Abstract—Steganography is the art of secret communication between two parties that not only conceals the contents of a message, but also its existence. Steganalysis attempts to detect the existence of embedded data in a steganographically altered cover file. Many algorithms have been proposed, but so far each has had some weakness that has allowed its effects to be detected, usually through first or second order statistical analysis of the image. Our algorithm, named J4, is a new steganographic technique that improves on our previous system, J3. J4 uses dual histogram restoration along with matrix encoding to minimize detection by first as well as second order steganalysis, whereas J3 only restored the global histogram. J4 embeds data efficiently by theoretical estimation of the optimal value of matrix encoding and payload for each coefficient pair, which is also done by the receiver, so that the stopping points do not need to be included as they were in J3.

Existing blind steganalysis tests performed on J4 show that it has a detection rate of 50-60% at 0.05-0.1 bit per non-zero coefficient (bpnc), as compared to 80-90% for F5, OutGuess, Steghide, MB1, MB2 and PQ. Its performance is comparable to the best known algorithm, nsF5, slightly outperforming it in some cases. Since 50% detection rate is classified as random guess using an SVM classifier, J4 is an ideal candidate for a steganographic algorithm.

Index Terms—Steganography, J4, Dual-Histogram Compensation, Information Hiding, JPEG Steganography, Steganalysis, Matrix Encoding, Statistical Restoration.

I. INTRODUCTION

Steganography is a technique to hide data inside a cover medium in a way that the existence of any communication itself is undetectable as opposed to cryptography where the existence of secret communication is known but is indecipherable. Steganography has an edge over cryptography because it does not attract any public attention, and the data may be encrypted before being embedded in the cover medium. Hence, it incorporates cryptography with an added benefit of undetectable communication. Image files are the most common cover medium used for steganography. With resolution in most cases higher than human perception, data can be hidden in the "noisy" bits or pixels of the image file. Because of the noise, a slight change in the those bits is imperceptible to the human eye, although it might be detected using statistical methods (i.e., steganalysis).

Support Vector Machines (SVMs) have recently become popular to classify if a given image is stego or a cover [1]. The training data set consists of a number of features extracted from a set of cover and stego images. Based on this training data, SVM can build a prediction model that can classify the images. These features are based on the statistical properties of the JPEG coefficients, since these statistical correlations are violated when DCT coefficients are modified during the embedding process. These statistical properties can be DCT features [2] or Markov features [3]. A more effective approach to steganalysis was achieved by combining, calibrating, and extending the DCT and Markov features together to produce a 274 merged feature set [4]. Results show that this method produces a better detection rate than using the DCT features or the Markov features alone. Our steganalysis experiment (discussed in section IX) uses this feature extractor with an SVM classifier.

The DCT features as proposed by Fridrich et al. [2], are very effective in attacking any known steganographic system. Their algorithm is based on extraction of a number of DCT features from the given image set to detect embedded data using a classifier. In our previous work, J3 [5], we embed data by compensating for global histogram changes. We extend the idea of J3 into J4, which not only compensates for the global histogram but also for dual histograms, preserving the individual DCT modes. J4 conceals data in a way that it completely preserves the first order statistical properties [6] of the image and hence, is resistant to chi-square attacks [7] and most feature-based steganalysis techniques. Moreover, in contrast to J3, we no longer embed any stop point information inside the header data (note that this is in the structure of the embedded data and not the JPEG header). Instead, the sender estimates the embedding capacity of each coefficient pair and stops embedding any data once the estimated capacity is reached. The capacity estimation is done in such a way that there are enough coefficients left in each pair to compensate for any changes and restore the global as well as dual histogram after embedding. Since the receiver has exactly the same inputs on which to base its estimation, it arrives at the same stop points as the sender, so these need not be sent explicitly.

In contrast to most approaches, including J3, J4 also embeds data in zero coefficients for lower coefficient indices. We consider zero coefficients since the number of zeros is extremely large as compared to other coefficients. Hence, to take advantage of the matrix embedding and minimize changes, we embed data in some zero coefficients. Note that only a small number of zero coefficients will be changed since the efficiency (bits embedded per coefficient change) increases due to a large number of available coefficients. Zero coefficients are changed in a way that doesn’t affect the histogram and the shape is retained. Matrix encoding, proposed by Crandall [8], can embed k bits of message in $2^k - 1$ cover bits by changing at most 1 bit. Hence, this encoding method is very efficient when the message length is shorter than the maximum embedding capacity. F5 was the first steganography algorithm to use matrix encoding.

We compared our results with F5, nsF5 (no shrinkage F5 with wet paper codes) [9], Steghide, OutGuess [10], MB1 (model-based without blockiness) [11], MB2 (model-based with blockiness), PQ (Perturbed Quantization) [12], PQT (Texture based PQ) and PQE (Energy-based PQ). Based on 3000 sample JPEG We compared our results with F5, nsF5 (no shrinkage F5 with wet paper codes) [9], Steghide, OutGuess [10], MB1 (model-based without blockiness) [11], MB2 (model-based with blockiness), PQ (Perturbed Quantization) [12], PQ (Texture based PQ) and PQE (Energy-based PQ). Based on 3000 sample JPEG images, our SVM-based steganalysis experiments show that J4 has a much lower detection rate than these algorithms except nsF5 which performs slightly better at 0.05 bits.
per non zero coefficient \((bpnz)\).

The rest of the paper is organized as follows. In Section II, we provide some background information on JPEG steganography and dual histograms. Section III deals with some of the previous work done in image steganography. In Section IV and V, we discuss our proposed J4 embedding and extraction algorithm in detail. Section VII deals with the theoretical estimation of the optimal value of \(k\) for matrix encoding along with embedding capacity of each coefficient pair. Section VIII shows the performance of J4 in terms of matrix embedding efficiency and embedding capacity at different \(bpnz\). Section IX compares the steganalysis results of J4 with other popular algorithms mentioned pervasively. Finally, section X concludes the paper with reference to future work in this area.

II. BACKGROUND

A. JPEG Steganography

There are two broad categories of image-based steganography that exist today: frequency domain and spatial domain steganography. The first digital image steganography was done in the spatial domain using LSB coding (replacing the least significant bit or bits with embedded data bits). Since JPEG transforms spatial data into the frequency domain where it then employs lossy compression, embedding data in the spatial domain before JPEG compression is likely to introduce too much noise and result in too many errors during decoding of the embedded data when it is returned to the spatial domain. These would be hard to correct using error correction coding. Hence, it was thought that steganography would not be possible with JPEG images because of its lossy characteristics. However, JPEG encoding is divided into lossy and lossless stages. DCT transformation to the frequency domain and quantization stages are lossy, whereas entropy encoding of the quantized DCT coefficients (which we will call JPEG coefficients to distinguish them from the raw frequency domain coefficients) is lossless compression. Taking advantage of this, researchers have embedded data bits inside the JPEG coefficients before the entropy coding stage.

The most commonly used method to embed a bit is LSB embedding, where the least significant bit of a JPEG coefficient is modified in order to embed one bit of message. Once the required message bits have been embedded, the modified coefficients are re-encoded using entropy encoding to finally produce the JPEG stego image. During the extraction process, the JPEG file is entropy decoded to obtain the JPEG coefficients, from which the message bits are extracted from the LSB of each coefficient.

B. Dual Histogram and Individual DCT mode

Fridrich et. al [2] proposed a steganalysis technique based on the DCT properties of the JPEG images. This approach outperformed existing techniques by a huge margin. Several of these features were based on the global histogram, dual histograms, and the individual DCT modes. The individual DCT mode (also called individual histogram) is the occurrence of total number of each coefficient in a particular row and column of the 8x8 DCT matrix. For example, an individual histogram for \((1,1)\) would be the occurrence of all coefficients at \((1,1)\) location across all DCT blocks of the given JPEG images. Dual histograms are preserved automatically if an algorithm preserves all the individual histograms. In J4, we preserve all the individual histograms for all the coefficients for each index position except \(-1, 0\) and \(1\). Let \(d(h)\) denote the quantized DCT coefficient for \((i,j)\) coefficient in the \(k_{th}\) block. The individual histogram for \((i,j)\) is defined as:

\[
H^{i,j} = \{h^i_{w_x},...,h^i_{w_y}\}
\]

where \(h^i_{w}\) is the frequency of occurrence of coefficient \(x\) at location \((i,j)\) in all the DCT blocks. To reduce complexity of the SVM, feature selection usually truncates the range \((-\infty,\infty)\) to \((-5,5)\) since the frequency of higher order coefficients is low, and limits \(i = J < 6\) since most of the non-zero coefficients are in this range.

C. LSB-based Embedding Technique

LSB embedding (see sources [13], [14], [15]) is the most common technique to embed message bits DCT coefficients. This method has also been used in the spatial domain where the least significant bit value of a pixel is changed to insert a zero or a one. A simple example would be to associate an even coefficient with a zero bit and an odd one with a one bit value. In order to embed a message bit in a pixel or a DCT coefficient, the sender increases or decreases the value of the coefficient/pixel to embed a zero or a one. The receiver then extracts the hidden message bits by reading the coefficients in the same sequence and decoding them in accordance with the encoding technique performed on it. The advantage of LSB embedding is that it has good embedding capacity and the change is usually visually undetectable to the human eye. If all the coefficients are used, it can provide a capacity of almost one bit per coefficient using the frequency domain technique. On the other hand, it can provide a greater capacity for the spatial domain embedding with almost 1 bit per pixel for each color component. However, sending a raw image such as a Bitmap (BMP) to the receiver would create suspicion in and of itself, unless the image file is very small. Fridrich et. al. proposed a steganalysis method which provides a high detection rate for shorter hidden messages [16]. Westfeld and Pfitzmann proposed another steganalysis algorithm for BMP images where the message length is comparable to the pixel count [7]. Most of the popular formats today are compressed in the frequency domain and therefore it is not a common practice to embed bits directly in the spatial domain. Hence, frequency domain embedding is the preferred choice for image steganography.

III. PREVIOUS WORK

Jsteg [17] was one of the first JPEG steganography algorithms. It was developed by Derek Upham, and embeds message bits in LSB of the JPEG coefficients. JP Hide&Seek [18] is another JPEG steganography program, improving stealth by using the Blowfish encryption algorithm to randomize the index for storing the message bits. This ensures that the changes are not concentrated in any particular portion of the image, a deficiency that made Jsteg more easily detectable. However, both of these algorithms are easily detected by the chi-square attack [7] since they equalize pairs of coefficients in a typical histogram of the image, giving a “staircase” appearance to the histogram. F5 [19] is one of the most popular algorithms, and is undetectable using the chi-square technique. F5 uses matrix encoding along with permutating straddling to encode message bits. It also avoids making changes to any DC coefficients and coefficients with zero value. If the value of the message bit does not match the LSB of the coefficient, the coefficient’s value is always decremented, so that the overall shape of the histogram is retained. However, a one can change to a zero and hence the same message bit must be embedded in the subsequent coefficients until its value becomes non-zero, since zero coefficients are ignored on decoding. However, this technique modifies the histogram of JPEG coefficients in a predictable manner. This is because of the shrinkage of ones converted to zeros
increases the number of zeros while decreasing the histogram of other coefficients and hence can be detected once an estimate of the original histogram is obtained [20].

Another popular algorithm is Steghide [21], which uses graph theory techniques to preserve the histogram. Two inter-changeable coefficients are connected by an edge in the graph with coefficients as vertices of the graph. The message is then embedded by swapping the two coefficients connected in the graph. Since the coefficients are swapped instead of replacing LSBs, it is difficult to detect any distortion using first order statistical analysis. But the steganalysis results show that it has a very high detection rate as compared to J4 even at lower embedding rates.

Another technique of steganography, proposed by Marvel et al., [22] uses spread spectrum techniques to embed data in the cover file. The idea is to embed secret data inside a noise signal which is then combined with the cover signal using a modulation scheme. Every image has some noise in it because of the image acquisition device and hence this property can be exploited to embed data inside the cover image. If the noise being added is kept at a low level, it will be difficult to detect the existence of message inside the cover signal. To make the detection hard, the noise signal is spread across a wider spectrum. At the decoder side, image restoration techniques are applied to guess the original image which is then compared with the stego image to estimate the embedded signal. Several other data hiding schemes using spread spectrum have been presented by Smith and Comiskey in [23]. Steganalysis techniques to detect spread spectrum steganography have been shown in [24], [25], where the authors claim to detect 70% of the embedded message bits and 95% of the images respectively.

A. Statistical Restoration Techniques

Statistical Restoration refers to the a class of embedding data such that the first and/or higher order statistics are preserved after the embedding process. As mentioned earlier, embedding data in a JPEG image can lead to change in the typical statistics of the image which in turn can be detected by steganalysis. Most of the steganalysis methods existing today employ first and second order statistical properties of the image to detect any anomaly in the stego image. Statistical restoration is done to restore the statistics of the image as close as possible to the given cover image. Our algorithm, J4, falls under the category of first order statistical restoration or preservation schemes [10], [21], [26], [6], [27].

OutGuess, proposed by Niels Provos, was one of the first algorithms to use statistical restoration methods to counter chi-square attacks [10]. The algorithm works in two phases, the embed phase and the restoration phase. After the embedding phase, using a random walk, the algorithm makes corrections to the unvisited coefficients to match it to the cover histogram. OutGuess does not make any change to coefficients with 1 or 0 value and uses a error threshold to determine the amount of change which can be tolerated in the stego histogram. This means that that algorithm may not be able to restore the histogram completely to the cover image. If the error threshold is too small, the data capacity can reduce drastically since there will be too many unused coefficients. Also, the fraction of coefficients used to hold the message, α, is inversely proportional to the total number of coefficients in the image. This means OutGuess will perform poorly when the number of available coefficients is too large.

Another statistical restoration technique is presented by Solanki et al [26] where authors claim to achieve zero K-L divergence between the cover and the stego images using their method while hiding at high rates. The probability density function (pdf) of the stego signal exactly matches the cover signal. They divide the file into two separate parts, one used to hiding and the other for compensation. The goal is to match the continuous pdf of the cover signal to the stego signal. They used a magnitude based threshold where they avoid hiding any data in symbols whose magnitude is greater than T. For JPEG images, they use 25% of the coefficients for hiding while preserving the rest for compensation. This approach is not very efficient because it does not use all the potential coefficients for hiding data. The coefficients in the compensation stream are modified using minimum mean-squared error criteria [26]. However, they do not consider the intra and inter block dependency amongst JPEG blocks which are important tools used by steganalysis to detect stego images.

Another higher order statistical restoration technique has been presented by the same authors [28] where they use the earth-mover’s distance (EMD) technique to restore the second order statistics. EMD is a popular distance metric used in computer vision application. The cover and the stego images have different PMF’s. The EMD is defined as the minimum work done to convert the host signal to the stego signal. The authors have considered the concept of bins where each bin stored a horizontal transition from one coefficient to another. Each block is stored in 1-D vector in zigzag scanning order. Hence, we have 64 columns and Nr rows where Nr is equal to the total number of blocks in the image. This 2-D matrix can help capture both inter as well as intra block dependencies. The transitions are stored in bins. If any of the coefficients is modified, one of the bins maybe modified depending on change. Depending on the change, they try to find an optimal location to compensate that change in the bins so that the bin counts remain as in the cover image. However, the authors have only considered the horizontal transitions probability in both inter/intra block dependency. They have not considered the diagonal and the vertical transitions which are also an important factor to restore the second order statistics. Moreover, they do not provide any concrete steganalysis results to prove the effectiveness of their method.

IV. J4 Embedding Module

![Fig. 1: Block diagram of J4 embedding module.](image-url)
As discussed before, J4 tries to restore the individual histograms as well as the global histogram. Restoration of the individual histograms would also result in the restoration of the global histogram. However, J4 doesn’t restore the individual and global histogram of 1, -1, and 0 coefficients. This is to reduce the overall number of changes. To minimize impact, the ratio of 1’s to -1’s is always maintained throughout the embedding process. The embedding process of J4 goes through a number of steps as discussed below. Figure 1 shows the embedding module of J4.

A. Pre-processing Stage:

Preprocessing estimates embedding capacity for each coefficient pair. In order to simplify the algorithm, the DCT coefficients of an 8x8 matrix are considered as a one dimensional array where the coefficients are arranged in zigzag order. The right half of the coefficients in the array are high frequency components, and are mostly zeros after quantization. Therefore, we only consider the first 28 elements of the array for embedding and compensation since these hold most of the non-zero coefficients. The first element, that is, the DC coefficient is ignored. The frequency count of each coefficient is calculated for each index position in the array over all the DCT blocks. This arrangement can be viewed as a two dimensional bin or a matrix, where the row signifies the index in the DCT array and the column signifies the coefficient value. Hist\(i\) represents the total occurrences of coefficient value \(y\) at position \(i\) in each DCT array of a JPEG image. Once the individual histograms are calculated, capacity of each of the pairs in the bins is estimated theoretically (Section VII). For example, we estimate the capacity of pair \((2y, 2y + 1)\) for each position \(i\) in the array, also represented as \(Est\(i\)(2y, 2y + 1)\). The estimated capacity is kept in separate 2-D bins. We also have an additional 2-D bin that keeps track of the current number of coefficients that have been used from a given pair, denoted by \(Used\(i\)(2y, 2y + 1)\), where \((2y, 2y + 1)\) form a coefficient pair and \(i\) is the coefficient index.

B. Header Information:

Once the preprocessing is over, we determine the appropriate values of \(n\) and \(k\) for matrix encoding. The matrix encoding is denoted by \((1, n, k)\), which means \(k\) bits of message can be embedded in \(n\) coefficients by changing at most 1 coefficient out of \(2^n - 1\). The ratio \(n/k\) can be obtained by summing the total estimated capacity in bits divided by the total message bits. The receiver needs to know \(k\) in order to properly decode the message, so it is stored in the header bits of the message. The header also stores useful information such as the data length, coefficient threshold, and DCT array index limit. The header information itself is not matrix embedded since the receiver needs to decode the header bits to know the embedding rate, \(k\). The structure of the header is given in Table I.

<table>
<thead>
<tr>
<th>Value of matrix encoding, (k)</th>
<th>Data length in bytes, (M_L)</th>
<th>Coefficient threshold, (\text{Coeff}<em>{f</em>{th}})</th>
<th>DCT index threshold, (\text{Index}<em>{f</em>{th}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Bits</td>
<td>20 Bits</td>
<td>4 Bits</td>
<td>2 Bits</td>
</tr>
</tbody>
</table>

TABLE I: Header structure for J4 algorithm

Explanation of Header fields:
- \(k\) = value of \(k\) in matrix encoding \((1, 2^k - 1, k)\).
- \(M_L\) = total message length in bytes, not including the length of header.

- \(\text{Coeff}_{f_{th}}\) = absolute value of upper coefficient limit for DCT coefficients. Any DCT coefficient value higher than this will be ignored during the embedding process.
- \(\text{Index}_{f_{th}}\) = represents the boundary index in the 1-D 8 x 8 zigzag array after which we ignore all the coefficients. The actual index position is encoded using two bits, where \(00 = 28, 01 = 56, 10 = 43, 11 = 54\). J4 by default uses value 00, i.e. 28 coefficients in the array for embedding, since half of the values in the JPEG coefficients are zero in a typical image.

C. Embedding Stage:

The cover image is first entropy decoded to obtain the JPEG coefficients. The message to be embedded can be encrypted using AES or DES. A pseudo-random number generator is used to visit the coefficients in random order to embed the encrypted message, seeded using shared password \(P\). The algorithm always makes changes to the coefficients in a pairwise fashion; e.g. \((2, 3)\) corresponds to a pair. A JPEG coefficient with a value of \(2x\) will only change to \(2x + 1\) to encode message bit 1, and a coefficient with a value \(2x + 1\) will only change to \(2x\) to encode message bit 0.

In contrast to the general trend of not changing any zero valued coefficients, J4 makes changes to the zero coefficients. This is done in order to leverage the potential of extremely high number of zero coefficients in a typical JPEG image. Since we use matrix encoding, using the zero coefficients would result in a overall reduction in the number of coefficient changes and hence reduce detection. This heuristic is proven by the extremely low detection rate which has been discussed in the steganalysis section. Coefficients with value 0, 1 and -1 have a different embedding strategy since their frequency is very high as compared to other coefficients and form a triplet, \((-1, 0, 1)\). A -1 or 1 coefficient is equivalent to message bit 1 and 0 is equivalent to message bit 0. To encode message bit 0 in a coefficient with value \(1\) or \(-1\), we always change its value to 0. Similarly, to encode bit 1 in 0 coefficient, we change it to either a 1 or a -1. Change of a 0 to a 1 or -1 would depend on the total number of changes made to 1 and -1 before. The changes are done in such a way that the approximate ratio of -1 and 1 is maintained as in the cover file to thwart chi-square attacks. The algorithm keeps track of the imbalance in 1 and -1. If the imbalance in -1 exceeds that of 1, a zero would be changed to a -1 instead of 1 and vice-versa. Hence, the overall shape of the histogram is retained as in the cover image.

The embedding coefficient pairs are \((-2n, -2n - 1) \cdots (-2, -3), (-1, 0, 1), (2, 3) \cdots (2n, 2n + 1)\), where \(2n + 1\) and \(-2n - 1\) are the threshold limits for positive and negative coefficients, respectively, i.e. \(|-2n - 2n - 1| = |2n, 2n + 1| = \text{Coeff}_{f_{th}}\). During the embedding process, if the number of bits encoded for a particular pair, \((x, y)\), equals the estimated value, \(Est\(i\)(x, y)\), we stop considering pair \((x, y)\) for matrix embedding. Here \((x, y)\) denotes coefficients \(x\) and \(y\) at index \(i\) in the 1-D DCT array. Unused coefficients for that pair will be used later to compensate for the imbalance. Theoretical estimation ensures that enough coefficients are left untouched in order to restore the global and dual histogram after the embedding process. Note that full coefficient restoration is crucial as the receiver must calculate the same estimated capacity in order to decode the data properly and to know when to stop. Since the estimated capacity depends on the total number of each coefficient, we need to restore each coefficient pair for each index fully. If we cannot restore the coefficient fully, we need to make sure that estimation of the coefficient after the compensation would yield the same value as before the embedding process.

D. J4 Embedding Algorithm in Detail

Following terminology is used throughout the algorithm and this paper.
- Hist\(i(y)\) – Total number of coefficient \(y\) at index \(i\) of the DCT array initially present in the cover image. Therefore, Hist\(i(y) = \sum_{y,i} \delta_i(C_i = y)\), where \(\delta = 1\) if \(C_i = y\).
- TR\(i(y)\) – Remaining number of coefficients \(y\) at index \(i\) which are unused and untouched during embedding at index \(x\).
- TC\(i(y)\rightarrow\bar{y}\) – Total number of coefficient \(y\) changed to \(\bar{y}\) at index \(i\) during embedding.
- \(\Delta'(y)\) – Represents the unbalance in coefficient \(y\) at index \(i\) as compared to Hist\(i(y)\).
- \(N_b\) – Total number of blocks in the cover image.
- \(C_i\) – Value of coefficient at index location \(i\) in the cover image where \(1 \leq i \leq Index_{th}\).
- Coeff_{total} – total number of coefficients in the image = 64 \(\times N_b\).

**Example 1** This example demonstrates the embedding process using the above terminology. At the start of embedding process, assume the following values:

Hist\(5(2) = 500\), Hist\(5(3) = 200\), Est\(5(2,3) = 400\)

Assume the following scenario during embedding:

TC\(5(2)\rightarrow3\) = 100, TC\(5(3)\rightarrow2\) = 50, Used\(5(2,3) = 400\)

Since, Est\(5(2,3) = Used\(5(2,3)\), the algorithm would ignore any \(2,3\) pairs at index 5 for matrix encoding from this stage.

At the end of the embedding, we calculate the net imbalance in the pairs for each index. Since 100 \(2\)'s have been changed to a 3 and 50 \(3\)'s have been changed back to 2, we have an imbalance in the histogram.

\[\Delta_5(2) = TC_5(3\rightarrow2) - TC_5(2\rightarrow3) = -50\]

\[\Delta_5(3) = TC_5(2\rightarrow3) - TC_5(3\rightarrow2) = -\Delta_5(2) = 50\]

This means we have 50 more \(3\)'s at index 5 overall in DCT arrays than required and 50 fewer \(2\)'s than needed to balance the histogram pair \((2,3)\) to its original values. Hence, we need at least 50 \(3\)'s to balance the pair \((2,3)\). This compensation is done after the embedding stage by visiting 50 unused \(3\)'s in random order at index 5 and changing it to 2.

J4 embedding algorithm is divided into smaller subroutines. Assume that the cover image is decoded to obtain the quantized DCT coefficients. Each of the DCT blocks is used as is, i.e. it is not zig-zag encoded and contains 64 coefficients in row-major order. The embedding algorithm first calculates the appropriate \(n\) and \(k\) for the matrix embedding. It then determined the estimated capacity of each of the coefficient pairs using the equations given in Section VII. A procedure is then called to embed the header bits of the structured, embedded data. The value of coefficient threshold, Coeff_{th} and index threshold, Index_{th} is fixed on both sender and receiver side. We embed these just for robustness since these parameters are important in encoding and decoding the message. Another procedure returns the index of the next random unused eligible coefficient. Eligibility is determined by the index position of the coefficient and its value. If coefficient value exceeds Coeff_{th} or the position of the coefficient in the 64 element array is greater than Index_{th}, it is marked unusable. DC coefficients are also marked unusable. This procedure also keeps track of how many coefficients of each pair for every index position have been used so far. If the used coefficient count exceeds the estimated capacity for the current coefficient pair in consideration, the flag is set for that pair. The procedure will ignore all remaining coefficients of that pair during the entire embedding process once the flag is set.

Once the header bits are embedded, the main algorithm then embeds the message bits using a random traversal order which is determined by the seed generated using the shared password. In each step of the embedding, it considers \(n\) coefficients to embed \(k\) message bits. To embed the message bits using matrix embedding, a procedure is called that gives the index position of the coefficient whose LSB is to be flipped. This procedure also keeps track of the number of changes made in each coefficient pair for every index position. Once all the message bits have been embedded, a balance procedure then calculates the net change in individual coefficients pairs at each index and restores the count to its original values using the unused coefficients at those indices. Triplet (-1,0, 1) are encoded without any compensation but the ratio of -1 and 1 is maintained as mentioned earlier. This is shown in Figure 3 which shows that the percent change in -1 and 1 is almost equal.

Let \(P\) be the shared password between the sender and the receiver. This password is used to generate the seed for pseudo-random numbers between 0 and \(64 \times N_b\). The same password is also used for encrypting and decrypting the data. Let

\[\ Enc(M,P) = Encryption \ of \ message \ M \ using \ P \ as \ key \ with \ AES \ or \ DES \ standard.\]

\[\ RNG(P,x) = Unique \ random \ number \ generating \ a \ number \ between \ 0 \ and x \ using \ P \ as \ given \ seed. \ The \ uniqueness \ can \ be \ implemented \ using \ a \ bitmap \ to \ store \ the \ used \ indices.\]

\[\ M_E = Total \ number \ of \ bits \ in \ encrypted \ message, \ M_E.\]

\[\ Coef(x) = Coefficient \ value \ at \ index \ x \ in \ DCT \ block \ y.\]

\[\ Used(x) = Total \ number \ of \ x \ coefficients \ used \ irrespective \ of \ their \ index \ position.\]

\[\ Hist(x) = Total \ number \ of \ x \ coefficients \ in \ the \ image \ irrespective \ of \ their \ index \ position \ in \ the \ array.\]

**V. J4 Extraction Module**

This section deals with the extraction of message \(M\) from a given J4 stego image. The extraction algorithm is simple, as the receiver has to only calculate the estimated capacity of each of pairs using \(k\) from header and stop decoding that pair when the estimated capacity equal the number of bits used up in that pair. Password \(P\) is used to generate the random number sequence used to permute the coefficient indices.

Fig. 2: Block diagram of J4 extraction module.
for visitation order. The header is decoded first to get the values of k, message length, coefficient threshold and the index threshold.

Once all the header bits have been extracted, the extraction process starts decoding the message bits, taking care to stop extraction from a coefficient pair once its estimated capacity has reached. The decoding algorithm is given below. As explained earlier, we will only show the algorithm for positive coefficients. Similar rules apply to the negative coefficients, with slight modification. A block diagram of J4 extraction module is given in Figure 2.

VI. J4 Extraction Algorithm

The extraction algorithm is divided into two modules. The first decodes the header to recover required information for decoding the message. The second extracts the encrypted message bits, which are then decrypted to recover the actual message. These first make the same estimation of the stop points as the embedding algorithm, and like the embedding algorithm, stop when this number of coefficients from a pair have been visited for extraction. When all the bits of a matrix word have been obtained from the pairs, the error syndrome is used to extract the actual encrypted message bits. Of course, the extraction algorithm ignores all the coefficients of a pair once the threshold has been reached, including those modified by the embedding algorithm to balance the histograms.

VII. Theoretical Estimation of Embedding Capacity

This section shows how to estimate the expected embedding capacity of a cover file using J4. This is done by estimating the value of n for matrix embedding and using that to estimate the capacity of each of the coefficient pairs. Initially, we have not analyzed the coefficient pairs in terms of their index position for simplicity, but it will be included in the final estimated result. We also show the calculation for positive coefficients only. The calculation for the negative coefficients pair are similar with slight modifications.

- **coeff<sub>th</sub>** = Coefficient threshold to consider for embedding.
- **Pc<sub>2x+1</sub>** = Probability of encountering an odd number with value (2x+1) in traversing the coefficients.
- **Pc<sub>2x</sub>** = Probability of encountering an even number with value 2x in traversing the coefficients.
- **coeff<sub>total</sub>** = Total number of eligible coefficients in the input image.
- **Pr(x → y)** = Probability of coefficient x being changed to coefficient y.

\[
\text{coeff}_{total} = \sum_{x=2}^{\text{Hist}(x) - \text{coeff}_{th}}
\]

\[
Pc_{2x+1} = \frac{\text{Hist}(2x+1)}{\text{coeff}_{total}}
\]

\[
Pc_{2x} = \frac{\text{Hist}(2x)}{\text{coeff}_{total}}
\]

An odd coefficient can only decrease or retain its value to embed a data bit. Similarly, an even number can only increase or retain its value to embed a data bit, as explained in embedding module. Since, we are using matrix embedding, only at most one coefficient will change out of n in (1,n,k) matrix embedding.

\[
Pr(2x+1 \rightarrow 2x) = \frac{1}{(n+1)} Pc_{2x+1}
\]

\[
Pr(2x \rightarrow 2x+1) = \frac{1}{(n+1)} Pc_{2x}
\]

\[
Pr(2x+1 \rightarrow 2x+1) = \frac{n}{(n+1)} Pc_{2x+1}
\]

\[
Pr(2x \rightarrow 2x) = \frac{n}{(n+1)} Pc_{2x}
\]

Let \(\gamma(2x, 2x+1)\) = Total number of eligible coefficients visited so far at any instant. Although, -1, 0 and 1 are also eligible coefficients in J4, we do not consider them for calculation of \(\gamma(2x, 2x+1)\) since we are not doing any estimation or compensation on them.

Let \(TC_{Ex}(x \rightarrow y)\) be the expected number of coefficients with value x changed to y to embed a data bit.

Let \(TR_{Ex}(x)\) be the expected number of coefficients with value x remaining unchanged and unused.

\[
TC_{Ex}(2x+1 \rightarrow 2x) = \gamma(2x, 2x+1) Pr(2x+1 \rightarrow 2x)
\]

\[
TC_{Ex}(2x+1 \rightarrow 2x+1) = \gamma(2x, 2x+1) Pr(2x+1 \rightarrow 2x+1)
\]

\[
TC_{Ex}(2x \rightarrow 2x+1) = \gamma(2x, 2x+1) Pr(2x \rightarrow 2x+1)
\]

\[
TC_{Ex}(2x \rightarrow 2x) = \gamma(2x, 2x+1) Pr(2x \rightarrow 2x)
\]

\[
TR_{Ex}(2x+1) = \text{Hist}(2x+1) - \left[TC_{Ex}(2x+1 \rightarrow 2x) + TC_{Ex}(2x+1 \rightarrow 2x+1)\right]
\]

\[
TR_{Ex}(2x) = \text{Hist}(2x) - \left[TC_{Ex}(2x \rightarrow 2x+1) + TC_{Ex}(2x \rightarrow 2x)\right]
\]

A. Calculation of Stop Position for each Pair

Let \(\Delta_{Ex}(x)\) be the expected net unbalance of coefficients with value x.

Since we have estimated \(TR_{Ex}(i)\) for all the coefficients, we can now calculate the condition when we should stop embedding any data in a coefficient pair, since we will be left with only enough amount of untouched (remaining) coefficients in that pair to balance the histogram after the embedding process. The condition is:

\[
\Delta_{Ex}(2x+1) = TC_{Ex}(2x \rightarrow 2x+1) - TC_{Ex}(2x+1 \rightarrow 2x),
\]

\[
TC_{Ex}(2x \rightarrow 2x+1) \geq TC_{Ex}(2x+1 \rightarrow 2x)
\]

\[
\Delta_{Ex}(2x) = TC_{Ex}(2x+1 \rightarrow 2x) - TC_{Ex}(2x \rightarrow 2x+1),
\]

\[
TC_{Ex}(2x+1 \rightarrow 2x) \geq TC_{Ex}(2x \rightarrow 2x+1)
\]

The stop condition is:

\[
TR_{Ex}(x) = \Delta_{Ex}(x)
\]

Replacing LHS of equation 14 with RHS of equation 12, we get

\[
\text{Hist}(2x+1) - \left[TC_{Ex}(2x+1 \rightarrow 2x) + TC_{Ex}(2x+1 \rightarrow 2x+1)\right]
\]

\[
= TC_{Ex}(2x \rightarrow 2x+1) - TC_{Ex}(2x+1 \rightarrow 2x)
\]

Using equation 8, 9 and 10, we get:

\[
\text{Hist}(2x+1) - \gamma(2x, 2x+1) Pr(2x+1 \rightarrow 2x+1) =
\]

\[
\gamma(2x, 2x+1) Pr(2x \rightarrow 2x+1)
\]

Solving for \(\gamma(2x, 2x+1)\) using equation 5 and 6, we get:

\[
\gamma(2x, 2x+1) = \frac{(n+1) \text{Hist}(2x+1)}{(Pc_{2x} + n Pc_{2x+1})}
\]

Simplifying using equation 1, 2 and 3, we get:

\[
\gamma(2x, 2x+1) = \frac{(n+1) \text{Hist}(2x+1) \text{coeff}_{total}}{\text{Hist}(2x) + n \text{Hist}(2x+1)}
\]
Adding Hist coefficient pair $\gamma(2x, 2x+1)$ as:

$$\gamma(2x, 2x+1) = \frac{(n+1) \text{Hist}(2x) \text{coeff}_\text{total}}{n \text{Hist}(2x) + \text{Hist}(2x+1)} \quad (20)$$

Let equation 19 be represented as $\gamma(2x, 2x+1)_\alpha$ and equation 20 as $\gamma(2x, 2x+1)_\beta$ for convenience.

**Theorem 1** The estimated stop point for pair $(2x, 2x+1)$, $\gamma(2x, 2x+1)_{\min}$, is the minimum of $\gamma(2x, 2x+1)_\alpha$ and $\gamma(2x, 2x+1)_\beta$.

$$\gamma(2x, 2x+1)_{\min} = \min\left\{\gamma(2x, 2x+1)_\alpha, \gamma(2x, 2x+1)_\beta\right\}$$

**Proof:** Let the maximum coefficient index be represented by $\text{Index}_{\text{max}}$. The maximum index value is equal to the maximum number of eligible coefficients in the image. Hence, $\text{Index}_{\text{max}} = \text{coeff}_\text{total}$

Any stop point, $\gamma(2x, 2x+1)$ cannot exceed the total number of coefficients. Hence $\gamma(2x, 2x+1)_\alpha \leq \text{Index}_{\text{max}} \implies \gamma(2x, 2x+1)_\alpha \leq \text{coeff}_\text{total}$

Using equation 18 and 19 and substituting for $\gamma(2x, 2x+1)_\alpha$, we get

$$\frac{(n+1) \text{Hist}(2x+1) \text{coeff}_\text{total}}{n \text{Hist}(2x) + \text{Hist}(2x+1)} \leq \text{coeff}_\text{total} \quad (21)$$

Simplifying equation 21 we get

$$\text{Hist}(2x) \geq \text{Hist}(2x+1)$$

Adding $n \text{Hist}(2x)$ on both sides, we get,

$$n \text{Hist}(2x) + \text{Hist}(2x) \geq \text{Hist}(2x+1) + n \text{Hist}(2x) \quad (23)$$

$$\implies \frac{(n+1) \text{Hist}(2x)}{\text{Hist}(2x+1) + n \text{Hist}(2x)} \geq 1$$

$$\implies \frac{(n+1) \text{Hist}(2x)}{\text{Hist}(2x+1) + n \text{Hist}(2x)} \text{coeff}_\text{total} \geq \text{coeff}_\text{total} \quad (24)$$

From equation 20, L.H.S. of the above equation is $\gamma(2x, 2x+1)_\beta$ and R.H.S. is $\text{Index}_{\text{max}}$, $\gamma(2x, 2x+1)_\beta \geq \text{Index}_{\text{max}}$, which is not valid.

Similarly, using $\gamma(2x, 2x+1)_\beta$ as the starting point for proof, we get $\gamma(2x, 2x+1)_\alpha \geq \text{Index}_{\text{max}}$

We now know that one of the $\gamma(2x, 2x+1)$ will be greater than or equal to the $\text{coeff}_\text{total}$. Hence, the smaller one of the two is the legitimate value. Hence, $\gamma(2x, 2x+1)$ can be written as

$$\gamma(2x, 2x+1)_{\min} = \min\left\{\gamma(2x, 2x+1)_\alpha, \gamma(2x, 2x+1)_\beta\right\} \quad (25)$$

From the above, we conclude that the stop point for the pair $(2x, 2x+1)$ would likely be the coefficient index at which the current value of $\gamma(2x, 2x+1)$ satisfies Eq. 25.

**B. Capacity Estimation**

The estimated embedding capacity in bits, $\text{Est}(2x, 2x+1)$, for coefficient pair $(2x, 2x+1)$ is:

$$\text{Est}(2x, 2x+1) = TC_{\text{Est}}(2x \rightarrow 2x+1) + TC_{\text{Est}}(2x \rightarrow 2x) + TC_{\text{Est}}(2x+1 \rightarrow 2x+1) \quad (26)$$

Simplifying equation 26 and using the valid $\gamma$, we get

$$\text{Est}(2x, 2x+1) = \gamma(2x, 2x+1)_{\min} \left[PC_{2x} + PC_{2x+1}\right] \quad (27)$$

Since, we change a ±1 to 0 and vice-versa without any compensation, the estimated capacity of triplet (-1, 0, 1), $\text{Est}(-1,0,1)$, would be $\left[\text{Hist}(-1) + \text{Hist}(0) + \text{Hist}(1)\right]$. Total expected capacity including negative coefficients and (-1,0,1) triplet is:

$$\text{Est}_\text{total} = \text{Negative Coefficient pairs Capacity} + \text{Est}(-1,0,1) + \text{Positive Coefficient capacity}.$$

$$\text{Est}_\text{total} = \sum_{x=1}^{\text{coeff}_\text{total}/2} \left[\text{Hist}(-1) + \text{Hist}(0) + \text{Hist}(1)\right]$$

$$+ \sum_{x=1}^{\text{coeff}_\text{total}/2} \left[\gamma(2x, 2x+1)_{\min} \left(\text{PC}_{2x} + \text{PC}_{2x+1}\right)\right]$$

Let $M_E$ be the total number of message bits. Since, we are using matrix embedding with code $(1,n,k)$, we can rewrite equation 28 as:

$$\text{Est}_\text{total} = \frac{n}{k} \text{ where } n = 2^k - 1 \quad (29)$$

The appropriate value of $n$ and $k$ can be then determined by iterating over each value of $k \geq 1$ till $\frac{M_E}{\text{Est}_\text{total}} \leq \frac{1}{2}$. In the previous calculations we did not consider the position of individual coefficients in the DCT array. We can rewrite equation 27 as:

$$\text{Est}^t(2x, 2x+1) = \gamma(2x, 2x+1)_{\min} \left[PC_2^t + PC_2^{t+1}\right] \quad (30)$$

where $\text{Est}^t(2x, 2x+1)$ represents the estimated capacity of coefficient pair $(2x, 2x+1)$ occurring at $i_{th}$ index of the DCT array. $\text{Est}^t(2x, 2x+1)$ can be calculated once the value of $n$ is determined from equation 29.

**VIII. Results**

The algorithm was implemented in Java which includes code to, 1) decode a JPEG image to get the JPEG coefficients, 2) embed data in eligible coefficients, 3) balance the dual histograms to their original values, and finally, 4) re-encode the image in JPEG format with modified coefficients while preserving the original quantization tables and other properties of the image. Tests were performed on 3000 different JPEG color images of varying size and texture obtained from National Geographic. Every image was embedded with random data bits using a randomly generated password. The password is used to generated the pseudo random number sequence for determining the traversal sequence for coefficients.

**A. Increase in 1, -1 and 0 coefficients**

J4 doesn’t compensate for changes made to -1, 1 and 0 coefficients. However, as mentioned earlier, the ratio of the total number of -1 and 0 coefficients is maintained in order to retain the shape of the histogram and reduce detection. J4 doesn’t make changes to any 0, 1 and -1 which are outside the $\text{Index}_{ih}$ range. This is done to ensure we don’t change a 0 to a 1 in the right part of the DCT array where most of the coefficients are 0. Figure 3 shows the percentage increase in total number of 1 and -1 coefficients for 0.05 bits per non zero coefficient. The figure shows that the increase in 1 and -1 doesn’t exceed more than 2% of its initial count. It also shows that the increase is almost identical in both since the lines overlap in most of the areas. This confirms that the ratio of 1 and -1 is maintained even after the embedding process. The secondary axis shows the decrease in the number of zeros. The total number of zeros will decrease since the number of zeros is much larger than 1 and -1 and hence, more zeros will change to either 1 and -1 as compared to 1 and -1 being changed to zeros. Again, the percentage decrease in number of zeros is below 0.2 percent which is almost negligible. Around 3000 images were used for this experiment, but we have only shown a random fraction of them for simplicity.
B. Embedding Efficiency

Figure 4 shows the embedding efficiency for $bpnz$ of 0.05, 0.1, 0.2 and 0.3. Efficiency is defined as the average number of bits embedded per change. Again, the sample has been taken from 3000 images but only a small number of them have been considered for clarity of the graph. The efficiency for 0.05 $bpnz$ is around 8 which means we can embed 8 bits of message by changing just one bit.

C. Payload Analysis

Figure 5 shows the actual amount of data embedded at $bpnz$ of 0.05 and 0.1. Steganalysis discussed in Section IX would show that $bpnz$ of 0.5 and 0.1 have a very low detection rate. This graph shows how much data we can embed in a typical JPEG images without being detected. As shown, we can embed 1 KB of data in a 64 KB image or 4 KB of data in a 240 KB image. Since 240 KB is a typical size for a JPEG image, we can embed a decent amount of information without being detected by any current algorithm. The graph also shows the maximum capacity which is the amount of data we can embed with full histogram restoration. However, the detection rates would increase drastically at the maximum capacity because of second order statistical changes to the image. These changes include inter-block and intra-block correlations which are very difficult to compensate for in a given image. The sharp rise and fall in the graph is due to the number of non zeros in those images. The fall in the curve indicates that the number of non zeros are less than the previous image where a rise indicates an increase in the non zeros. Non zeros will increase in images that have more features and edges in them such as an image with lots of trees. The decrease indicates an image with less features like a plain sky or images with less edges in them.

IX. Steganalysis of J4

Steganalysis experiments for J4 are based on Support Vector Machines (SVM) for classification of images embedded with the following stego algorithms: F5, nsF5 (no shrinkage F5 with wet paper codes), Outguess, Steghide, MB1 (model-based without blockiness), MB2 (model-based with blockiness), PQ (Perturbed Quantization), PQT (Texture based PQ), PQE (Energy-based PQ) and J4 along with the cover images. We use soft margin SVM (C-SVM) with RBF (Radial Basis Function) kernel, which is one of the most popular choices of kernel type for SVMs. We use “LIBSVM” [29] tool, which is a library for SVM classification. The experiments use a feature extractor which extracts 274 merged Markov and DCT features.
for steganalysis as mentioned in [4]. We used this merged feature extractor since it outperforms DCT or Markov based steganalysis by itself, as shown in the authors’ results.

A. Pre-Processing of Images for Steganalysis

The following steps are carried out for all classification experiments:

1) Embed equal data in the images using the given algorithms with a fixed $bpnz$.
2) Extract the 274 merged features from the images.
3) Transform the extracted features to the LIBSVM format.
4) Perform simple scaling on the transformed data.
5) Use cross validation (grid search) to find the best $(C, \gamma)$. We use a cross validation tool provided in the LIBSVM library for this purpose.
6) Use the optimized $(C, \gamma)$ to train the randomly chosen training set.
7) Perform prediction on the testing data using the trained model. Training and testing data are exclusive.
8) Randomize the training and testing set and repeat steps 6 and 7. $(C, \gamma)$ remains constant throughout all the iterations.
9) Calculate the average result from all the iterations.

3000 JPEG color images with different texture and size ranging from 60 KB to 1000 KB were used for the steganalysis experiment. Every image was embedded with random data using the 10 above mentioned algorithms. At the end of embedding process, we have 10 sets of images containing 3000 stego images in each set. Each set consists of all the stego images embedded with only one of the 10 algorithms. We also have one set of cover images without any embedding. 70% of the images from each set were used for training and the rest 30% were used for testing. The training and testing sets are mutually exclusive. We performed 100 iterations of each experiment by randomizing the training and testing data to get a more accurate result.

B. Binary classification

We first performed a binary classification where only one of the stego sets and the cover set were used for training and testing. We performed this binary classification for all the 10 algorithms. At the end of embedding process, we have 10 sets of images containing 3000 stego images in each set. Each set consists of all the stego images embedded with only one of the 10 algorithms.

We also have one set of cover images without any embedding. 70% of the images from each set were used for training and the rest 30% were used for testing. The training and testing sets are mutually exclusive. We performed 100 iterations of each experiment by randomizing the training and testing data to get a more accurate result.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$bpnz = 0.05$</th>
<th>$bpnz = 0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQ</td>
<td>97.11</td>
<td>97.18</td>
</tr>
<tr>
<td>PQT</td>
<td>96.77</td>
<td>97.77</td>
</tr>
<tr>
<td>PQE</td>
<td>98.15</td>
<td>98.95</td>
</tr>
<tr>
<td>Outguess</td>
<td>97.84</td>
<td>97.79</td>
</tr>
<tr>
<td>Steghide</td>
<td>83.26</td>
<td>93.66</td>
</tr>
<tr>
<td>MB1</td>
<td>78.13</td>
<td>92.96</td>
</tr>
<tr>
<td>MB2</td>
<td>80.74</td>
<td>93.59</td>
</tr>
<tr>
<td>F5</td>
<td>93.47</td>
<td>93.61</td>
</tr>
<tr>
<td>nsF5</td>
<td>47.43</td>
<td>59.19</td>
</tr>
<tr>
<td>J4</td>
<td>47.10</td>
<td>58.28</td>
</tr>
</tbody>
</table>

TABLE II: Comparison of detection rate(%) of J4 with other algorithms using SVM binary-classifier with 0.05 and 0.1 $bpnz$. TP = True Positive, TN = True Negative.

Results in Table II and III show that J4 outperforms other algorithms (with the exception of nsF5) by a huge margin in terms of detection rate with $bpnz$ of 0.05. nsF5 doesn’t do any compensation. J4 incurs extra cost by compensating for all the individual DCT modes. All the dual histograms are exactly the same as the cover image in J4. “True Positive” (TP) here refers to the correct prediction accuracy for the stego images whereas “True Negative” (TN) refers to the correct prediction accuracy of cover images against that particular stego algorithm. Hence, the lower the TP and TN are, the better the stego algorithm is. Results show that the SVM classifier was only able to classify 47% of the images in J4 category at 0.05 $bpnz$. It classified 51% of the J4 images as cover images, which proves that J4 resembles the characteristics of a cover image when the payload is less. Other algorithms classification rate was more than 80% on average at the same embedding rate. With 0.1 $bpnz$, J4 has a true positive rate of 58% whereas most algorithms have a true positive rate of more than 90% except nsF5. It also outperforms nsF5 by a small margin at 0.1, 0.2, and 0.3 $bpnz$. Since 50% detection rate is classified as a random guess, a detection rate of 50-60% for J4 proves that J4 could be an ideal candidate for a JPEG steganography algorithm.

X. Conclusion

J4 is a new JPEG steganography algorithm that uses LSB encoding to embed data and individual and global histogram compensation to balance all the coefficients changed during the embedding process. J4 makes changes to the coefficients in a way that the individual histograms are preserved as in the cover image. The preservation scheme does not apply to 1, 0 and -1 coefficients. This is done to leverage the high number of these coefficients for increased efficiency since J4 uses matrix embedding to decrease the number of coefficient changes. All the coefficients except 1, 0 and -1 are changed in pairs. The theoretical estimation ensures that enough coefficients are left after the embedding process to compensate for the changes to the coefficients at each index position in the DCT array.

We compared J4 with 9 popular algorithms: F5, nsF5, Steghide, OutGuess, MB1, MB2, PQ, PQT and PQE. Extensive steganalysis performed on these algorithms prove that detection rate of J4 is around 47%, 58%, 76% and 88% at 0.05, 0.1, 0.2 and 0.3 bits per non-zero coefficients ($bpnz$), respectively. On the other hand, the average detection rate of other algorithms except nsF5 were 90%, 95%, 97% and 98% for the same $bpnz$. The detection rate for other algorithms were around 40% higher on average compared to J4. nsF5 performs equally well as J4 at 0.05 $bpnz$ but J4 outperforms nsF5 with a small margin at higher embedding rates. Moreover, nsF5 doesn’t do any compensation in contrast to J4. Since, 50-60% detection is classified as random guess in SVM classification, the
results prove that J4 would be an ideal candidate for embedding data at low rates.

The steganalysis method was based on 3000 color JPEG images downloaded from National Geographic website. The 274 merged DCT and Markov feature extractor used for steganalysis was the best available at the time of writing this article in detection accuracy, as claimed by the authors in [4]. Although, the steganalysis method (feature extractor) in the experiments uses both first and second order statistics to detect anomalies and J4 is a first-order restoration scheme, it is still able to perform extremely well on lower data rates and beat this steganalysis system.

REFERENCES