond this artificial case as well.

6.1.2 Shuffle mode detection

We run the Shuffle mode LSB steganography detection algorithm on the stegified set and the set of 19 cover images. We get a 80.57% true positive rate and 0% false positive rate.

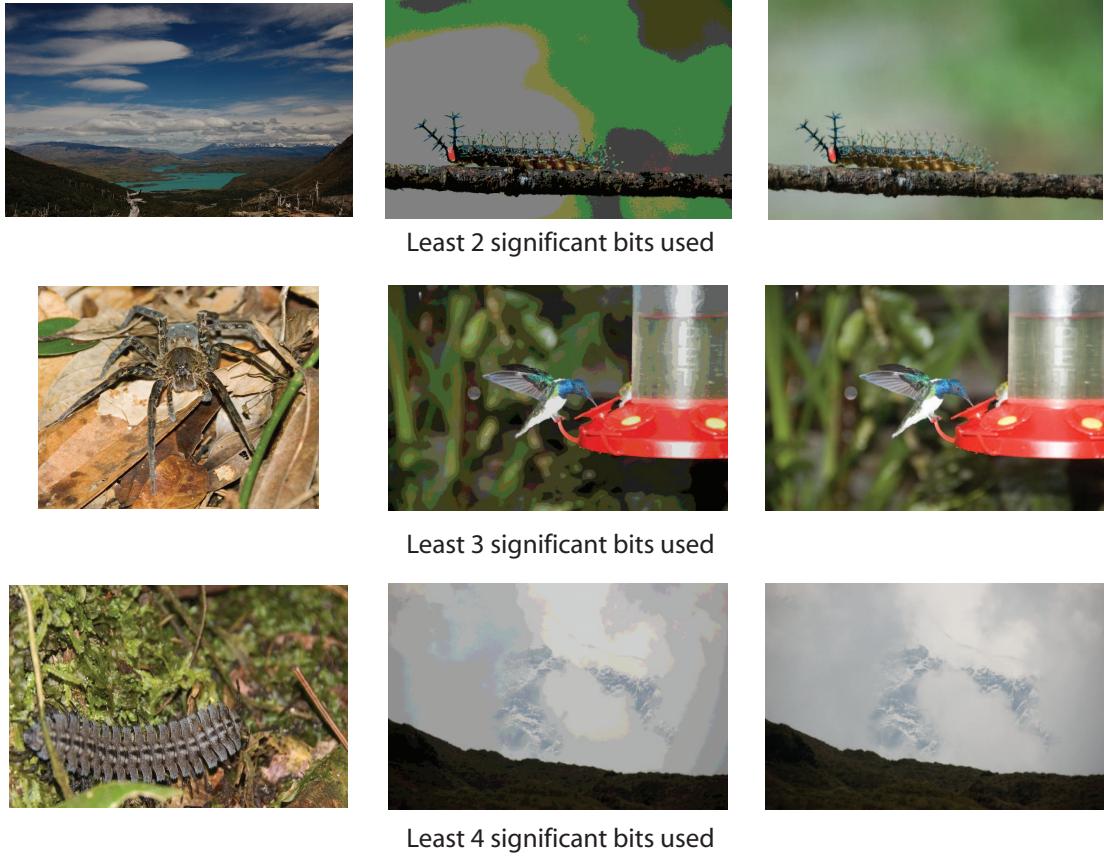


Figure 5: The left most column shows some stegified images. The middle column shows the recovered hidden images from our algorithm. The right most column shows the corresponding ground truth hidden images. Color quantization seen in the recovered images is an artifact of the hiding algorithm. The recovered quality increases as the number of bits used increases. The stegified images have been made smaller in size for better fit on page.

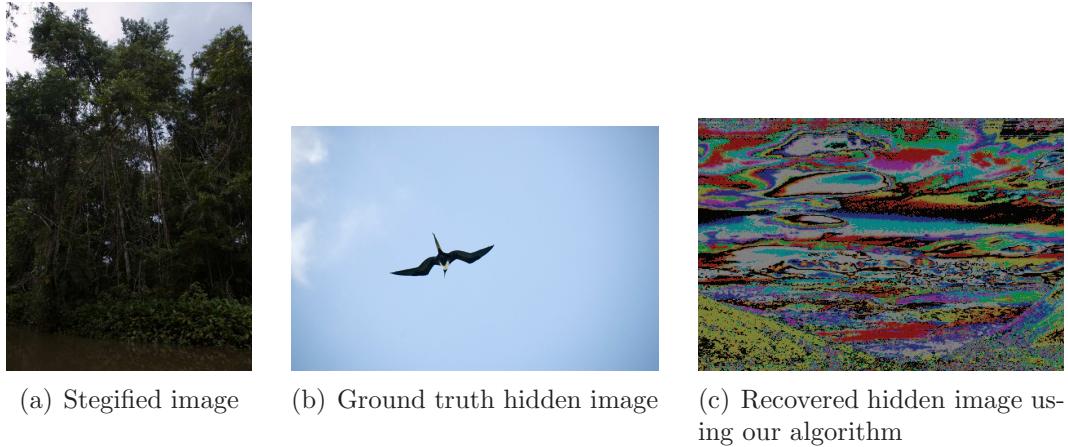


Figure 6: Failure case in shuffle mode LSB steganography detection

The algorithm takes around 15 seconds to run on a 6MP image. The algorithm also requires a thresholding parameter which we empirically choose as 0.7. The recovered images and quantization artifacts are very similar to the Simple mode and hence we are not showing the recovered images here. Figure 6 shows a failure case for Shuffle mode detection which

happens because of bad initialization as mentioned in Section 5.

6.2 Analyzing cover images for StegDetect - an existing steganography detection tool

We want to analyze if certain images are inherently good to act as cover images. This means that the detection tool is not able to detect steganography inside a cover image irrespective of the hidden image. We decided use an already available steganography detection tool StegDetect which is targeted at detecting a group of algorithms and JPHide is one of these. We first generate the stegified image database again by hiding the 19 images in one another to get 361 images. Again the hidden image versions are reduced-size versions of original (factor of 4 in each height and width). JPHide can only hide data upto a limit and hence we were not able to generate stegified images for all the combinations. Because if this, we have 285 stegified images in our test database instead of expected 361 images. Besides these, we also have the original 19 cover images which have nothing hidden inside them. We then run StegDetect on this combined test set to detect steganography. StegDetect requires a threshold parameter which was empirically chosen to be 2.0 for these experiments.

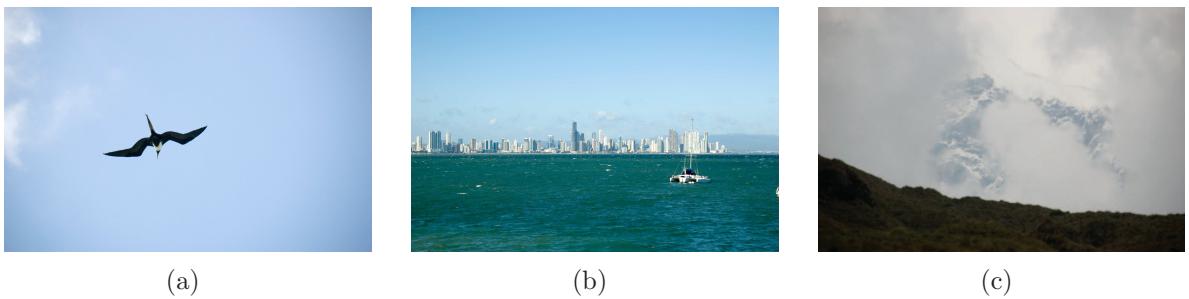


Figure 7: Cover images which have very low probability of being detected by StegDetect irrespective of the hidden image

From the detection results on the stegified images, we can compute detection probabilities conditioned on cover images or hidden images. Barring three cover images, all cover images are detected irrespective of the hidden image. Figure 7 shows these images. We hypothesize that images with large homogeneous regions and small clusters of high frequencies are good for cover images though more rigorous experimentation is required. Note again that this condition for good cover images is only applicable to the StegDetect detection tool when images are hidden using JPHide tool. We tried to quantify this observation by finding the correlation of this probability with the fraction of high frequency pixels in the image but it turns out that this image measure does not define the intuitive idea well. We have not been able to come up with a good quantitative image measure for this correlation.

Next we study the effect of varying thresholding parameter for the StegDetect tool. The tool manual says that as this threshold increases, the tool becomes more sensitive. Figure 8 shows the plotted ROC curve for different threshold values - 1.0, 2.0, 3.0 an 4.0. An ROC curve plots the true positive rate vs the false positive rate. Good performance is marked by a high true positive rate and low false positive rate. We observe that this performance does not depend much on the threshold and the value of 2 (second point from left) is good enough.

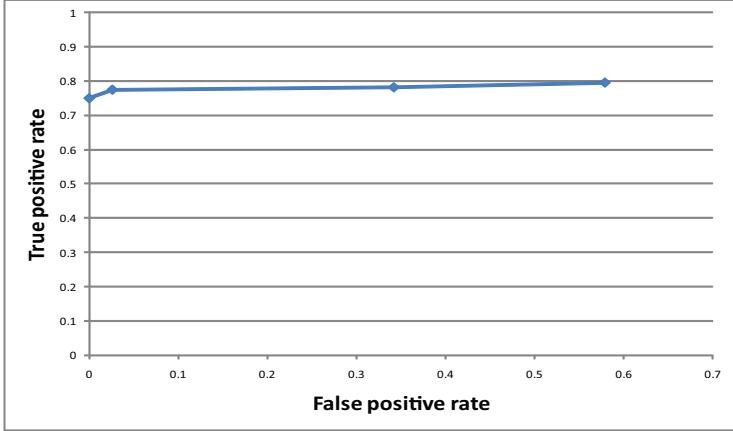


Figure 8: ROC Curve for steganography detection using the StegDetect tool. It is observed that variation in threshold value does not affect performance much.

7 Conclusion

In this paper, we presented an overview of Least Significant Bit image steganography techniques. Due to lack of available databases and tools, we build our own tools and perform a quantitative evaluation. Our tool gives good detection results on our dataset asserting that image statistics are a useful prior. We also undertook investigation of the properties of good cover images by running StegDetect detection tool on images stegified using JPHide. We found that images with small clusters of high frequency pixels perform better as cover images though more extensive experiments are required to support the claim.

Image steganography is a very wide topic in general. Besides LSB steganography techniques, there are methods that work in transform domains and also use encryption schemes. Due to lack of time and downloadable implementations, we have not been able to analyze all these techniques but doing a thorough detection analysis of all the algorithms will give useful insights about image steganography. This comprehensive analysis can also give better pointers for what type of images are good cover images. We note that if the hidden image is encrypted before, then we cannot use image priors *directly* for steganography detection, but they should still prove useful coupled with other methods.

Steganography in itself is a very wide topic with techniques to hide and recover video, audio and text signals in various media. We have only covered some aspects of image-based steganography in this paper and have shown that image statistics can be exploited to aid decryption.

References

- [1] Ismail Avcibaş, Mehdi Kharrazi, Nasir Memon, and Bülent Sankur. Image steganalysis with binary similarity measures. *EURASIP J. Appl. Signal Process.*, 2005(1):2749–2757, 2005.
- [2] Ismail Avcibas, Nasir Memon, and Blent Sankur. Steganalysis using image quality metrics. *IEEE transactions on Image Processing*, 12:221–229, 2001.

- [3] R. Chandramouli, M. Kharrazi, and N. Memon. Image steganography and steganalysis: Concepts and practice. pages 35–49, 2003.
- [4] Rob Fergus, Barun Singh, Aaron Hertzmann, Sam T. Roweis, and William T. Freeman. Removing camera shake from a single photograph. *ACM Trans. Graph.*, 25(3):787–794, 2006.
- [5] Mehdi Kharrazi, Husrev T. Sencar, and Nasir Memon. Cover selection for steganographic embedding. *IEEE International Conference on Image Processing*, 2006.
- [6] Mehdi Kharrazi, Husrev T. Sencar, and Nasir Memon. Performance study of common image steganography and steganalysis techniques. *Journal of Electronic Imaging*, 15(4), 2006.
- [7] Niels Provos. Defending against statistical steganalysis. In *10th USENIX Security Symposium*, pages 323–335, 2001.
- [8] Niels Provos and Peter Honeyman. Detecting steganographic content on the internet. Technical report, In ISOC NDSS02, 2001.
- [9] Niels Provos and Peter Honeyman. Hide and seek: An introduction to steganography. *IEEE Security and Privacy*, 1(3):32–44, 2003.
- [10] P. Sallee. Model-based methods for steganography and steganalysis. 5(1):167–189, January 2005.
- [11] J. Sun, Z.B. Xu, and H.Y. Shum. Image super-resolution using gradient profile prior. In *Computer Vision and Pattern Recognition, 2008. CVPR '08. IEEE Conference on*, pages 1–8, 2008.
- [12] Y. Weiss and W. T. Freeman. What makes a good model of natural images? In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 2007.
- [13] Andreas Westfeld. F5-a steganographic algorithm. In *Information Hiding*, pages 289–302, 2001.

8 Appendix

8.1 Database

Figure 9 shows the images in our database.

8.2 Choice of $F(x, y, k)$

As described in section 4, our algorithm is based upon $F(x, y, k)$ defined as:

$$\begin{aligned} F(x, y, k) = & (I_k(x, y) == I_k(x - 1, y)) \wedge (I_k(x, y) == I_k(x, y - 1)) \\ & \wedge (I^{8-k}(x, y) \neq I^{8-k}(x - 1, y)) \wedge (I^{8-k}(x, y) \neq I^{8-k}(x, y - 1)) \end{aligned}$$

An argument against the choice of such a definition could be that it relies on the assumption that there will be some pixels of the cover image with higher order bits different, and a more tolerant definition of it could be

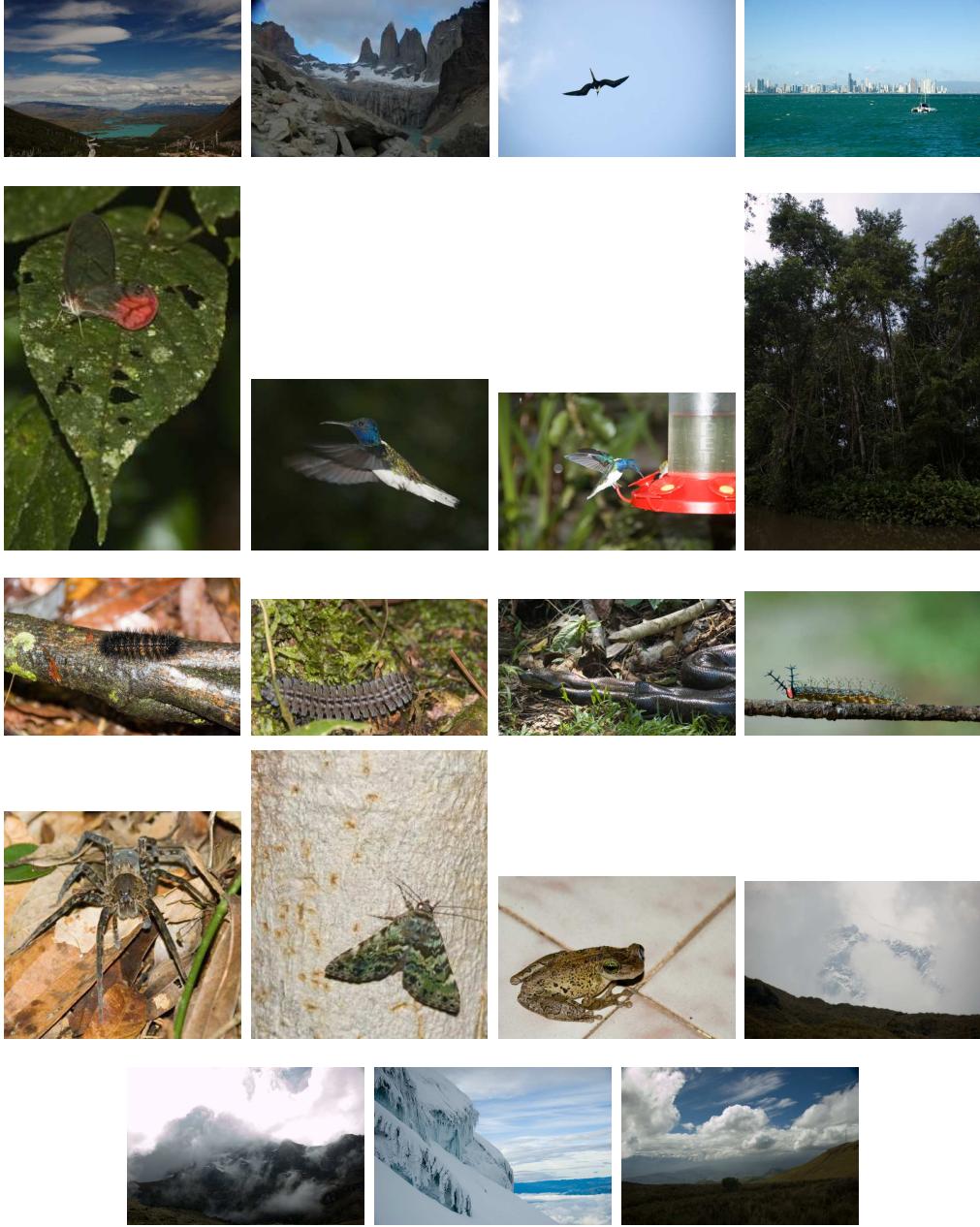


Figure 9: Set of chosen 19 images

$$F(x, y, k) = (I_k(x, y) == I_k(x - 1, y)) \wedge (I_k(x, y) == I_k(x, y - 1))$$

The above equation captures our intuition that adjacent pixels tend to be same in images. However, this is true for both the cover image and the hidden image and the second definition does not exactly capture that and may mislabel benign pixels as hidden pixels. Though this is somewhat offset by the fact that in order for this to happen, the lower order bits of those pixels in the cover image need to be same which is less likely. However, Figure 10 compares the output of the two definitions. Note that using the second definition, other candidate regions pop up even though the correct hidden region is still the dominant one. This is even more likely to happen in case of man made scenes, which will have very smooth homogeneous regions. Our database lacks such images. Using the first definition, the candidate region clearly stands out. The first definition

tends to be biased towards not marking the pixels as hidden. Such a bias is even more important in case of shuffle mode detection as we do not want our algorithm to drift from the solution.

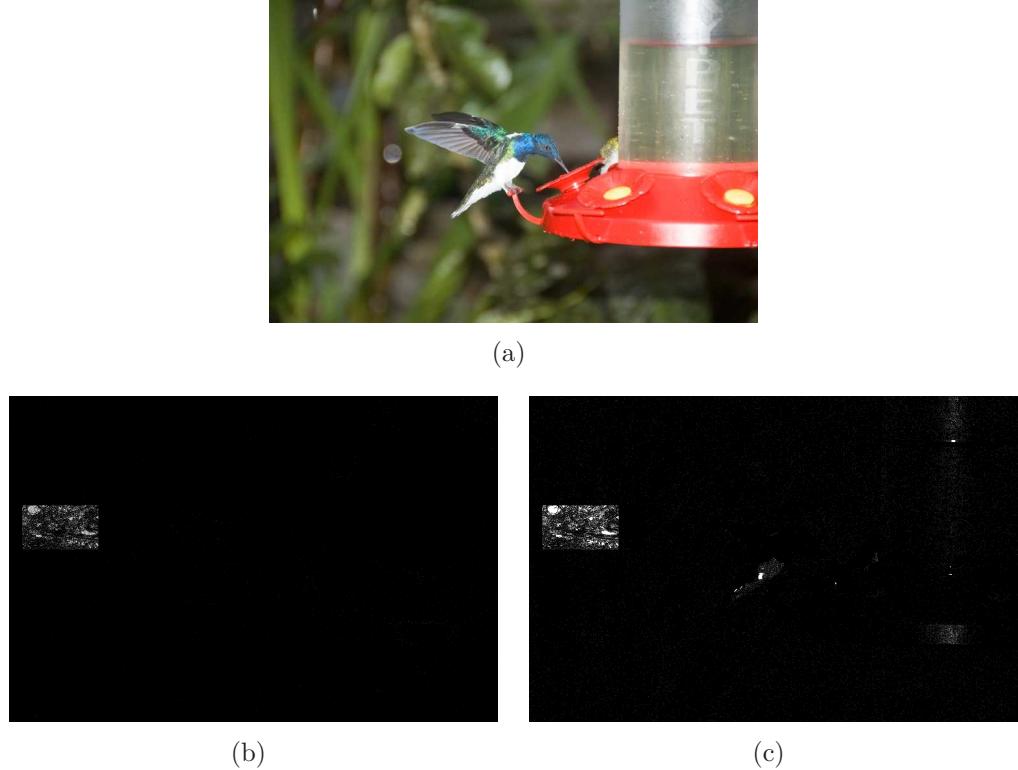


Figure 10: (a) shows a stegified image containing a hidden image in simple mode. (b) and (c) show output of $F(x, y, k)$ using correct definition of k using the first and the second definition of $F(x, y, k)$ respectively.

Our empirical testing shows that both the definitions work well while detecting simple mode but the first definition seems to fare better in case of shuffle mode. Of course, the first definition will fail, say in case of a completely white cover image. But such images are also more susceptible to visual attacks i.e. the injected data is more likely to create visual artifacts.