ANNEXURE II

(A). LIST OF PUBLICATIONS

JOURNALS:


Abstract

Compression is important because it helps us to reduce the consumption of expensive resources, such as disk space and cost for using bandwidth. Use of the right compression method and encryption technique yield more compact and secure data that are eligible to transfer over public channel. We introduce an optimal approach to compress and then encrypt images. The proposed method is used to increase the compression ratio for images by segmenting an image into non-overlapping segments called edge and non-edge segments and applying different compression depths for these segments. In an edge segment (a region that contains important information) the compression ratio is reduced to prevent loss of information, whereas in a non-edge segment (a smooth region which does not have important information), a high compression ratio is achieved and for encryption a non linear chaotic map is used. The performance of the proposed system is quantified using the peak signal-to-noise ratio to test the compression ratio and also the encrypted image has huge key space which makes harder for hacker to decrypting it using wrong key.

Keywords: Data compression, Scanning Pattern, Image Encode, Linearization, Chaotic Map.

I. Introduction

Data compression has important application in the areas of data transmission and data storage. Many data processing applications require storage of large volumes of data, and the number of such applications is constantly increasing as the use of computers extends to new disciplines. At the same time, the proliferation of computer communication networks is resulting in massive transfer of data over communication links. Compressing data to be stored or transmitted reduces storage and/or communication costs. When the amount of data to be transmitted is reduced, the effect is that of increasing the capacity of the communication channel. Similarly, compressing a file to half of its original size is equivalent to doubling the capacity of the storage medium. It may then become feasible to store the data at a higher, thus faster, level of the storage hierarchy and reduce the load on the input/output channels of the computer system. Broadly, compression is classified as lossless and lossy. The lossless technique generates an exact replica of the original data stream during decompression. This technique is used when storing data base records, spreadsheets or word processing files. On the other hand lossy compression is preferred for archiving and often for medical imaging, technical drawings, etc. where quality of the data is compromised to an acceptable level. Developing innovative schemes to accomplish effective compression has gained enormous popularity in recent years. A brief review of some recent significant research is presented here.

Gupta and Anand [10] introduced algorithm based on adaptive quantization coding (AQC) algorithms. The objective is to reduce bit rate produced by AQC while preserving the image quality. The proposed algorithms used only selected bit planes of those produced by encoder using bit plane selection using threshold (BPST) technique. The bit planes are selected by using an additional processing unit to check the intensity variation of each block according to a predefined threshold. John and Girija[20] proposed novel and high performance architecture for image compression based on representation in the frequency domain. The digitized image is compressed using discrete Hartley transform (DHT), discrete Walsh transform, discrete Fourier transform, and discrete Radon transform and their combinations with DHT. DHT is used as a basic transform because of its...
Douak et al. [5] have proposed a new algorithm for image compression. After a pre-processing step, the DCT transform is applied and followed by an iterative phase including the threshold, the quantization, dequantization, and the inverse DCT. Pixels in an image generally, exhibit a certain amount of correlation with their neighbours. In other words, they are highly redundant. DCT transforms such correlated data into an uncorrelated data with minimum information redundancy since the cosine function comprises an orthogonal basis [25]. This transformation is a lossless for any dimensional data; therefore, the inverse operation of the same reconstructs original data ideally. A multidimensional transform can be derived from consecutive application of one-dimensional (1-D) transform in an appropriate direction. Because of energy packing efficiency performance criteria, a two dimensional (2-D) version of DCT is widely used in image processing and analysis in problems of spectral analysis, data compression, and pattern matching and so on. A 2-D DCT is obtained by performing a 1-D DCT on the columns of a matrix and then 1-D DCT on the rows of a matrix [6], [9]. These two operations are interchangeable, for any higher dimensions. The same can also apply for an inverse transformation. The results obtained from 2-D DCT will be in an ordered fashion where the mean value of a matrix, known as dc coefficient is in the upper left (0, 0) of the matrix and small high-frequency values, known as ac coefficients are following it. In general, at first, segmenting the image into several square blocks is performed, and DCT is applied to each block. In practice, the block size N most often equals 8 because a larger block takes a great deal of time to perform DCT calculations, which can produce unreasonable trade off. A 2-D DCT operates on one block at a time in a left-to-right and top-to-bottom manner for JPEG images in general [15]. For a N x N image block sequence D(i, j) with {0 ≤ i, j < N}, it is defined as,

\[ D(i, j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} P(x, y) \cos \left[ \frac{(2x+1) \pi i}{2N} \right] \cos \left[ \frac{(2y+1) \pi j}{2N} \right] \]  

(1)

\[ C(i)C(j) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{for } i, j = 0 \\ 1, & 1 \leq i, j \leq N - 1 \end{cases} \]  

(2)

D (i, j) and P (x, y) represents an input pixel. Since DCT is perfectly reversible, the inverse DCT (IDCT) is defined as [11],

\[ P(x, y) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i)C(j)D(i, j) \cos \left[ \frac{(2x+1) \pi i}{2N} \right] \cos \left[ \frac{(2y+1) \pi j}{2N} \right] \]  

(3)

When DCT is applied to a square block, it converts highly correlated data set in a relatively independent data set [5] and results a dc coefficient, and a series of ac coefficients of zero values at high frequency and small values at low frequency. Since DCT is a lossless transformation, the function Inverse-DCT (IDCT) results original pixel value [25].

Quantization is the process of approximating the resultant DCT matrix into a small set of values, to determine what information can be discarded safely without a significant loss in visual fidelity [25]. Consequently, that leads to the development of lossy compression [10]. The quantization is performed by using the following equation.

\[ P_{\text{quant}}(x, y) = \text{Round} \left( \frac{P(x, y)}{Q(x, y)} \right) \]  

(4)
Before encoding dc coefficients directly, a Differential Pulse Code Modulation (DPCM) is performed in order to reduce the further size of it. In which the difference between the successive block dc coefficients is calculated as,

\[ \Delta DC_i = DC_i - DC_{i-1} \]  

To obtain the best possible compression ratio, the next step is to apply an adaptive scanning [11] providing, for each \((n \times n)\) DCT block a corresponding \((n \times n)\) vector containing the maximum possible run of zeros at its end. The last step is the application of a modified systematic lossless encoder. In this paper, the proposed method achieves a new image compression ratio because of a scanning pattern called ZZRD, which makes a balance on compression ratio and image quality by compressing the vital portions of the image with high quality. In this approach, the main subject in the image is more significant than the background image. The performance of the proposed scheme is evaluated in terms of the peak signal to noise ratio and the compression ratio attained. The experimental results demonstrate the effectiveness of the proposed scheme in image compression. In the image compression algorithm, the input image is RGB space, and then the image is initially classified into edge and non-edge portions using Canny method [3]. Then the image is subdivided into 8x8 blocks and DCT coefficients are computed for each block. The quantization is performed conferring to a quantization table [8],[9],[10]. The quantized values are then rearranged according to a new scan arrangement as described in next section rather than traditional methods shown in Fig. 1.

\[ \text{Figure. 1 The various methods of DCT coefficient scanning methods: (a) Vertical, (b) Hilbert, (c) Zigzag (d) Horizontal} \]

II. MATERIALS AND METHODS

Scanning Patterns

A scan method can convert a difficult to use signal into an easy to use one by exploring source redundancies within the input signal, which could be very useful in image compression. In Joint Picture Expert Group (JPEG) image compression [22] encoding an image can lead to not an optimum result, if the quantized two-dimensional discrete cosine transform coefficients are not arranged in linear sufficiently. An efficient linearization scheme is necessary to keep highly redundant coefficients consecutively in one dimensional sequence space so that other techniques involved in the compression process can be expected to yield better results. In image compression, particularly JPEG images, exploring the content of an image depends on the way in which it is scanned.
For an instance, information obtained by scanning the image horizontally differs from those obtained by scanning it vertically [15], [16], [17]. Since there are several ways to scan the image, there are also several possible interpretations of its content. Thus finding the scan that provides more useful and relevant information of the image can be useful for image processing. In the compression context, efficient scanning must be able to explore most redundancy in the image. The JPEG encoder for instance, scans the image block in Zigzag pattern. It is proved that the performance of such encoding depends partly on the way in which the image was scanned. Thus, searching a way to find the most suitable scan may be useful for image compression[19],[20].

In this paper, a novel scanning scheme proposed by S Sankar et.al., 2014 has been implemented for performing linearization where a hybrid procedure is employed by integrating a portion of Zigzag with new Raster Diagonal patterns shown in figure 2 to analyse correlation in the image and strengthen the correlation in the resulting linear pixel sequence, which is easily exploitable for purposes of compression.

Zigzag procedure begins at 1 and ends at 2 and Raster Diagonal begins at 3 and continues till the end of the matrix is reached. The algorithms are applied individually into JPEG Base Line compression process and the performance is analyzed for discrete images with distinguishing factors. The result of quantization process is a 2-D square matrix that holds a dc coefficient and mixed frequency symbols of ac coefficients. Before we start encoding the coefficients, they can be further reduced by applying Differential Pulse Code Modulation (DPCM) method for dc and Run Length Encode (RLE) for ac coefficients respectively. RLE always expects a high
frequency \[24\] of the same symbol in a consecutive manner so that it can be replaced by a low frequency symbol, if so, the encoder will produce a smaller number of bits otherwise not. As a fact of this, a better scanning pattern is required to gather high frequency of symbols in places where assorted symbols are presented. Therefore, the scanning pattern only determines the order in which the AC coefficients should be considered to facilitate the encoding process on them.

**ZigZagRasterDiagonal Pattern**

In which, a 2-D square matrix is split into exactly two halves diagonally, and the upper left part is scanned in Zigzag fashion, and the lower right part is in a RasterDiagonal fashion as shown in figure 2b. The Zigzag pattern scans \(\frac{(n^2)}{2}\) + \(\frac{n}{2}\)-1 number of matrix locations whereas RasterDiagonal pattern scans \(\frac{(n^2)}{2}\) - \(\frac{n}{2}\)+1 amount locations where n is the total number of the rows or columns of a square matrix. The algorithm for the ZigzagRasterDiagonal is given below.

**Algorithm ZigzagRasterDiagonal**

**Input:** \(Q[]\): 2-D array, \(n\): is the total number of rows or columns of a square matrix  
**Output:** \(A[]\): 1-D array  
row ← col ← 1  
i ← t ← 0  
\(N\)← \(n^2\)  
\(A[]\) ← a new 1-D array of size \(N\)  
while i is less than \(N\) do  
    \(A[i]\) ← \(Q[\text{row} – 1][\text{col} – 1]\)  
    if \(i < \frac{(N/2)+(n/2)-1}{2}\) then  
        if \((\text{row} + \text{col}) & 1 == 0\) then  
            if \(\text{col} < \text{n}\) then  
                \(\text{col}++\)  
            else  
                \(\text{row} += 2\)  
        else  
            if \(\text{row} > 1\) then  
                \(\text{row}--\)  
            else  
                if \(\text{row} < \text{n}\) then  
                    \(\text{row}++, \text{col}==2\)  
                else  
                    if \(\text{col} > 1\) then  
                        \(\text{col}--\)  
                    else  
                        if \(\text{row} < \text{n}\) then  
                            \(\text{row}++, \text{col}==2\)  
                        else  
                            \(\text{row}++, \text{col}--\)  
                            \(\text{row}, \text{row} ← t\)  
    end while  
return \(A\)

**Step 0:** Initialize \(i=1\).

**Step 1:** Follow the procedure Zigzag until to reach the bottom most left cell.  
**Step 2:** Move to top right by assigning incremented column value to row and old row value to the column.  
**Step 3:** Move to the bottom once by incrementing row by 1 and decrementing column by 1 until to reach the bottom side.  
**Step 4:** Repeat step 2 and 3 until to reach the last cell.

**Compression at different depths**

The proposed compression algorithm named as Depth 4 (D4). We evaluate the efficiency of compression by evaluating the Peak Signal to Noise Ratio (PSNR). Images of different sizes (512x512 and 256x256) are considered in the experiment, most of which are commonly used in the evaluation of computer vision and image processing algorithms.

Depth 1 (D1): Without classifying the image as edge and non-edge, all AC coefficients of the edge blocks and non-edge blocks on each component (RGB space) are used for compression.

Depth 2 (D2): All AC coefficients of the edge blocks on each component (RGB space) are used. After quantization and new scan the non-zero of the quantized coefficients is counted and all AC coefficients will be used as the input of the Huffman coding. The non-edge block will be coded using only the DC coefficient.

Depth 3 (D3): In this method, we tried to reduce the number of AC coefficients used in coding the edge blocks. This will reduce the effect of image noise, increase the compression ratio, and accelerate the coding process, which only the quantized DC coefficient value will be used for non edge blocks.
Depth 4 (D4): a 50% (chosen experimentally) of the non-zero AC coefficients of the edge blocks on R component, 50% of the non-zero AC coefficients of the edge blocks on G component, and 50% of the non-zero AC coefficients of the edge blocks on B component are used. After quantization the non-zero quantized coefficients are counted and only the first 50% of the non-zero AC coefficients on each component (R, G, and B component) is used as the input for Huffman coding. In case of non-edge blocks, only DC coefficients are taken into consideration for encoding.

Non Linear Chaotic Map (NCP)

In the context of Information Science, data compression is called as source coding where encoding done at the source of the data before it is stored or transmitted and decoding is done at the destination of the data. When it is desired to transmit redundant data over an insecure and bandwidth constrained channel, it is customary to first compress the data and then encrypt it. Compression always relies on high redundant data in order to gain size reduction. Since encryption destroys redundancy, the compression algorithm would not be able to give much size reduction, if it is applied on encrypted data. Compression schemes work by finding patterns in input data that can be destroyed during the encryption which increases compression times. Compressed data can vary considerably for small changes in the source data, therefore making it very difficult to perform differential cryptanalysis on compressed data. For these reasons, we preferred to perform compression before encryption.

The image encryption algorithm used in this study is based on the proposed NCA map. It uses chaotic sequence generated by NCA map to encrypt image data with different keys for different images. Original chaotic sequence \( \{x_0, x_1, x_2, \ldots\} \) consists of decimal fractions. However images are digital. So a map is defined to transform the chaotic sequence to another sequence which consists of integers. Then plain-image image can be encrypted by use of XOR operation with the integer sequence. It is depicted in the below figure 3. The encryption steps are as follows:

Step 1: Set encryption keys for the plain-image, including structural parameters \( a, b \) and initial value \( x_0 \).

Step 2: Do 100 times of chaotic iteration as formula, and obtain the decimal fraction \( x_{100} \).

Step 3: If the encryption work is finished, then go to step 6; otherwise do three times of chaotic iteration; and as a result, a decimal fraction, such as \( x_{103} \), will be generated, which is a double value and we choose its first 15 significant digits.

Step 4: Divide the 15 digits into five integers with each integer consisting of three digits. For each integer, do mod 256 operation, and another 5 bytes of data will be generated.

Step 5: Do XOR operation using the 5 bytes of data with 5 bytes of image data (grey value or color RGB value).

Step 6: Pass the encrypted image through communication channel.

The nonlinear aspect of the algorithm is provided by the use of a power function and a tangent function in the recursive generation of the \( x_n \). The formulae used to generate the chaotic matrix are as follows:
The comparison of performance is the variance of the (I, j) spectral component. Clearly, NCV(n) provides a measure for the percentage of results. For edge blocks, some of the non-zero quantized AC coefficients will be eliminated based on its power. As known, the quantization matrix is computed based on the variance of the DCT coefficients. The quantization of a single coefficient in a single block causes the reconstructed image to differ from the original image by an error image proportional to the associated basis function in that block. Moreover, the elimination of some quantized coefficients may give clearly visible errors, i.e., the blockiest of the artifacts distinguishes them from the original image content. We tried to address this problem using two experimental tests. These tests can be summarized as follows.

Step 1: For edge blocks the statistical variances of the DCT coefficients will be estimated and the normalized cumulative variance (NCV) of the AC coefficients will be computed. The NCV values are recorded according to the spectral component index. The NCV at the nth spectral component, where n ∈ [0, N - 1], is defined as

\[ NCV(n) = \frac{\sum_n \sigma_{ij}^2}{\sum_n \sigma_{ij}} \]  
where \( \sigma_{ij}^2 \) is the variance of the (I, j) spectral component. Clearly, NCV(n) provides a measure for the percentage of the AC coefficients that can be selected for accepted quality. A set of images with different details has been used to test the NCV(n). On the average, 18% of the DCT coefficients contain about 80% of the total power of the image signal.

Step 2: Assume that the edge variance V is the sum of the squared difference for all such pixel pairs

\[ v = \sum (x1 - x2)^2 \]  
where x1 and x2 are the image values of two pixels that are next to each other in the same row, but are in different blocks. The edge variance is estimated for the original image (Vo) and the reconstructed image (Vr). Using the pixels just beside the edge on both sides and taking the average. Experimentally, for (Vr=Vo) > 1.3 the blocking artifact will be clearly visible. A set of images with varying sizes range from 128x128 to 1024x1024.

III. RESULTS AND DISCUSSION

The best coding results are achieved with the D4 coding scheme based on ZZRD scan [25]. The ZZRD scan has a no effect even the higher complexity of the D4 coding scheme. The time complexity of ZZRD is O (N), where N corresponds to the total number of pixels in the image. The proposed method tested with 1GB of standard test images with varying sizes range from 128x128 to 1024x1024. Figure 4a and 4b show that comparison of PSNR and MSE of proposed method with classical methods. It can be seen that proposed method outperforms with at least 1-2.4 dB for PSNR up to and at least 2.22% of error percentage in average. The difference is much greater at high bit-rate. It can be seen that depth D1 in the proposed scheme maintains an acceptable PSNR (higher than 3.3 dB) even at a bit-rate as high. A 256 × 256 size 8 bits gallopinghorse.jpg image has been considered as an example for encryption. The encrypted image is depicted in Figure 4. The image is encrypted with initial parameters are (\( \alpha, \beta, \gamma, \chi \)) = (1.47, 5, 4.876545676545671, 2.3).
### Table 1. Reconstructed images obtained from experiment

<table>
<thead>
<tr>
<th>Test Images</th>
<th>Original</th>
<th>Reconstructed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>D1</td>
</tr>
<tr>
<td>Lena</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
<tr>
<td>Galloping Horse</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
<tr>
<td>Black Hood Lady</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
<tr>
<td>Prague</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
</tbody>
</table>

**Figure 4(a) PSNR (b) MSE**
The best coding results are achieved with the D4 coding scheme based on ZZRD scan [25]. The ZZRD scan has a no effect even the higher complexity of the D4 coding scheme. The time complexity of ZZRD is O (N), where N corresponds to the total number of pixels in the image. The proposed method tested with 1GB of standard test images with varying sizes range from 128x128 to 1024x1024. Figure 4a and 4b show that comparison of PSNR and MSE of proposed method with classical methods. It can be seen that proposed method outperforms with at least 1–2.4 dB for PSNR up to and at least 2.22% of error percentage in average. The difference is much greater at high bit-rate. In fact, it can be seen that depth D1 in the proposed scheme maintains an acceptable PSNR (higher than 3.3 dB) even at a bit-rate as high. A 256 × 256 size 8 bits gallopinghorse.jpg image has been considered as an example for encryption. The encrypted image is depicted in Figure 4. The image is encrypted with initial parameters are \((\alpha, \beta, \gamma, \tau, \chi_0) = (1.47, 5, 4.876545676545671, 2.3)\).

![Figure 5](image_url)

The secret key should produce a completely different encrypted image. For testing the key sensitivity of the proposed image encryption procedure, we use the wrong key, initial parameters are \((\alpha, \beta, \gamma, \tau, \chi_0) = (1.47, 5, 4.876545676545672, 2.3)\). We can note that image still unclear as seen in figure 6.
For a secure image cipher, the key space should be large enough to make the brute force attack infeasible. The key of the new algorithm consists of three floating-point numbers. If we use the first 20 digits of a floating-point number, then there are 100 uncertain digits are possible. So the possible key space is $2^{100}$. An image cipher with such a long key space is sufficient for reliable practical use.

IV. CONCLUSION

In this paper, we have implemented a hybrid scan called ZZRD and an image encoding at depth 4 for effectual compression of images and for encryption. A non-linear chaotic map is used. The performance of the technique with the recent research is conducted, and shown superior performance of our algorithm in terms of quantitative distortion measures, as well as visual quality and PSNR. The experimental results demonstrate the effectiveness of the proposed scheme in image compression. The encrypted image of compressed image has long key space i.e. $2^{100}$ which is good enough to make chaos for the hacker.

V. REFERENCES


Non Linear Chaotic Map for Secure Data Transmission

S Sankar  
Research Scholar, Department of IT  
Hindustan University Chennai, India  
sankar_jaikissan@yahoo.com

S Nagarajan  
Professor, Department of IT  
Hindustan University, Chennai, India  
snagarajan1960@gmail.com

Abstract: In today’s world, Internet plays a major role in people’s communication. People nowadays share and transfer variety of multimedia information through the Internet. Although a lot of benefit with it, data transfer over Internet is vulnerable to attack is a major hindrance. Cryptography is the science used to keep the information safe from attack. In the case of text data transfer more number of encryption techniques are existing whereas when it comes to image very less number of techniques are available. Also, the traditional image encryption methods are not viable enough for modern images due to their different storage formats. Hence research on image encryption becomes inevitable. In this paper, we have proposed Non linear chaotic map technique to encrypt the images and performance of the same has been evaluated. This study shows Non linear chaotic map has higher performance for images.

Keywords – Chaotic map, Cryptography, Image encryption, Data Security

1. INTRODUCTION

With the rapid development of multimedia and network technologies, the security of multimedia becomes more and more important, since multimedia data are transmitted over open networks more and more frequently. Typically, reliable security is necessary to content protection of digital images. Encryption can be defined as the art of converting data into coded form which can be decode by intended receiver only who poses knowledge about the decryption of the ciphered data. Encryption can be applied to text, image, and video for data protection. Image compression is an application of data compression that encodes the original image with few bits. The objective of image compression is to reduce the redundancy of the image and to store or transmit data in an efficient form.

Even though both data compression and encryption are methods to transform data into different representation, the goals tried to achieve by them are different. Data compression is done with the intension of decreasing the size of data, where encryption is done to keep the data secret from third parties. Data compression offers an approach for reducing communication costs, at the same time it is vulnerable to attack during the transmission. If it is compromised then it is not possible to get actual data during the decompression. Therefore security is needed to preserve the compressed data. Compression always relies on high redundant data in order to gain size reduction. Since encryption destroys redundancy [7], the compression algorithm would not be able to give much size reduction, if it is applied on encrypted data. For that reason, compression before encryption is the highly preferable order. In this chapter, we discuss a unique chaos based crypt analysis.

The complexity inherent in chaotic systems gives rise to the term chaos, which does not indicate complete disorder, as in everyday usage. Rather, chaos is the apparently random behavior of a system which is in fact deterministic. This means that the system has no inherent randomness or noise, and that the irregular behavior arises from its nonlinearity. Given a specific initial condition for a chaotic system, its behavior for all future time is well-defined and predictable. Several important characteristics serve to define a chaotic system. First, such systems are highly sensitive to initial conditions - two states with an arbitrarily small initial separation may have widely different final states, and perturbations from initial conditions grow exponentially. The phase space of a chaotic system is topologically mixing, such that any subset of phase space will eventually overlap with any other given subset. Phase space may contain structures such as regions of stability and points or regions of accumulation which are “strange” or otherwise quite complex. Finally, orbits of a chaotic system in phase space are by definition a periodic.

II. PROPOSED SYSTEM

The existing chaos based algorithms operate on two stages: the shuffling stage and the substitution stage. In the shuffling stage, the position of the pixels from the original image is changed by chaotic sequences [2] or by some matrix transformation,
such as Arnold transformation, magic square transformation, and so forth. These shuffling algorithms can be easily realized. Since these shuffling algorithms just involve changing the position of the pixels but not changing the pixel values it leads to histogram of the encrypted image same as the original image, thus the security of the image is threatened by statistical analysis.

Compared to the method of shuffling the method of substitution is more efficient and more secure as it involves changing the pixel values. Even such shuffling when applied alone, it leads to weaker encrypted image. Thereby in order to improve the security shuffling and the substitution are combined by some researchers [3, 4]. Chaotic Image encryption is a branch of cryptography in which we encrypt image data with the help of cryptographic tools based on chaos theory. Matthews first proposed the chaos-based encryption scheme in 1989 [3], and Fridrich first adopted chaotic map into image encryption in 1997 [4]. Since then, many chaos-based image encryption algorithms have been designed to realize secure communications. Chaotic systems have many important properties, such as the sensitive dependence on initial conditions and system parameters, pseudorandom property, no periodicity and topological transitivity, etc. Most properties meet some requirements such as diffusion and mixing in the sense of cryptography [4]. Therefore, chaotic cryptosystems have more useful and practical applications.

Non linear system is a chaotic system in which output of the system is totally unpredictable and dynamic since it uses chaotic maps. The chaotic maps are getting more attention recently in cryptanalysis since it is easy to solve but the result is bifurcation where at every point it changes from one functional behavior to another functional behavior. The chaotic map when it is iterated by a function f, in a space S then there is a change from one state to another, that is,

\[ s_{n+1} = f(s_n) \]  

where \( s_n \in S \) indicates the system state at discrete time. In chaos cryptography, the state space is typically finite binary space

\[ S = P = C \{0, 1\}^n, \quad n=1,2,3, \ldots \]  

where P is plain text and C is Cipher text. The initial value of a control parameter \( s_0 \in S \) is maintained throughout the iterations and it results dynamic \( c_n \in C \). Thus, depending upon the value of input control parameters, the non linear mathematical model gives unpredictable results. With more than one control parameters and initial conditions, high dimensional chaotic systems are most complex and have a big key space. However, complex calculations make the encryption algorithm too slow. To overcome these drawbacks, a nonlinear chaotic map (NCM) [29] is adopted. Encryption method based on nonlinear chaotic algorithm uses tangent and power function to give large key space. The experimental results show that this chaotic map has more complex chaotic behaviors than the linear chaotic map.

\[ x_{n+1} = (1 - \beta^4) \cdot c \cdot \tan \left( \frac{\alpha}{1 + \beta} \right) \cdot \left(1 + \frac{1}{\beta}\right)^\beta \cdot \tan(\alpha x_n) \cdot (1 - x_n)^\beta \]  

where \( \alpha, \beta \) are control parameters. When \( x_0 \in (0,1), \quad \alpha \in (0,1.4), \quad \beta \in (5, 43), \) or \( x_0 \in (0,1), \quad \alpha \in (1.4, 1.5), \quad \beta \in (9, 38), \) or \( x_0 \in (0,1), \quad \alpha \in (1.5, 1.57), \quad \beta \in (3, 15), \) NCM performs chaotic phenomena.

To get a faster encryption speed, every time NCM is iterated as a result, \( n \) bytes random numbers are gained. The standard image data sets are used to test the algorithm. It uses chaotic sequence generated by NCM map to encrypt image data with different keys for different images. Original chaotic sequence \( \{x_0, x_1, x_2, \ldots\} \) consists of decimal fractions. However images are all digital. So a map is defined to transform the chaotic sequence to another sequence which consists of integers. Then plain-image image is encrypted by performing XOR operation with the integer sequence. The encryption steps are as follows:

Step 1: Split the square image into \( n \) number of blocks of equal size and from left to right and top to bottom, we transform two-dimensional image to one-dimensional.

Step 2: Set encryption key \( K = (\alpha, \beta, x_0) \), with initial values.

Step 3: Do 100 times of chaotic iteration as formula, and obtain \( n \) bytes of random numbers.

Step 4: If the encryption work is finished for all blocks, then go to step 6; otherwise do three more times of chaotic iteration; and as a result, a decimal fraction will be generated, which is a double value and we choose its first 15 significant digits.

Step 5: Divide the 15 digits into five integers with each integer consisting of three digits. For each integer, do mod 256 operation, and another 5 bytes of data will be generated.

\[ \text{ISSN: 2321-8169} \]

Volume: 5 Issue: 5

380 – 384

IJRITCC | May 2017, Available @ http://www.ijritcc.org
Step 6: Do XOR operation using the 5 bytes of data with 5 bytes of image block (grey value or color RGB value). Output the calculation result to the object image and go to step 4.

Step 6: Regroup the blocks

III. RESULTS AND DISCUSSIONS

The image is encrypted with initial parameters \((\alpha, \beta, \gamma_0) = (1.47, 5, 3)\). In Figure 1a and 1b below, we show the histograms of RGB values of the plain images and those of the cipher images. The ideal histogram of a cipher image is uniform which indicates brute force attack on images makes difficult.

Fig. 1 (a)                                                                   (b)

The histogram of the encrypted image is reasonably consistent and notably dissimilar from the original image which directly indicates if any statistical attack is performed on encrypted image, and then the attacker will not give any hint to know about original image. Image pixel correlations are calculated using the following formulae to predict the quality of the image after decryption.

\[
E(x) = \frac{1}{N} \sum_{i=1}^{n} x_i
\]

\[
D(x) = \frac{1}{N} \sum_{i=1}^{n} (x_i - E(x))^2
\]

\[
cov(x,y) = \frac{1}{N} \sum_{i=1}^{n} (x_i - E(x))(y_i - E(y))
\]

\[
\rho_{xy} = \frac{cov(x,y)}{\sqrt{D(x)D(y)}}
\]
where $E(x)$ is the expectations of $x$, $D(x)$ is the variance of $x$ and $x, y$ are two neighboring pixel values, $N$ is the total number of pixels of image. A perfect encryption means there should be no correlation between the neighboring pixels. The following figure 2 shows that.

![Fig. 2. Correlation Before and After Encryption](image)

The percentage of correlation of both original and reconstructed image is close to each other which indicate quality of reconstruction of image from encrypted image is perfect. Since the complexity of the chaotic map is simple and even if it is introduced into an iterative function several times, the time for creating key space is negligible (less than 5 msec). This directly shows the encrypting and decrypting an image using NCM is more comfortable than the linear maps. For testing the sensitivity of the key, a wrong key is used to decrypt the encrypted image with same initial parameter values and it is important to note that image is still strange as seen in figure 4.

The difference of Number of Pixel Change Rate (NPCR) is estimated between the image which is encrypted using original key and wrong keys and we identified that the difference is random. During the each iteration, the algorithm generates 32 bit precision value among which first 15 precisions were taken into consideration as a key and it is XORed with pixel values. In addition to that 3 control parameters are used in the algorithm. If $n$ is the number of digits of a key and the number of control parameters are $m$ then $10^{n^m}$ possible keys can be generated which makes brute force attack impractical.

![Fig. 3. Correlation between Original and Reconstructed Images](image)
CONCLUSION

Non linear chaotic map is used in this paper to encrypt images, the result shows that the proposed method is performing well in terms of security. The key space generated by the algorithm is high enough against the several attacks. The percentage of correlation of both original and reconstructed image is close to each other which indicate quality of reconstruction of original image from encrypted image is perfect.

REFERENCES


ZZRD and ZZSW: Novel Hybrid Scanning Paths for Squared Blocks

S Sankar¹ and Dr. S Nagarajan²

¹Research Scholar @ Hindustan University and Associate Professor
@ KCG College of Technology, Chennai, India.
²Professor @ Department of Information Technology,
Hindustan University, Chennai, India, snagarajan1960@gmail.com
¹Corresponding author, e-mail: sankar_jaikissan@yahoo.com

ABSTRACT

A scan method can convert a difficult to use signal into an easy to use one by exploring source redundancies within the input signal, which could be very useful in image compression. In Joint Picture Expert Group (JPEG) image compression, encoding an image can lead to not an optimum result, if the quantized two-dimensional discrete cosine transform coefficients are not arranged in linear sufficiently. An efficient linearization scheme is necessary to keep highly redundant coefficients consecutively in one dimensional sequence space so that other techniques involved in the compression process can be expected to yield better results. In this paper, we propose two novel scanning algorithms for performing linearization where a hybrid procedure is employed by integrating a portion of Zigzag with new Raster Diagonal pattern and Saw tooth Wave pattern. The result shows proposed methods had competitive performance in comparison with the conventional Zigzag scanning scheme.

Keywords: Image compression, Linearization, Quantization, Zigzag scan.

1. INTRODUCTION

Compression is a technique used in a variety of disciplines to reduce the size of an object for saving storage space. For instance, in computer science and information theory, it is recognized as data compression, which lessens the number of bits needed to represent, store and transmit data. It plays a vital role in the areas of file storage, online and off-line data processing and mobile computing. Since rapid development in an object-oriented design, multidimensional data representation, digitization and
visualization, real-time data sensing, multimedia practice and others, many data
formats are not very space conscious. Such data are notorious for occupying disk
space, increasing transmission rates in terms of cost and time and decreasing the
availability of bandwidth and other network resources. On the other hand, a
compressed data appears to minimize the storage requirement [2], increases the speed
of data transfer, and the availability of networking resources over an uncompressed
data.

In JPEG image compression, exploring the content of an image depends among
other things, on the way in which it is scanned. For example, information obtained by
scanning the image horizontally differs from those obtained by scanning it vertically.
Since there are several ways to scan the image, there are also several possible
interpretations of its content. Thus finding the scan that provides more useful and
relevant information of the image can be useful for image processing. In the
compression context, efficient scanning must be able to explore more redundancy in
the image. The JPEG encoder for instance, scans the image block in Zigzag pattern. It
is proved that the performance of such encoding depends partly on the way in which
the image was scanned. Thus, searching a way to find the most suitable scan may be
useful for image compression [8]. In this paper, we propose two new scanning
algorithms namely, ZigZagSawtoothWave (ZZSW) and ZigZagRasterDiagonal
(ZZRD) to analyze correlations in the image and strengthen the correlation in the
resulting linear pixel sequence, which is easily exploitable for purposes of
compression. The algorithms are applied individually into JPEG Base Line
compression process and the performance is analyzed for discrete images with
distinguishing factors. The proposed methods are easy to implement and apparently
improve the encoding efficiency for the JPEG images.

The rest of the paper is organized as follows. Section 2 discusses about the already
existing JPEG image compression standard. Section 3 explains the design of proposed
algorithms. Section 4 gives the asymptotic nature of the algorithms. Section 5 presents
an empirical study of the algorithms. Experimental results are presented elaborately in
Section 6. It is then concluded in Section 7.

2. JPEG IMAGE COMPRESSION STANDARD
The aim of JPEG image compression is to make both gray scale and color image file
smaller and having them appear virtually the same as the original so that storing or
transmitting over the network makes sense in this present world. It uses Discrete
Cosine Transform (DCT) as a transform to produce an uncorrelated data from highly
correlated data [4, 5]. Pixels in an image generally, exhibit a certain amount of
correlation with their neighbors. In other words, they are highly redundant. DCT
transforms such correlated data into an uncorrelated data with minimum information
redundancy since the cosine function comprises an orthogonal basis. This
transformation is a lossless for any dimensional data; therefore, the inverse operation
of the same reconstructs original data ideally. A multidimensional transform can be
derived from consecutive application of one-dimensional (1-D) transform in an
appropriate direction. Because of energy packing efficiency performance criteria, a
two dimensional (2-D) version of DCT is widely used in image processing and analysis in problems of spectral analysis, data compression, and pattern matching and so on. A 2-D DCT is obtained by performing a 1-D DCT on the columns of a matrix and then 1-D DCT on the rows of a matrix. These two operations are interchangeable, for any higher dimensions. The same can also apply for an inverse transformation. The results obtained from 2-D DCT will be in an ordered fashion where the mean value of a matrix, known as dc coefficient is in the upper left (0, 0) of the matrix and small high-frequency values, known as ac coefficients are following it.

In general, at first, segmenting the image into several square blocks is performed, and DCT is applied to each block. In practice, the block size N most often equals 8 [2] because a larger block takes a great deal of time to perform DCT calculations, which can produce unreasonable tradeoff. A 2-D DCT operates on on one block at a time in a left-to-right and top-to-bottom manner for JPEG images in general [2]. For a N x N image block sequence D(i, j) with \(0 \leq i, j < N\), it is defined as [11],

\[
\begin{align*}
D(i,j) = & \frac{1}{\sqrt{2N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} P(x, y) \cos \left( \frac{(2x+1)i\pi}{2N} \right) \cos \left( \frac{(2y+1)j\pi}{2N} \right) \\
C(i), C(j) = & \begin{cases} 
\frac{1}{\sqrt{2}}, & \text{for } i, j = 0 \\
1, & 1 \leq i, j \leq N - 1 
\end{cases}
\end{align*}
\]

When DCT is applied to a square block, it converts highly correlated data set in a relatively independent data set [5] and results a dc coefficient, and a series of ac coefficients of zero values at high frequency and small values at low frequency. Since DCT is a lossless transformation, the function Inverse-DCT (IDCT) results original pixel value.

2.1 Quantization
Quantization is the process of approximating the resultant DCT matrix into a small set of values, to determine what information can be discarded safely without a significant loss in visual fidelity. Consequently, that leads to the development of lossy compression [10]. The quantization is performed by using the following equation.

\[
\begin{align*}
P_{\text{quant}}(x,y) = & \text{Round} \left( \frac{P(x,y)}{Q(x,y)} \right) 
\end{align*}
\]
Table 1. Quantization Table [10]

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>11</td>
<td>10</td>
<td>16</td>
<td>24</td>
<td>40</td>
<td>51</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>14</td>
<td>19</td>
<td>26</td>
<td>58</td>
<td>60</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>16</td>
<td>24</td>
<td>40</td>
<td>57</td>
<td>69</td>
</tr>
<tr>
<td>14</td>
<td>17</td>
<td>22</td>
<td>29</td>
<td>51</td>
<td>87</td>
<td>80</td>
</tr>
<tr>
<td>18</td>
<td>22</td>
<td>37</td>
<td>56</td>
<td>68</td>
<td>109</td>
<td>103</td>
</tr>
<tr>
<td>24</td>
<td>35</td>
<td>55</td>
<td>64</td>
<td>81</td>
<td>104</td>
<td>113</td>
</tr>
<tr>
<td>49</td>
<td>64</td>
<td>78</td>
<td>87</td>
<td>103</td>
<td>121</td>
<td>120</td>
</tr>
<tr>
<td>72</td>
<td>92</td>
<td>95</td>
<td>98</td>
<td>112</td>
<td>100</td>
<td>103</td>
</tr>
</tbody>
</table>

The JPEG quantization table assigns the larger quantization step size, usually seen in the lower right region for the high frequency components to discard the redundant information, and smaller quantization step size for the low frequency components seen in the upper left region to preserve the significant information.

2.2 Zigzag scan

Zigzag pattern (ZZ) is a common scanning pattern used in image compression, which is performed on the result of quantization process where the pixel values in a 2-D square matrix is reordered into a 1-D matrix [11]. Subsequently, a lossless encoding procedure called RLE is applied to the result of Zigzag scan. During the scanning, we visit each cell exactly once in some order and bring into being a 1-D matrix. Zigzag pattern scans the 2-D square matrix in a horizontal-diagonal-vertical-diagonal fashion starting from upper left to lower right as shown in figure 1.

![Zigzag scan](image)

**Fig. 1. Zigzag scan [6, 7, 8].** The number indicates a shift in its procedure.

The algorithm for ZZ is presented below.
Step 0: Initialize row =1 and column =1
Step 1: Move right once by incrementing column by 1
Step 2: Move to the bottom left by incrementing row by 1 and decrementing column
by 1 until to reach the left side.
Step 2.1 Check if bottom most left is reached. If true, go to step 6 or else continue
Step 3: Move to the bottom once by incrementing row by 1
Step 4: Move to the top right by decrementing row by 1 and incrementing column by 1 and continue this until to reach top side
Step 5: Repeat step 1 to 4 until the condition specified at step 2.1 is satisfied.
Step 6: Do step 1
Step 7: Move to the top right by decrementing row by 1 and incrementing column by 1 and continue this until to reach right side
Step 8: Do step 3
Step 9: Move to the bottom left by incrementing row by 1 and decrementing column by 1 until to reach the bottom side
Step 10: Repeat step 6 to 9 until to reach the last cell

2.3 Differential Pulse Code Modulation
Before encoding dc coefficients directly, a Differential Pulse Code Modulation (DPCM [8]) is performed in order to reduce the further size of it. In which the difference between the successive block dc coefficients is calculated as,

\[ \Delta DC_i = DC_i - DC_{i-1} \]  

(5)

Where \(i=1, 2, ..., N\) and on the decoding side, is used to obtain the actual dc value of \(i\).

2.4 Run Length Encode
The idea is to represent long repeated occurrences of a symbol with a pair called \((n, symbol)\) where \(n\) is the total number of occurrences of a symbol. A version of Run Length Encode (RLE) called Zero Run Length Encoding is used in JPEG compression. In which ac coefficient is represented in \((RUN, VALUE)\) format, where \(RUN\) is the total number of continuous zero ac coefficients preceding the nonzero ac coefficient and on the decoding side, the inverse process of zero run length encoding takes the zero run length encoded data as its input and produces the original ac coefficients as its output.

2.5 Encoding and Decoding
The encoding is performed with the help of standard JPEG Baseline Encoder Huffman Tables for the dc and ac coefficients, respectively. Each differential coded dc coefficient is encoded as \((SIZE, VALUE)\) format where \(SIZE\) indicates the number of bits required to represent the \(VALUE\), and \(VALUE\) is the equivalent binary values of the dc coefficient. The ac coefficient is encoded as \((RUN, SIZE, VALUE)\) where \(RUN\) is the number of continuous zero ac coefficients preceding the nonzero ac coefficient and \(SIZE\) indicate the number of bits required to represent the \(VALUE\), and \(VALUE\) is the equivalent binary values of ac coefficient. Decoding is merely the inverse of the encoding process.
3. PROPOSED METHODS
The result of quantization process is a 2-D square matrix that holds a dc coefficient and mixed frequency symbols of ac coefficients. Before we start encoding the coefficients, they can be further reduced by applying DPCM method for dc and RLE for ac coefficients respectively. RLE always expects a high frequency of the same symbol in a consecutive manner so that it can be replaced by a low frequency symbol, if so, the encoder will produce a smaller number of bits otherwise not. As a fact of this, a better scanning pattern is required to gather high frequency of symbols in places where assorted symbols are presented. Therefore, the scanning pattern only determines the order in which the ac coefficients should be considered to facilitate the encoding process on them. Zigzag (ZZ) scanning pattern is such a method and is widely used in many compression techniques, in particular, JPEG image compression. A study is conducted to determine the best performing scanning pattern. We focused on developing the algorithms ZZSW and ZZRD, and their performance is compared with the conventional pattern. The figure 2 illustrates the working principle of ZZSW and ZZRD.

![Fig. 2. (a) ZigzagSawtoothWave. The number indicates a shift in procedure. Zigzag procedure starts at 1 and ends at 2 and Saw tooth Wave procedure starts at 3 and continues to the end of the matrix. (b) ZigZigRasterDiagonal in which Zigzag procedure starts at 1 and ends at 2 and Raster Diagonal starts at 3 and continues till the end of the matrix is reached.]

3.1 ZigzagSawtoothWave Algorithm
It is a combination of Zigzag and Saw tooth Wave pattern. The Zigzag pattern starts scanning a 2-D square matrix at first, and it visits \((n^2)/2\) + \(n/2\)+1 number of matrix locations in a horizontal-diagonal-vertical-diagonal fashion starting from upper left to lower left. The Saw tooth Wave pattern starts scanning next to Zigzag pattern, and it visits total of \((n^2)/2\) - \(n/2\)-1 number of locations in a diagonal-vertical
fashion starting from lower left to lower right where \( n \) is the number of the rows or columns of a square matrix which is shown in figure 2a. The algorithm for ZZSW is presented below.

\[
\text{Algorithm ZigZagSawtoothWave}
\]

**Input:** \( Q[][] \): 2-D array, \( n \): is the total number of rows or columns of a square matrix  
**Output:** \( A[] \): 1-D array  
row ← col ← \( d1 \) ← 1  
i ← \( d2 \) ← 0  
N ← \( n*n \)  
\( A[] \) ← a new 1-D array of size \( N \)  
while \( i \) is less than \( N \) do  
    \( A[i] \) ← \( Q[\text{row}-1][\text{col}-1] \)  
    if \( i < [(N/2)+(n/2)+1] \) then  
        if \( (\text{row} + \text{col}) \& 1 == 0 \) then  
            if \( \text{col} < n \) then \( \text{col}++ \) else \( \text{row} +=2 \)  
            if \( \text{row} > 1 \) then \( \text{row}-- \)  
        else  
            if \( \text{row} < n \) then \( \text{row}++ \) else \( \text{col} +=2 \)  
            if \( \text{col} > 1 \) then \( \text{col}-- \)  
    else  
        if \( \text{row} < n \) then  
            if \( (\text{row} + \text{col}) \& 1 == 0 \) then  
                if \( \text{row} > 1 \) then \( \text{row}++ \)  
            else  
                if \( \text{row} < n \) then \( \text{row}++ \)  
        else  
            if \( (\text{row} + \text{col}) \& 1 == 0 \) then  
                \( d1 += 2 \)  
                if \( \text{row} >= n \) then \( \text{row} -= d1 \)  
                if \( \text{col} > 1 \) then \( \text{col}++ \)  
            else  
                \( d2 += 2 \)  
                if \( \text{row} >= n \) then \( \text{row} -= d2 \)  
                if \( \text{col} > 1 \) then \( \text{col}++ \)  
    i++  
end while  
return \( A \)

The working principle of ZZSW is as follows:  
Step 0: Initialize \( i=1 \)  
Step 1: Follow the Zigzag procedure until to reach bottom most left cell.  
Step 2: Move right once by incrementing column by 1.  
Step 3: Move to top right by decrementing row by i.
Step 4: Move to the bottom once by incrementing row by 1 until to reach the bottom side.
Step 5: Increment i by 1 and repeat step 3 and 4 until to reach the last cell

3.2 ZigZagRasterDiagonal Algorithm
In which, a 2-D square matrix is split into exactly two halves diagonally, and the upper left part is scanned in Zigzag fashion, and the lower right part is in a RasterDiagonal fashion as shown in figure 2b. The Zigzag pattern scans \((n^*n)/2\) + \((n/2)-1\) number of matrix locations whereas RasterDiagonal pattern scans \((n^*n)/2\) - \((n/2)+1\) amount locations where n is the total number of the rows or columns of a square matrix. The algorithm for the ZigzagRasterDiagonal is given below.

**Algorithm ZigZagRasterDiagonal**

<table>
<thead>
<tr>
<th><strong>Input:</strong></th>
<th>Q[i][i]: 2-D array, n: is the total number of rows or columns of a square matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output:</strong></td>
<td>A[ ]: 1-D array</td>
</tr>
</tbody>
</table>

row ← col ← 1
i ← t ← 0
N ← n*n
A[ ] ← a new 1-D array of size N
while i is less than N do
  A[i] ← Q[row–1][col–1]
  if i < [(N/2)+(n/2)-1] then
    if [(row + col) & 1 == 0] then
      if col < n then col++ else row += 2
      if row > 1 then row--
    else
      if row < n then row++ else col += 2
      if col > 1 then col--
    end if
  else
    if row < n then row++, col--
    else t ← col + 1, col ← row, row ← t
  end if
end while
return A

The functioning of ZZRD is given below:
Step 0: Initialize i=1
Step 1: Follow the procedure Zigzag until to reach the bottom most left cell.
Step 2: Move to top right by assigning incremented column value to row and old row value to the column
Step 3: Move to the bottom once by incrementing row by 1 and decrementing column by 1 until to reach the bottom side.
Step 4: Repeat step 2 and 3 until to reach the last cell
The grayscale image called blackhoodlady.jpg with varying dimensions of $2^n \times 2^n$ where $n=1, 2, 3..., 13$ is created. Then the pixel value of each dimensional image is extracted to construct a 2-D square matrix of sizes $2 \times 2$, $4 \times 4$, $8 \times 8$, $16 \times 16$, $32 \times 32$, $64 \times 64$, $128 \times 128$, $256 \times 256$, $512 \times 512$, $1024 \times 1024$, $2048 \times 2048$, $4096 \times 4096$ and $8192 \times 8192$ respectively. Such matrices are considered as data set for testing the algorithms.

4. ASYMPTOTIC BEHAVIOR OF THE ALGORITHMS
An asymptotic analysis is performed in order to predict the behavior of algorithms at the worst case; that is, for the largest input data, to compare the effectiveness of the algorithms and to determine, which is appropriate for an application. Let $f_1(n)$, $f_2(n)$ and $f_3(n)$ be the number of steps required to solve the problem of size $n$ for ZZ and ZZRD and ZZSW respectively. Growth rate of these functions is estimated in order to determine the effectiveness of an algorithm where we analyze the growth rate with respect to increase in the input size. In the worst case, it is calculated as, $f_1(n) = 14n + 8$ for ZZ, $f_2(n) = 19n + 8$ for ZZRD and $f_3(n) = 32n + 8$ for ZZSW respectively. In which, the behavior of the algorithm at upper bound is described using asymptotic notation namely, Big-O notation.

The function growth behavior, that is, running time complexity $t(n)$ required to solve a problem size of $n$ is measured equivalent asymptotically for all algorithms, since if we ignore the multiplicative constant in the higher order term and real constant, then the time complexity of these functions turns out to be $O(n)$. Intuitively, this is a linear growth in which $T_i(n)$, where $i=1, 2, 3$ increases directly with the size of the problem. The figure 3 illustrates the same where the problem size $n$ increases as 1, 2, 3, and so on regardless of the constant factors for all running time functions. When the problem size gets sufficiently large, the high and low order terms in complexity function do not matter. However, this means that two algorithms can have the same time complexity, even though one is always faster than the other. In this case, the constants and low order terms do matter in terms of which algorithm is actually faster.

Fig. 3. Asymptotic growth rate of functions
An algorithm with complexity \( f(n) \) is said to be not slower than another algorithm with complexity \( g(n) \) if \( f(n) \) is bounded by \( g(n) \) for large input size \( n \). That is, \( 14n+8 \leq c_1 g_1(n) \), if \( g_1(n) = n \), then lets \( c_1 = 16 \), and clearly, for \( n \geq 4 \), \( c_1 g_1(n) \geq f_1(n) \). As a fact of this, whenever \( n \) is big enough, \( c_1 g_1(n) \) will be bigger than \( f_1(n) \). In generic, as long as there is a \( c > 0 \), and \( n_0 \geq 0 \) such that \( c . g(n) \geq f(n) \) for all \( n \geq n_0 \), we say that \( f(n) = O(g(n)) \), that is, \( 14n+8 = O(n) \), \( 19n+8 = O(n) \) and \( 32n+8 = O(n) \). In mathematics, we say that \( f(n) = O(g(n)) \), iff \( \exists \ c > 0 \forall n, n \geq n_0, f(n) \leq c . g(n) \) \([9]\). Alternatively, \( O(g(n)) = \{ f(n) : \exists \ c > 0 \forall n, n \geq n_0, 0 \leq f(n) \leq c . g(n) \} \).

Informally, whenever \( n \geq 4 \) and \( c_1 = 16 \), the growth rate function for ZZ \( f_1(n) = 14n+8 \) falling behind \( c_1 \), \( g_1(n) \) at upper bound where \( g_1(n) = O(n) \) as shown in figure 4a, equally, whenever \( n \geq 3 \) and \( c_2 = 22 \), the function for ZZRD \( f_2(n) = 19n+8 \) falling behind \( c_2 \), \( g_2(n) \) at upper bound where \( g_2(n) = O(n) \) as shown in figure 4b. The same is true for function ZZSW, \( f_3(n) = 32n+8 \) which underneath \( c_3 \), \( g_3(n) \) at upper bound where \( c_3 = 36 \), \( n \geq 2 \) and \( g_3(n) = O(n) \) as shown in figure 4c.

Fig. 4. a) ZZ algorithm growth rate against \( g_1(n) \) b) ZZRD algorithm growth rate against \( g_2(n) \) c) ZZSW algorithm growth rate against \( g_3(n) \).

5. EMPIRICAL PERFORMANCE EVALUATION
A software model is developed to answer what is the effect on run time while increasing the input size by four times for each time. Furthermore, performance of algorithms at upper bound is measured and analyzed in specific machine model and implementation method. Because of the indistinguishable growth rate established for
all algorithms asymptotically, it is difficult to answer which one is better for the implementation than others are. The difference between algorithms with the same order of growth is usually a constant factor, at the same time the difference between a high-quality algorithm and a low-quality algorithm is unbounded since the relative performing algorithm's strength varies from hardware to hardware, the size of the problem and the details of data sets being used. By specifying a system model, analyzing the number of operations an algorithm requires under the given model and how well the actual running time of the algorithm is tracked can notably explore a solution to this problem. In view of this fact, we tested the algorithms on an Intel Pentium Dual Core micro architecture CPU T2370, 1.73 GHz with 2 GB RAM and two cores running Windows-8 Enterprise 32-bit operating system, x64-based processor. It has 2x32 KB L1 cache and shared 1 MB L2 cache for each core.

Algorithms are implemented in Java and benchmarking the algorithm is carried out. Benchmarking an algorithm can be performed by noting the starting and ending time of algorithm execution with respect to specific system time or by profiling the algorithm. We performed profiling the algorithm using Net Beans IDE 6.9.1 automatic profile tool under JDK 6 VM, CPU profiling Type, and Lazy instrumentation scheme settings. The algorithms are performing a variety of arithmetic, bit wise and logical operations during the execution for the given input are shown in figure 5. We found that ZZ is performing a less number of computations than ZZRD and ZZSW per cell. It is substantiated by quantitative data that except ZZ, the other two performs a nearly equal number of operations. We obtained that the number of operations performed per cell is quite fixed on average of 21.75 operations per cell for ZZRD and ZZSW respectively.

![Fig. 5. Number of operations performed by ZZ, ZZRD and ZZSW](image)

The experiment is repeated for several times with the same input and settings, and the average of the result is taken for consideration. The input size n, is categorized as small for n ≤ 5000, medium if n is in between 5000 and 10000 and large if n ≥ 10000.
The table 2 shows the quantitative measurements of the running times that we obtained for the algorithms.

**Table 2. Running Time measurement**

<table>
<thead>
<tr>
<th>Matrix Size $2^n$</th>
<th>Running Time (milliseconds)</th>
<th>ZZ</th>
<th>ZZRD</th>
<th>ZZSW</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^n$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0.006</td>
<td>0.008</td>
<td>0.008</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0.009</td>
<td>0.012</td>
<td>0.014</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0.01</td>
<td>0.014</td>
<td>0.024</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0.028</td>
<td>0.032</td>
<td>0.043</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0.105</td>
<td>0.125</td>
<td>0.139</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>0.478</td>
<td>0.45</td>
<td>0.561</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>1.69</td>
<td>1.86</td>
<td>3.4</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>2.81</td>
<td>4.68</td>
<td>7.28</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>8.5</td>
<td>10.6</td>
<td>11.1</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>19.4</td>
<td>47.5</td>
<td>47</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>194</td>
<td>271</td>
<td>282</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>2385</td>
<td>3330</td>
<td>3526</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>11054</td>
<td>15736</td>
<td>15869</td>
</tr>
</tbody>
</table>

**Fig. 6. Running time for ZZ, ZZRD and ZZSW**

The observation shows that for the small category of input data, running times taken by the algorithms ZZ and ZZRD are fairly consistent, on average about 0.11 milliseconds and for ZZSW, it is increased by two times of ZZ or ZZRD. For medium category, the running time ZZRD is increased by eight times of ZZ and for ZZSW, it is increased by nine times of ZZ and one time of ZZRD. For larger category, ZZRD has taken 42% more time than ZZ and for ZZSW, it is 44% more than that of ZZ and 2% more than that of ZZRD. Figure 6 which clearly reproducing the same; furthermore, if we extrapolate the data from this, one can note that $f_1(n)$ is substantially growing faster than $f_2(n)$ and $f_3(n)$, correspondingly, $f_2(n)$ growing...
faster than $f_1(n)$ in the upper bound and also equal growth rate for all functions at the lower bound. In other words, ZZ outperforms than other two for sufficiently large $n$ by taking the least number of computations to perform a task. The empirical study accordingly reflects the truth that what we have explored during asymptotic analysis.

6. RESULTS AND DISCUSSIONS

The effectiveness of scanning algorithm not only depends on its execution speed but also its high quality. The quality of the scanning algorithm is measured in terms of efficiency of grouping the high frequency ac coefficients together during the linearization, in other words, how much the algorithm is user friendly to RLE and encoding process. To determine the quality, the algorithms are introduced into JPEG image compression process and the results that we obtained are presented in Table 6. The Grayscale JPEG natural images with size 128 x 128 we used to test the algorithms are shown in figure 7.

Since the aim of the study is evaluating the user friendliness of various scanning algorithms in JPEG image compression, a 100% quality of the reconstructed image is not taken into consideration. Contrarily, we suggest considering quality parameters during the compression process, if quality is a matter.
6.1 Bit reduction performance measure
To verify the Bit Reduction (BR) performance of the proposed methods, we formulated the equation 6.

$$BR(\%) = \frac{X_{y_1} - X_{y_2}}{X_{y_1}} \times 100$$ (6)

Where $X$ is the total number of bits in the image after encoding by a method, $y_1$ is ZZ and $y_2$ can be either ZZRD or ZZSW. BR>0 indicates increasing the performance of a method, as if BR=0, no change in performance and if BR < 0 indicates decreasing performance of a method.

Fig. 8. a) Increasing order of high frequency ac coefficients in 8x8 quantized DCT matrix from bottom to top. b) Increasing order of high frequency of ac coefficients from top to bottom

All algorithms are resulting equal amount of bit reduction when the lower-right diagonal quarter in an 8x8 quantized DCT matrix contains zero coefficient distribution in all places regardless the order of coefficient distribution in the upper left diagonal quarter, and the same is true for those images containing complex texture. When the ac coefficients in the quantized DCT matrix are increasing diagonally and horizontally from lower side to upper side in the lower-right diagonal
quarter as shown in figure 8a, performance of ZZ is identical to ZZSW and better than ZZRD, contrarily, bit reduction performance of ZZRD is better to ZZ and alike to ZZSW whenever the ac coefficients are in pattern as shown in figure 8b. Furthermore, we noticed that ZZRD provides a good bit reduction performance compare to ZZSW for any image regardless of the shape or texture.

**Table 6. Bit Reduction Performance comparison of conventional and proposed methods**

<table>
<thead>
<tr>
<th>Image (Size: 128x128)</th>
<th>Conventional Method</th>
<th>Proposed methods</th>
<th>Bit reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ZZ (bits)</td>
<td>ZZRD (bits)</td>
<td>ZZSW (bits)</td>
</tr>
<tr>
<td>airplane</td>
<td>15230</td>
<td>15230</td>
<td>15230</td>
</tr>
<tr>
<td>baboon</td>
<td>16967</td>
<td>16967</td>
<td>16967</td>
</tr>
<tr>
<td>blackhoodlady</td>
<td>15474</td>
<td>15465</td>
<td>15489</td>
</tr>
<tr>
<td>boat</td>
<td>15913</td>
<td>15913</td>
<td>15913</td>
</tr>
<tr>
<td>cameraman</td>
<td>17683</td>
<td>17675</td>
<td>17778</td>
</tr>
<tr>
<td>coin</td>
<td>25295</td>
<td>25284</td>
<td>25477</td>
</tr>
<tr>
<td>einstein</td>
<td>15208</td>
<td>15208</td>
<td>15208</td>
</tr>
<tr>
<td>gallopinghorse</td>
<td>22968</td>
<td>22855</td>
<td>23421</td>
</tr>
<tr>
<td>horse</td>
<td>20860</td>
<td>20853</td>
<td>20865</td>
</tr>
<tr>
<td>lenna</td>
<td>17147</td>
<td>17144</td>
<td>17158</td>
</tr>
<tr>
<td>prague</td>
<td>22812</td>
<td>22790</td>
<td>22873</td>
</tr>
<tr>
<td>shannon</td>
<td>17136</td>
<td>17136</td>
<td>17136</td>
</tr>
<tr>
<td>statue</td>
<td>13861</td>
<td>13861</td>
<td>13861</td>
</tr>
<tr>
<td>william</td>
<td>22946</td>
<td>22942</td>
<td>23061</td>
</tr>
<tr>
<td>wilson</td>
<td>18461</td>
<td>18447</td>
<td>18527</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

†indicates the best result

From table 6 we observed that the method ZZRD gave bits reduction about 0.059 percentage improvements on average and 0.492 percentage bit reduction at maximum compared to ZZ. We also confirmed that the method ZZRD results superior performance for any image compared to the conventional method ZZ in all instances.

**7. CONCLUSIONS**

Linear representation of the coefficients is a decisive point for efficient encoding in JPEG image compression. In this paper, we have focused on how efficiently ac coefficients can be represented linearly. We have proposed two novel scanning algorithms namely ZZRD and ZZSW and are independently used in the place of conventional method in JPEG Image Compression process. In case of ZZRD, linearization is considerably improved, and it competes successfully with the classical approach. It results (0.059% - 0.492%) improvements in bit reduction than the
conventional method. In case of ZZSW, it offers an identical performance for the times.

8. ACKNOWLEDGEMENTS
The authors wish to express sincere thanks to the organizations and individuals who have allowed us to use their images in this paper free of charge. They include the following: wikipedia.org, deviantart.com, onsecretethunt.com, indiancoinsinfo.blogspot.in, and picturesofalberteinstein.com.

9. REFERENCES
A Comparative Study: Data Compression on TANGLISH Natural Language Text

S Sankar
Associate Professor, KCG College of Technology
Research Scholar, Hindustan University
Chennai, Tamilnadu, India

Dr. S Nagarajan
Professor and Head of Department of IT
Hindustan University
Chennai, Tamilnadu, India

ABSTRACT
In this age of information and in the era of distributed on-line and mobile computing, one thing is on the rise at an exponential rate is storage space for information. Growing office automation, digitizing libraries, on-line business transactions, and Meta data storage we need a huge storage space. Since more and more new users become a part of the Internet society the significance of data transmission develops to a great extent as never prior to. If data to be stored or transmitted represented efficiently this can be conquered. Data compression techniques are playing a vital role in representing the information. This paper investigates the use of lossless data compression on the Tanglish language text and compares the performance based upon Huffman coding.

Keywords
Lossless Data compression, Huffman coding, and Tanglish Language.

1. INTRODUCTION
Data compression is a method of bit reduction technique that uses a smaller amount of bits to represent information. The purpose is to reduce the amount of memory space, transmission bandwidth required. Also, it helps to increase data transmission rate over wired or wireless networks. On the other side, compressed data must be decompressed in order to make use of them. Compression schemes are generally classified into lossless and lossy. In lossless compression schemes what is compressed can be recovered without any loss of information. For that reason, they are appropriate to compress only textual data where meaning and clarity of the information is greatly anticipated. With lossy compression schemes, there will be some loss of information during decompression that is acceptable or unnoticed. So they are suitable for processing audio and video files where loss of resolution is ignored, depending on the preferred quality. The method characterized in this study is a kind of lossless compression used to compress a plain text.

A plain text is a form of highly unformatted text usually consists of alphanumeric and control characters. These characters are represented either in fixed length or variable length in the form of binary numbers 0 and 1 while storing and transmitting them. ASCII code and Unicode are fixed length coded character set tables that comprise numbers, letters, punctuation and various typographic and mathematical symbols and other characters. Each character in the set is represented by unique binary numbers. ASCII stands for American Standard Code for Information Interchange contains a set of 128 characters where the capital letter ‘A’ has decimal number 65 and is not stored as it is rather as 1000001. A character in the ASCII table has 7 bit length.

Unicode is a Universal character set table contains 65535 characters that cover almost all the characters, punctuations, and symbols in the world. UTF-8 (Unicode Transformation Format in 8 bits) is a type of Unicode character set where each character has 1 to 4 bytes long, for example, a Tamil language character ‘உ’ (read as ‘a’) has decimal value 2949 and is stored as 11100000 10101110 10000101. In a fixed length code, each character has the same length, so it is possible to calculate exactly where each character begins therefore it is quicker to find a particular character for the decompression but it occupies more memory.

Consider a message M having symbol set {abcaabaaab}. If it is represented in fixed length code as a=00, b=01 and c=10 then each bit string is a codeword for a symbol. Now the message can be encoded as 0001100000011000000000101100 where the total number of bits is 26. If a message contains n number of distinct symbols then each symbol will require exactly \[\log_2 n\] length of bits. In the example just given, number of distinct symbols in the message is n=3, as a result we need \[\log_3 3\] = 2 bits to represent each symbol. Grouping of n bit length from the beginning of the encoded binary string will result a symbol. By scanning from the beginning the first two bits 00 results a symbol ‘a’ and the next two bits 01 result a symbol ‘b’ and so on. If n bits are required to encode a symbol, then 2n distinct symbols can be encoded. In ASCII, each character has exactly 7 bit length, so that there are 27=128 distinct symbols. Total number of bits required to encode complete message can be computed as, length of the message * number of bits per symbol. Reducing the length of the code is very important since the amount of time required for the data transmission always proportional to the number of bits required to encode it.

A variable length code comes with the solution for this where each character will be represented with different length of code. Suppose the symbols represented as a=0, b=10, and c=01 then the entire message is encoded as 0100100100100 101001 where the total number of bits are 20 which is 23.08% less compared to fixed length variable code. But the problem with this is how to recognize the end of one codeword and the beginning the next one during decode. The first bit is 0 and the next bit is 1 so whether to decode it as 0 which is ‘a’ or as 01 which is ‘c’. Thus this is ambiguous. To prevail it a prefix code is followed. A prefix code is a variable length code it uses prefix rule where no codeword is a prefix of another. Once a certain bit pattern is assigned as codeword of a symbol, no other codeword should start with that bit pattern. If a bit pattern 0 was assigned as the codeword of ‘a’, then no other codes could start with 0. Therefore the codeword for the symbol ‘c’ should not be 01 rather than it can be 11. If so, all codeword can be unambiguously decodable since once we get...
a match, there is no longer codeword that can also match. Various lossless variable length code algorithms have been proposed and used. Some of the techniques in use are the Huffman Coding, Run Length Encoding, Arithmetic Encoding and Dictionary Based Encoding [8]. This paper studies the efficiency of codeword created for the Tanglish language text using Huffman coding.

2. HUFFMAN CODING
The Huffman code algorithm generates a prefix and variable length codeword for a symbol based on the symbol probability distribution \( p_i \), where \( i = 1, 2, 3 \ldots n \). The frequency distribution of all the symbols of the source is calculated in order to calculate the probability distribution. According to the probabilities, the codeword for each symbol are assigned. It assigns shorter codeword for higher probability symbols and longer codeword for smaller probability symbols [8]. This algorithm is optimal in the sense that the average number of bits required to represent a symbol is minimized, subject to the constraint that the codeword satisfies the prefix rule, as defined above. Average number of bits required to encode the message can be computed as,

\[
L = \sum_{i=1}^{n} l_i \cdot p_i
\]

where \( l_i \) is the codeword length of a symbol and \( p_i \) is the probability of a symbol [13]. The compression ratio as a measure of efficiency has been considered and can be calculated as, Compression ratio= \( \frac{\text{Compressed file size/Source file size}}{100} \% \) [8].

The idea behind the algorithm is, first construct an optimal binary tree so-called Huffman tree which adopts a greedy approach. The greedy method suggests construct a solution through a sequence of steps, considering one input at a time. At each step, make a locally optimal choice among the currently available all feasible choices; once made it cannot be changed on subsequent steps and that choice may lead to the development of the globally optimal solution. An optimal merging pattern is followed to construct the Huffman tree in which sort the symbols in increasing order based on their probabilities \( p_1 \leq p_2 \leq \ldots \leq p_n \). At each step merge two smallest probability symbols together. If any two symbols have equal probabilities, interchange them based on their appearance in the ASCII or Unicode table. When more than two sorted symbols are to be merged together, the merge can be accomplished by repeatedly merging sorted symbols in pairs. The leaf nodes represent the given symbols and are called as external nodes. The remaining nodes are called as internal nodes. Each internal node has exactly two children and its value is obtained by merging the probabilities of its two children. Tree is built in a bottom up fashion [14].

2.1 Algorithm for Huffman Tree

**Input:** Symbol set with their respective probability distribution. (Probability of a symbol=frequency of a symbol / total number of symbols in the message)

**Output:** Huffman Tree \( T^* \)

1. Sort the symbol set based on their probability in non-decreasing order.
2. Construct a forest tree \( F \) for the given symbol set where each tree having only one node include the symbol and its probability.
3. Repeat \{ 
   3.1 Choose the nodes with the minimum and next to minimum probabilities respectively
   3.2 Create a new node \( T^* \). Node value for \( T^* = \text{sum} (\text{minimum probability node, next to minimum probability node}) \)
   3.3 Attach a minimum probability node on left to \( T^* \) and next to the minimum probability node on right to \( T^* \)
   3.4 Assign 0 to left branch and 1 to the right branch.
   3.5 Sort the tree \( F \) in non decreasing order
4. } Until (no more than one node tree in \( F \))
5. Now \( F \) has only one tree \( T^* \). Output \( T^* \).

2.2 Encoding
A path from root of \( T^* \) to the corresponding leaf node defines codeword for a particular symbol. Right margins should be justified, not ragged.

2.3 Algorithm for Decoding

**Input:** Codeword generated during encode

**Output:** Symbol

1. Start from the root of the tree \( T^* \).
2. Examine the first bit in the input
3. If it is 1, move to the left child.
4. If it is 0, move to the right child.
5. If it is leaf node then output its symbol.
6. If it is not a leaf node then
   6.1 Examine the next bit in the input
   6.2 Go to step 3 and proceed

3. TANGLISH

Tanglish is a dialect in which a sentence is formed in Tamil borrows words from English. It is an informal language. If it is difficult to find appropriate Tamil words in writing or verbal communication, identical word in English will be used in place. Now it is becoming a fashionable language used by Tamil speaking people in Tamilnadu, a southern state of India. It is a new hybrid language where a sentence is formed by mixing of linguistic features of both Tamil and English even though both has a different syntactical pattern, has found its way into the media - electronic and print [7].

A sentence in Tanglish language can be constructed in three different forms:

**Form 1:** Inserting English word in the place of the Tamil word in a sentence usually found in Tamil magazine and oral communication

**Form 2:** Representing English word in Tamil usually found in web pages

**Form 3:** Representing Tamil word in English frequently used in SMS (Simple Message Service) messages and web pages

4. PROPOSED APPROACH

This paper applies the Huffman code technique on the above said forms and compares the compression ratio to determine which form is an efficient. A sentence, “car is breakdown” is taken as a sample message for the demonstration. This sentence can be written in Tanglish as:

**Form 1:** car breakdown aagivitathu
**Form 2:** kaar piraeugtvon aagivitathu
**Form 3:** vakanam paLuthagi vitathu

The following section illustrates the creation of Huffman tree and the codeword for these different forms of representation.
4.1 Data compression on Form 1

Figure 1 shows Huffman tree generated for the message “car breakdown aagivitathu”. Table 1 shows the codeword generated for the same.

The encoded message is now,

\[ \text{11000010010001100101101110100110111011010011011011101101111110} \]

Average number of bits required to encode = \( \frac{5}{28} \times 2 + \frac{3}{28} (3 + 3 + 4 + 4) + \frac{1}{28} (5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5) \)

= \( \frac{98}{25} = 3.92 \)

Space required for actual message= 25 x 8 = 200 bits

Space required for encoded message= 98 bits

Compression ratio= 49%

Table 1. Codeword for Form 1 message

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
<th>Probability</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>5</td>
<td>5/25</td>
<td>01</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
<td>2/25</td>
<td>000</td>
</tr>
<tr>
<td>i</td>
<td>2</td>
<td>2/25</td>
<td>1000</td>
</tr>
<tr>
<td>t</td>
<td>2</td>
<td>2/25</td>
<td>1101</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>1/25</td>
<td>10010</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
<td>1/25</td>
<td>10011</td>
</tr>
<tr>
<td>v</td>
<td>1</td>
<td>1/25</td>
<td>10100</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>1/25</td>
<td>10101</td>
</tr>
<tr>
<td>w</td>
<td>1</td>
<td>1/25</td>
<td>10110</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>1/25</td>
<td>10111</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>1/25</td>
<td>11000</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1/25</td>
<td>11001</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td>1/25</td>
<td>11100</td>
</tr>
<tr>
<td>k</td>
<td>1</td>
<td>1/25</td>
<td>11101</td>
</tr>
<tr>
<td>u</td>
<td>1</td>
<td>1/25</td>
<td>11110</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1/25</td>
<td>11111</td>
</tr>
</tbody>
</table>

4.2 Data compression on Form 2

Figure 2 illustrates the Huffman tree generated for encoding the message “kaar piraegtovun aagivitathu”. Table 2 information is used to encode this message. The encoded message is,

\[ \text{1010000000111101101011111011100110011011111101111011} \]

Average number of bits required to encode = \( \frac{6}{25} \times 2 + \frac{3}{25} (3 + 3 + 4 + 4 + 4 + 4) + \frac{1}{25} (5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5) \)

= \( \frac{100}{28} = 3.57 \)

Table 2. Codeword for Form 2 message

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
<th>Probability</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>6</td>
<td>6/25</td>
<td>00</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>3/25</td>
<td>010</td>
</tr>
<tr>
<td>i</td>
<td>3</td>
<td>3/25</td>
<td>011</td>
</tr>
<tr>
<td>g</td>
<td>2</td>
<td>2/25</td>
<td>1000</td>
</tr>
<tr>
<td>u</td>
<td>2</td>
<td>2/25</td>
<td>1001</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
<td>2/25</td>
<td>1100</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
<td>1/25</td>
<td>10101</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>1/25</td>
<td>10110</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>1/25</td>
<td>11011</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1/25</td>
<td>11100</td>
</tr>
<tr>
<td>k</td>
<td>1</td>
<td>1/25</td>
<td>11101</td>
</tr>
<tr>
<td>u</td>
<td>1</td>
<td>1/25</td>
<td>11110</td>
</tr>
</tbody>
</table>

4.3 Data compression on Form 3

Figure 3 shows the Huffman tree constructed for the message “vakanam paluthagi vitathu”. Table 3 shows the codeword generated for the same. The actual message is now encoded as,

\[ \text{1110101100011010111010011001110111010011001110000110101101111011} \]

Average number of bits required to encode = \( \frac{6}{28} \times 2 + \frac{3}{28} (3 + 3 + 4 + 4 + 4 + 4) + \frac{1}{28} (5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5) \)

= \( \frac{87}{25} = 3.48 \)

Space required for actual message= 28 x 8 = 224 bits

Space required for encoded message= 100 bits

Compression ratio= 44.64%

Table 3 shows the codeword generated for the above message.

Table 3. Codeword for Form 3 message

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
<th>Probability</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>7</td>
<td>7/28</td>
<td>00</td>
</tr>
<tr>
<td>r</td>
<td>5</td>
<td>5/28</td>
<td>010</td>
</tr>
<tr>
<td>i</td>
<td>5</td>
<td>5/28</td>
<td>011</td>
</tr>
<tr>
<td>g</td>
<td>4</td>
<td>4/28</td>
<td>1000</td>
</tr>
<tr>
<td>u</td>
<td>4</td>
<td>4/28</td>
<td>1001</td>
</tr>
<tr>
<td>r</td>
<td>4</td>
<td>4/28</td>
<td>1100</td>
</tr>
<tr>
<td>p</td>
<td>2</td>
<td>2/28</td>
<td>10101</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
<td>2/28</td>
<td>10110</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>2/28</td>
<td>11011</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>2/28</td>
<td>11100</td>
</tr>
<tr>
<td>k</td>
<td>2</td>
<td>2/28</td>
<td>11101</td>
</tr>
<tr>
<td>u</td>
<td>1</td>
<td>1/28</td>
<td>11110</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1/28</td>
<td>11111</td>
</tr>
</tbody>
</table>

4.4 Data compression on Form 4

Figure 4 shows the Huffman tree generated for the message “vakanam paluthagi vitathu”. Table 4 shows the codeword generated for the same. The actual message is now encoded as,

\[ \text{1110101100011010111010011001110111010011001110000110101101111011} \]

Average number of bits required to encode = \( \frac{6}{25} \times 2 + \frac{3}{25} (3 + 3 + 4 + 4 + 4 + 4) + \frac{1}{25} (5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5) \)

= \( \frac{100}{28} = 3.57 \)}
Space required for the actual message
= Total number of symbols * 8 bits per symbol
= 25 x 8
= 200 bits
Space required for encoded message= 87 bits
Compression ratio= 43.5%

Table 3. Codeword for Form3 message

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
<th>Probability</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>6</td>
<td>6/25</td>
<td>01</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>3/25</td>
<td>100</td>
</tr>
<tr>
<td>h</td>
<td>2</td>
<td>2/25</td>
<td>000</td>
</tr>
<tr>
<td>u</td>
<td>2</td>
<td>2/25</td>
<td>1100</td>
</tr>
<tr>
<td>i</td>
<td>2</td>
<td>2/25</td>
<td>1110</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
<td>2/25</td>
<td>1111</td>
</tr>
<tr>
<td>k</td>
<td>1</td>
<td>1/25</td>
<td>10100</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>1/25</td>
<td>10101</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
<td>1/25</td>
<td>10110</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td>1/25</td>
<td>10111</td>
</tr>
<tr>
<td>l</td>
<td>1</td>
<td>1/25</td>
<td>11100</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>1/25</td>
<td>11101</td>
</tr>
</tbody>
</table>

5. RESULTS AND DISCUSSION

Ten different plaintext files written in English with different sizes are considered. The file sizes are 2524 bytes, 2023 bytes, 5334 bytes, 2867 bytes, 4109 bytes, 2878 bytes, 4745 bytes, 5189 bytes, 5432 bytes, and 4901 bytes respectively. The content of each file is translated into above mentioned three different forms in Tanglish language and Huffman coding is applied on each of them and the results are compared.

5.1 Results

Table 4 shows Huffman coding results where it summarizes the results of average number of bits required to encode and compression ratio for each file. According to the results shown in Table 4, for file 1 and 10 the algorithm generates higher compression ratio in Form1. This happens due to the direct use of actual English words in the place of Tamil words at high rate. The files 4 and 6 has lower compression ratio due to lesser use of actual English words.

Table 4. Huffman coding results

<table>
<thead>
<tr>
<th>File</th>
<th>Average number of bits required to encode</th>
<th>Compression ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Form 1</td>
<td>Form 2</td>
</tr>
<tr>
<td>F1</td>
<td>3.92</td>
<td>3.57</td>
</tr>
<tr>
<td>F2</td>
<td>3.50</td>
<td>3.53</td>
</tr>
<tr>
<td>F3</td>
<td>3.55</td>
<td>3.53</td>
</tr>
<tr>
<td>F4</td>
<td>3.32</td>
<td>3.45</td>
</tr>
<tr>
<td>F5</td>
<td>3.95</td>
<td>3.91</td>
</tr>
<tr>
<td>F6</td>
<td>3.21</td>
<td>3.35</td>
</tr>
<tr>
<td>F7</td>
<td>3.87</td>
<td>3.84</td>
</tr>
<tr>
<td>F8</td>
<td>3.84</td>
<td>3.72</td>
</tr>
<tr>
<td>F9</td>
<td>3.90</td>
<td>3.87</td>
</tr>
<tr>
<td>F10</td>
<td>4.00</td>
<td>3.98</td>
</tr>
</tbody>
</table>

For all other files the compression ratio is almost same range in Form1. In Form2, the files 5, 7, 9 and 10 has higher compression ratio where it is found a large amount of actual English words are written in Tamil words. In Form3, most of the file has a lower compression ratio since average number of bits required to encode message is less for each file than for the same in Form 1 and 2.

5.2 Discussion

From the figure 4, it has been found that compression ratio gradually decreasing in the order of Form1>Form2>Form3 and compressing the files saves the disk space and transmission time. Also, we found from Table 1, 2 and 3 that Huffman code generates a short codeword for the symbol which has higher probability and lengthy codeword for the symbol which has lower probability.

6. CONCLUSION

A lossless data compression algorithm is carried out on different file. Each file is translated in three different forms of Tanglish language and Huffman coding is applied on each of them. The resulting compression ratios are compared. We can observe that often placing directly actual English words in the place of Tamil words in a Tanglish Language sentence turn out a better result than either representing the English words in Tamil or Tamil words in English in terms of data compression. Also, it is observed that Huffman code produces average number of bits required to encode a message is higher for Form1 than others.
7. REFERENCES


Encryption-Then-Compression System for Image using Prediction Error Clustering and Modified Flood fill Algorithm

S Sankar  
Research Scholar @ Hindustan University  
Associate Professor @ KCG College of Technology  
sankar@kcgcollege.com

S Nagarajan  
Professor @ Hindustan University  
snagarajan1960@gmail.com

M Veena  
PG Scholar @ KCG College of Technology  
veenakarthik29@gmail.com

Abstract—In modern world, encryption and compression plays a vital role. Encryption can be an important tool to help improve security. The second one is a compression which consists in reducing the size of information to be transmitted, by removing redundant information of them. In previous paper, compression is performed by arithmetic coding, efficiency still unsatisfactory. This paper deals with the problem of how to design a pair of system with better efficiency. In proposed method, encryption is operated in prediction error domain using random permutation to achieve high level of security and compression is operated via run length encoding and modified flood fill algorithm. This algorithm is newly proposed here, which is adopted to find similar characteristics of pixels. The Experimental results are demonstrated to compare with the state-of-the-art method in terms of compression efficiency.

Index Terms—Encryption, Compression, Random Permutation, Run Length Encoding, Modified Flood fill algorithm.

I. INTRODUCTION

With the rapid development of multimedia and network technologies, the security of multimedia becomes more and more important, since multimedia data are transmitted over open networks more and more frequently. Typically, reliable security is necessary to content protection of digital images. Encryption can be defined as the art of converting data into coded form which can be decode by intended receiver only who poses knowledge about the decryption of the ciphered data. Encryption can be applied to text, image, and video for data protection. Image compression is an application of data compression that encodes the original image with few bits. The objective of image compression is to reduce the redundancy of the image and to store or transmit data in an efficient form.

In Compression-Then-Encryption (CTE) paradigm, compression is performed before encryption. In that scenarios, Encryption algorithm may be remove the compressed bits in the image, so looks like encrypted only. To get efficient system, the order of applying the compression and encryption needs to be reversed, also should meet all the requirements in secure transmission. A big challenge within such Encryption-then-Compression (ETC) framework is that compression has to be conducted in the encrypted domain, as network provider does not access to the secret key K. In [13] introduced the basic concept of data compression which is applied to modern image and video compression techniques such as JPEG, MPEG, MPEG-4 and so on. Depends upon [2] and [6] the possibility of encrypted signals operated in the encrypted domain directly. Although number of number of signal samples can be packed together and process them as a unique sample is proposed in [4]. In this paper [2] investigating the implementation of discrete Fourier transform (DFT) in the encrypted domain by using the homomorphic properties of the underlying cryptosystem. Encryption algorithm [robust] is implemented using stream cipher. Xinpeng Zhang et al, also investigated about encryption phase, the original pixel values are masked by a modulo-256 addition with pseudorandom numbers that are derived from a secret key. According paper [6] possible to protect the private data against the service provider while preserving the functionality of the system. In addition [10] block ciphers operating in various chaining modes and are considered it is shown how compression can be achieved without compromising security of the encryption scheme. Also look over about, if data encrypted with block ciphers can be compressed without access to the key. The possibility of compressing encrypted grey level and color images, by decomposing them into bit-planes also has been discussed in [11]. Wei Liu et al., showed that encrypted images can be compressed progressively in terms of resolution. The decoder observes statistics of resolution, by using that information can be improved quality. Samir Kumar Bandyopadhyay et al., concentrated on the lossless compression of image using approximate matching technique and run length encoding. The performance of this method is compared with the available jpeg compression technique over a wide number of images, showing good agreements. The present work focuses on improving compression efficiency by applying same run length encoding technique.

II. EXISTING SYSTEM

In [1], Encryption is achieved via random permutation and compression technique is adaptive arithmetic coding. In terms of compression efficiency is slightly worse than the state-of-the-art lossless/lossy image coders, which take original, unencrypted images as inputs. First encryption consists of five stages. GAP is applied to whole image according to CALIC.
[8], then estimated values are mapped in the range [0-255] explained in [11] and [1].

The algorithmic procedure of performing the image encryption is in [1]. At first all the mapped prediction errors \( e_{i,j}' \) of the whole image \( I \) should be calculated. The prediction errors divide into \( L \) clusters \( C_k \) for \( 0 \leq k \leq L - 1 \), and each \( C_k \) is formed by concatenating the mapped prediction errors in a raster-scan order. Prediction errors in each \( C_k \) reshaping into a 2-D block having four columns and \( |C_k|/4 \) rows, where \( |C_k| \) denotes the number of prediction errors in \( C_k \). Performing two key-driven cyclical shift operations to each resulting prediction error block, and read out the data in raster-scan order to obtain the permuted cluster \( C'_k \).

Let \( CS_k \) and \( RS_k \) be the column secret key vector, Row Secret key in \( C_k \). Here, \( CS_k \) and \( RS_k \) are obtained from the key stream generated by a stream cipher, using different keys for same sessions. The assembler concatenates all the permuted clusters \( C'_k \), for \( 0 \leq k \leq L - 1 \), and generates the final encrypted Image. An adaptive AC is then employed to losslessly encode each prediction error sequence \( C'_k \) into a binary bit stream \( B_k \). The compression of the encrypted file \( I_e \) needs to be performed in the encrypted domain, as network provider does not have access to the secret key \( K \). The joint decompression and decryption is performed over the received bits to get reconstructed image, \( I' \).

III. PROPOSED SYSTEM

In this ETC system encryption is conducted prior to compression shown in figure (1). Here organized as encryption, compression, sequential decompression and decryption.

A. Encryption

This Encryption phase consists of six distinct stages. First input image (size 256×256) is divided into 32×32 block, called image segmentation. Each block is treated separately, applying GAP algorithm to each and every pixel in blocks. GAP-Gradient-Adjusted Prediction is estimated, this tries to detect how rapidly the edge is changing around pixel and then by classifying the tendency of edge changing into sharp, normal, and weak edge. Depends upon the CALIC [8], \( \Gamma_{ij} \) is estimated. Prediction error values \( e_{i,j} \) are calculated as follows.

\[
e_{i,j} = I_{i,j} - \Gamma_{i,j}
\]

For 8-bit images, the prediction error \( e_{i,j} \) can potentially take any values in the range \([-255, 255]\), it can be mapped into the range \([0, 255]\). The mean is calculated on known pixels; already know that the difference assumes value in the range \([-\text{mean}, 255-\text{mean}]\). If \(255-\text{mean}\) is greater than 127, the interval \([128, 255-\text{mean}]\) can be mapped in the interval \([-127, -\text{mean}-1]\). More specifically, if \( I_{i,j} \leq 128 \), we rearrange the possible prediction errors according to [7] each of which is sequentially mapped to a value between 0 to 255. Sequentially mapped prediction error values are reshaping into 2-D block and then ZZRD scanning is applied, to form clusters. To reduce computation complexity need to increase the number of clusters. Cluster numberings are varies from \(0 \leq k \leq L-1 \), where \( k \) represents last cluster in the process and \( L \) represents total number of clusters in the encryption operation. Number of clusters directly proportional to the image size. The Scanning algorithm ZZRD is a 2-D square matrix is split into exactly two halves diagonally, and the upper left part is scanned in Zigzag fashion, and the lower right part is in RasterDiagonal fashion slightly better than traditional zigzag, which is proved in [ZZRD].

The proposed permutation algorithm is performed over the clusters from \( C_0 \) to \( C_{L-1} \). Random key is genetated according to the size of the cluster. Finally permutation process is performed with key generation and encryption algorithm. Specifically, permutation-based image encryption approach conducted over the prediction error domain. Depends on [10] encryption is achieved using two cyclical shift operations in the clusters. Here pixel values are shuffled randomly using constant key values, resulted in modified locations not values of pixels. Permutated clusters called \( C'_k \). Finally encrypted image is called \( I_e \). Number of prediction errors equals that of the pixels, the file size before and after the encryption remains same. Due to the high sensitivity of prediction error sequence against disturbances, reasonably high level of security could be retained. Further compression, decryption and decompression carried out over the encrypted image results. Performance results are demonstrated to compare efficiency.
B. Compression

The encrypted Image (Ie) is again segmented into 32×32 sub blocks. Modified flood fill algorithm Here Run Length Encoding (RLE) technique is adopted for image compression. RLE is one of the lossless image compression techniques, which requires linear input. But in our image having 2D block, so modified flood fill algorithm is used to find same pixels as well as similar characteristics of pixels.

1) Modified Flood fill Algorithm

In segments pixels can be divided into top left corner, top right corner, bottom left corner, bottom right corner, left column, right column, and center pixels. Specifically, circular directions are also considered here. Eight directions namely N (North), NE (Northeast), E (East), SE (Southeast), S (South), SW (Southwest), W (West), NW (Northwest). Each direction can represent in binary values from 0, 00, 000, 0000, 00000, 000000, 0000000, and 00000000. Uncompressed bits can be represent in 1’s.

Scanning from top left corner pixel, considering in eight directions and search whether pixel value is identical or not. If pixel value is same as root pixel means that it will be stored that assigned direction. Sequentially that is converted into a corresponding binary value and flood fill algorithm produces sequential output. Furthermore, RLE is applied over these results to produce compressed bitstreams. Run Length Encoding is calculating the repeated runs and produces reduced bits, to enhance storage performance as well as increase the transmission speed of the multimedia communication.

Consider the top left corner pixel where location is (1,1), first scanning to north direction it comes to the bottom left corner pixel (32,1). If that pixel is identical to root pixel, it will stored pixels direction called N, make that direction as mark as visited. Again it checks with the North pixel’s eight directions, it checks with the order like N, NE, E, SE, S, SW, W, NW. If any of the pixel match with the root value store that direction, as well as stored that pixel as mark visited. Pixels are scanning according to circular shift format (clockwise direction). Otherwise it comes to the next direction NE, root pixel is moving to scan the north East pixel location is same pixel, because circular shift is comes to the same direction only. If root pixel is same with that location, it will stores the direction. Next scan moves to East pixel which comes to location (1,2), identical values stores the direction, it continues the scanning procedure until same pixel values comes under in mark as visited category. Next it moves to SouthEast direction called (2,2). Similarly it checks out with identical values. Afterwards comes to South(S) direction in circular shift moves to location (2,1). Then continues checking with southwest direction pixel which means that scan going to same pixel. According to west, scans top right corner pixel location is (1,32). Immediately it goes to NorthWest (NW) direction if any pixel not match with root value. scanning the bottom right corner pixel location is (32,32). If corresponding value is same it will stores the direction, then continues scanning with matched pixel values, otherwise continues the process according to procedure. This scanning procedure is applied to all of the pixels in each segments. After stores the directions, it should be converted into corresponding binary values. Here each segment consists of 1024 pixels. Every pixel scanning 8 directions in unique order, parallely stores the directions which is same with root value. If not match with root pixel values take moves to next directions, from left to right move and top to right move which appropriately suite this algorithm. Once complete the scanning in region 1 it should take the backtracking. Finding previous location of pixel which is stored like mark as visited then continuing the scanning of pixel where we resumed it off. Again it checks the pixel with root values once it match means check in 8 directions. After completing the scanning, in other words when all pixels are marked or comes to deadend position have to stop the scanning.
All directions are converted into binary values, these are compressed using run length encoding, where repeated binary values are compressed easily.

2) Run Length Encoding (RLE)

The Lossless Run Length Encoding (RLE) is a very simple form of data compression in which runs of data are stored as a single data value and count, rather than as the original run. Lossless methods are normally used when we can not afford any data loss. In this work, modified flood fill algorithm output is in binary format, consists of repeated string values. RLE replaces a string of repeated symbols with a single symbol and a count indicating the number of times the symbol is repeated. RLE can compress any type of data regardless of its information content, but the content of data to be compressed affects the compression ratio.

Consider a character run of 15 'A' characters which normally would require 15 bytes to store AAAAAAAAAAAAAAAAAA is stored as 15A. With RLE, this would only require two bytes to store, the count (15) is stored as the first byte and the symbol (A) as the second byte. The concept of run length is also used because using run length a row of image can be represented using much less literals than the original. This is most useful on data that contains many such runs: for example, simple graphic images such as icons, line drawings, and animations.

Algorithm Modified Flood Fill

Input: a[m][n]: 2-D array, m: is the total number of rows or columns of a square matrix.

Output: P[1]: 1-D array.

for row is from 1 to m
for col is from 1 to n
if row==1 && col==1 % Top Left corner pixel
if a(row,col) is equal to a(m,col) % North
P{row,col}{1}="N";
Mark as visited;
else a(row,col) is equal to a(row,col+1) % East
P{row,col}{2}="E";
Mark as visited;
else a(row,col) is equal to a(row+1,col) % South
P{row,col}{3}="S";
Mark as visited;
else a(row,col) is equal to a(row,m) % West
P{row,col}{4}="W";
Mark as visited;
else a(row,col) is equal to a(m,m) % North West
P{row,col}{5}="NW";
Mark as visited;
else a(row,col) is equal to a(m,m) % North East
P{row,col}{6}="NE";
Mark as visited;
else a(row,col) is equal to a(row+1,col+1) % South West
P{row,col}{7}="SW";
Mark as visited;
else a(row,col) is equal to a(m,m) % South East
P{row,col}{8}="SE";
Mark as visited;
end if
end if
end if
endfor
endfor

C. Joint decompression and decryption

In receiver side, compressed bit streams can be received with side information about segmentation size and key information. With using side information authorized receiver can be decompress the image in order to get original size. At first, run length decoding should be carried out to get the original bitstream. Reverse of the modified flood fill algorithm is applied over the results of run length decoding. In particular pixel values can be placed in correct positions. Finally encrypted pixel values can be obtained from this process. Reverse of Encryption process is applied to the retrieved pixels. According to the side information, receiver divides information into L segments, each of which is associated with a cluster of prediction errors. For each Bi, run length decoding can be applied to obtain the corresponding permuted prediction error sequence Cki. As receiver knows the secret key K, the corresponding de-permutation operation can be employed to get back the original Cki. With all the Cki, the decoding of the pixel values can be performed in a ZZRD-scan order. For each location (i, j), the associated error energy estimator Δi,j and the predicted value I' i,j can be calculated from the causal surroundings that have already been decoded. The first unused prediction error in the kth cluster is selected as e i,j, which will be used to derive e i,j according to I' i,j and the mapping rule described in Section III-A. The reconstructed pixel value can then be computed by

\[ I_{i,j} = I'_{i,j} + e_{i,j} \]  

As the predicted value I' i,j and the error energy estimator Δi,j are both based on the causal surroundings, the decoder can get the exactly same prediction I' i,j. In addition, in the case of lossless compression, no distortion occurs on the prediction error e i,j, which implies \( I_{i,j} = I_{i,j} \), i.e., error-free decoding is achieved.

IV. RESULTS AND DISCUSSIONS

In this section, the security of our proposed image encryption and the compression performance on the encrypted data are evaluated experimentally. In Figure3 illustrate original and encrypted images, which we can see that our encryption approach is effective in destroying the semantic meaning of the images. Due to pixel based image encryption can attain more level of security, where pixel locations are shuffled repeatedly. In compression part, two parameters (the PSNR and iterations in this case) are used to measure the performance of Compression how much is achieved.
Each sample in the diagram is produced by setting up the compression algorithm to compress the image at a specified quality. For instance, the quality parameter could be an integer ranging from 1 to 100. With quality 1, the algorithm would compress the image as much as possible. With quality 100, the algorithm would compress the image with minimum or no loss of information. Proposed compression algorithm requires only small number of iterations to reach high PSNR value. Fig 6 Experimental result shows that asymptotic behavior of the proposed system is better than previous system. Iterations showing the number of execution which is required and trying to attain high compression metric in proposed method. Number of pixels measured in bits per pixel is compressed highly in proposed system.

In state-of-the-art system CALIC performance is less compared to ETC. Fig 7 showing Image ID is in X-axis and percentage of bpp is plotted in Y-axis. So Experimental results shows that proposed system ETC is compressed efficiently. In proposed system we can able to compress maximum number of bits in encrypted image. The compression efficiency of our proposed method applied to the encrypted images is compared with the lossless rates given by the latest version of CALIC, a benchmark of practically good lossless image codecs, a state-of-the-art lossless compression approach on encrypted images. In Fig. 8, we also compare the rate-PSNR performance of our compression method. For bit rates above 2 bpp, our method achieves even higher PSNR values than JPEG 2000. The gain in PSNR over JPEG 2000 can be significant for high bit rates. As bit rate drops, the PSNR gain over JPEG 2000 decreases. When the bit rate is below 2 bpp, the PSNR gain over JPEG 2000 diminishes and starts to become negative. When the bit rate is around 2.50 bpp, the PSNR gain can be over 10 dB for the input image. We also notice that the method of scalable seems to suffer from the problem of performance saturation for images with intensive activities such as Harbor, Bridge in [14].

V. Conclusion

Image encryption and compression is an extremely important part of modern computing. This work proposed a novel idea for encrypting image and designed a practical scheme made up of image encryption and compression. In the encryption phase of the proposed system consists of, predictor called GAP, it had two potential improvements one is truly direction adaptive another one is from local to nonlocal prediction. Next is scanning method ZZRD is newly introduced here. It has performed better than traditional zigzag scan method. In permutation based image encryption, only the pixel positions are shuffled and the pixel values are not masked. The key generation process is unique and is a different process.
The disturbance of the strong correlation among the adjacent pixels assures high security of the images. This is obtained with the help of permutation process. In Compression stage, modified floodfill algorithm and run length encoding is used. Newly introduced modified floodfill is considering circular shift directions to find neighbourhood pixels. By finding connected pixels similar regions can be grouped together. Due to simplicity of RLE, Compression performance could be increased. This method can be extended in trying to handle multiple images instead of single image by another predictor called GED, and another scanning method ZZSW, Image Encryption Using Block-Based Transformation Algorithm. Extending compression part by using another lossless compression algorithm.

REFERENCES
[14] www.mathwork.org