2008

Using a novel variable block size image compression algorithm for hiding secret data

F. Keissarian
University of Wollongong in Dubai, farhadk@uow.edu.au

Using a novel variable block size image compression algorithm for hiding secret data

Abstract

This paper presents a data hiding scheme that hides the data in the compression domain of a digital image. A quadtree decomposition algorithm decomposes the host image into blocks of variable sizes according to histogram analysis of the block residuals. Variable block sizes are then encoded at different rates based on their visual activity levels. The majority of secrete data are embedded into the smooth area of the image, while a small portion of the secret data are hidden in the compression codes of the high detailed blocks. Experimental results confirm that the proposed scheme can embed a large amount of data in the compressed file while maintaining satisfactory image quality.

Disciplines

Physical Sciences and Mathematics

Publication Details

Keissarian, F. (2008). Using a novel variable block size image compression algorithm for hiding secret data. IEEE International Conference on Signal Image Technology and Internet Based Systems, 2008. SITIS 2008. (pp. 285-292). United States: IEEE.

Using a Novel Variable Block Size Image Compression Algorithm for Hiding Secret Data

Farhad Keissarian
University of Wollongong in Dubai
farhadkeissarian@uowdubai.ac.ae

Abstract

This paper presents a data hiding scheme that hides the data in the compression domain of a digital image. A quadtree decomposition algorithm decomposes the host image into blocks of variable sizes according to histogram analysis of the block residuals. Variable block sizes are then encoded at different rates based on their visual activity levels. The majority of secrete data are embedded into the smooth area of the image, while a small portion of the secret data are hidden in the compression codes of the high detailed blocks. Experimental results confirm that the proposed scheme can embed a large amount of data in the compressed file while maintaining satisfactory image quality.

1. Introduction

Message transmissions over the Internet still have data security problem, and secure communication methods are needed for transmitting messages over the Net without anything leaking out. One of the possible ways to accomplish this is to embed secret data inside a cover carrier that appears meaningful but not important. Through this way, the existence of the secret data can be concealed and the attention of attackers will be avoided. Data hiding techniques that follow this strategy are called 'methods of steganography [1].

Data hiding represents a class of processes used to embed data, such as copyright information, into various forms of media such as image, audio, or text with a minimum amount of perceivable degradation to the "host" signal; i.e., the embedded data should be invisible and inaudible to a human observer. Once data embedding is used for secret transmissions, two crucial requirements must be satisfied. One is minimizing the degradation in image quality after secret data is embedded. The other is making the hiding capacity of cover media as large as possible. Basically, the image quality of stego-image and hiding capacity are tradeoffs, so the capacity often causes greater distortion in the cover medium, and vice versa.

Digital data hiding techniques can be roughly classified into three kinds: spatial domain methods, frequency domain methods and compression domain methods. The difference here is the domain where the embedding happens. Methods in the spatial domain work by directly replacing the raw data of the digital image with secret data. One simple and well-known example of such schemes is the least-significant-bit (LSB) hiding [2]. As the name implies, it replaces the least significant bits of each pixel value in the original image with secret data bit by bit. Different from spatial domain methods, frequency domain methods first transform an image from the spatial domain to the frequency domain by using discrete cosine transform, discrete Fourier transform or discrete wavelet transform. And then, the secret data is hidden inside the transformed coefficients [3].

In recent years, some researchers have concentrated on a third possibility: embedding secret data into the compression domain. Image compression used in some of these studies includes vector quantization (VQ), and block truncation coding (BTC). Digital images are stored or transmitted in a compressed form so as to minimize memory space consumption or to deal with the limited-bandwidth problem. For this reason, if secret data can be directly embedded into the compressed codes of the image, then we can spare all those decompression and recompression processes. Furthermore, the compressed codes transmitted on the Net attract less attention than the raw data itself .

Side match vector quantization (SMVQ) with the concept of prediction, proposed in [4] applies an

978-0-7695-3493-0/08 $25.00 © 2008 IEEE
DOI 10.1109/SITIS.2008.82

adaptive data hiding scheme to maintain the visual quality of the stego-images. Two thresholds are used to adaptively select enough qualified smooth blocks from the cover image to hide secret data. However, the image quality of the stego-image in this scheme depends to a large extent on the number of smooth blocks in a cover image. In addition to the VQ compression domain, data embedding schemes based on BTC has also been proposed. The proposed method in [5], applies a 4 × 4 BTC to compress a gray scale cover image directly. Therefore, the output of BTC-encoded block contains two quantization values and one bitmap. A pre-defined threshold is then set to classify the type of each BTC-encoded block as smooth or complex. Subsequently, the secret data is embedded into the bitmap of the selected BTC-encoded blocks.

In this paper, we develop an image hiding scheme that can hide the secrete data into compression codes of the host image, generated by a quad-tree based compression technique which uses a pattern matching algorithm for coding of the image blocks. The proposed method is a combination of the works we reported earlier in [6] and [7].

In this study, a quadtree segmentation is employed to encode the image blocks of variable sizes at different rates according to the level of activity inside the block. Low activity blocks (uniform blocks) can be encoded by the block average while high activity blocks (edge blocks) are coded by a set of pre-defined block patterns. A new decision, based on the distribution of the block residuals and the shape of their histogram determines whether or not the processed block needs further divisions. The key point of the proposed scheme is to embed the majority of the data in the smooth area of the host image. These include the uniform blocks of size 16x16, 8x8, and also 4x4 pixels. A small portion of the secret bits are also embedded into the edge blocks.

The rest of the paper is organized into four sections. The concept of the proposed quadtree based compression algorithm is introduced in section 2. In section 3, the proposed hiding scheme is presented. Experimental results are given in section 4, and finally some conclusions are made in section 5.

2. Quadtree-based Compression Algorithm

Quadtree is a well-known data structure, which can be used to describe the spatial information of an image. It is also used in image compression techniques and image database systems. Quadtree segmentation decomposes an image into variable block size. The segmented block is then coded at a different rate according to the level of activity inside the block.

2.1. Quadtree Decomposition

A main point of quadtree segmentation is the evaluation criterion of image segmentation. This criterion evaluates the presence of image information and decides the division into sub quadrants. In quadtree decomposition, a judgment is first made to see whether a block can be represented by a single gray value or whether it must be divided into four subblocks. If the block is to be divided, then the same decisions are applied to four subblocks as well, to determine whether it needs further divisions, and so on.

In this paper, the decision method that we presented earlier in [7] is implemented in dividing process of quadtree. The method operates based on the distribution of the block residuals and determines whether the processed block needs further divisions. This is accomplished by classifying a block either as a low-detail (uniform) or as a high-detail (edge) block. The classifier employs the residual values of a block and classifies the block according to the shape of the histogram of the residuals. The classification is carried out through a peak detection method on the block histogram. A brief description of the classifier is as follows.

Each block of *n x n* pixels is converted into a residual block by subtracting the sample mean from the original pixels. Here, two of the most important local characteristics of the image block are considered*: central tendency*, represented by the mean value and the *dispersion* of the block samples about the mean, which is represented by the residual values. The challenge here is to analyze the dispersion of the residual values about the mean. One way of achieving this to sort the histogram of the block residual samples.

As the neighboring pixels in the original block are highly correlated, the residual samples will tend to concentrate around zero. One can then quantize the residual samples prior to forming the histogram. Based on the distribution of the residual samples within the test images, we choose to apply a coarse quantization, in particular a 15-level non-uniform quantizer. We now define q_j as the output of the quantizer with index *j*, as shown in Fig.1. The histogram of the quantized values, $h(q_j)$ may then be formed to

provide the frequency of the q_j .The quantized residual histogram (QRH) is then analyzed by simply detecting its peaks. In the resulting histogram, if $h(q_j) > h(q_{j-1})$ and $h(q_j) > h(q_{j+1})$ then a peak at index *j* is detected. A minimum score, $Score_{\min}$ for $h(q_j)$ can be defined below which a peak at index *j* is not detected. Based on the formed histograms of test images, we have chosen to use a $Score_{\min}$ of 80% of the total number of samples n^2 for the block size of *n x n* pixels. A peak on the histogram indicates a high score of residual values, therefore it is fair to conclude that there is a considerable number of pixels that have the same dispersion about the block mean. According to the number of detected peaks on the histogram, image blocks can be classified as either low-detailed or high-detailed blocks. A histogram with a unique peak at its center (uni-modal histogram) identifies a low detailed (uniform) block. The existence of two peaks or more implies that the processed block is a high detailed block.

In the decomposition approach, an image to be coded is first divided into blocks of 16x16 and then each block is repeatedly divided into four equal quadrants, if its residual histogram is not a uni-modal type. On the other hand, the decomposition process will stop if the residual histogram of the block has a dominant peak at its center. This block is regarded as a uniform block and all the pixels in the block will be represented by the block mean. If the smallest block size of 4x4 is reached and its residual histogram is still not a uni-modal type, it is regarded as an edge block. Fig. 2 depicts an example of a 4x4 uniform block and its histogram.

Since variable block sizes are used in quadtree segmentation, decoding of transmitted images requires the information about the size and location of each block. That is, if a block is divided into smaller blocks, the quadtree code is "1." Otherwise, the quadtree code is "0." When the block size is 4 x 4, if the block is a high-detail block, then the quadtree code is "1" Otherwise, the quadtree code is "0." This amounts to too much overhead information needed for transmission. To overcome this problem, we use the method presented in [8] which introduces 17 possible combinations within a 16x16 image block. Only a 6-bit binary sequence ($D^0D^1d^1d^2d^3d^4$) is required to represent each splitting mode as shown in Fig 3. The first bit D^0 indicates whether or not the 16x16 block is partitioned into four 8x8 blocks. If $D^0 = 1$, then the other four bits d^1, d^2, d^3, d^4 are required to indicate

j	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
q_j	-67	-52	-39	-28	-17	-10	-5	0	5	10	17	28	39	52	67

Fig. 1 : The quantizer output with index j

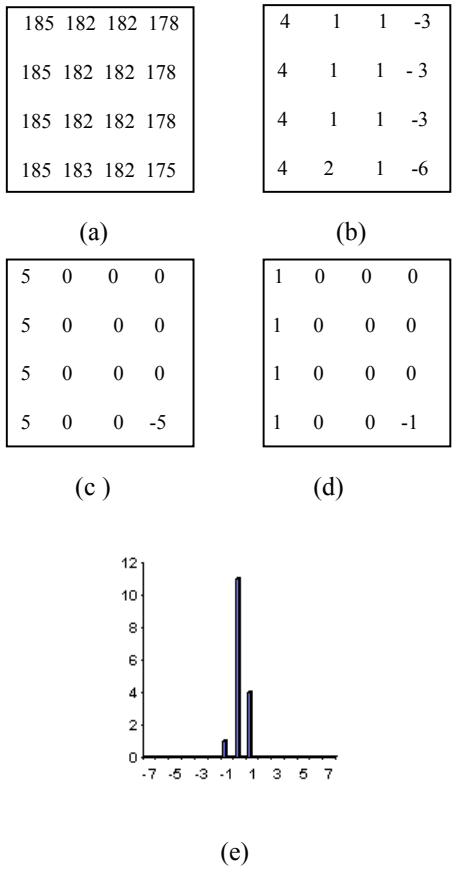

Fig. 2 : QRH of 4x4 uniform block : a) original block, b)Block residuals, c) quantized residuals, d) Quantizer index , e) uni-modal histogram of the block

whether to split each 8x8 block into four 4x4 blocks or not. The amount of side information is calculated as $\frac{MxN}{16x16} * 6$ bits for an $M\ x\ N$ image size. Since the total number of bits in the image is $(WxH) * 8$, the overhead will be around 0.003 bits per pixel (bpp) which is significantly small.

2.2 Coding of Edge Blocks

The uniform blocks of different sizes from 4 x 4 pixels to 16 x 16 pixels are coded by the block mean. To preserve edge integrity, a 4x4 edge block is coded by one of the finite number of pre-defined patterns that

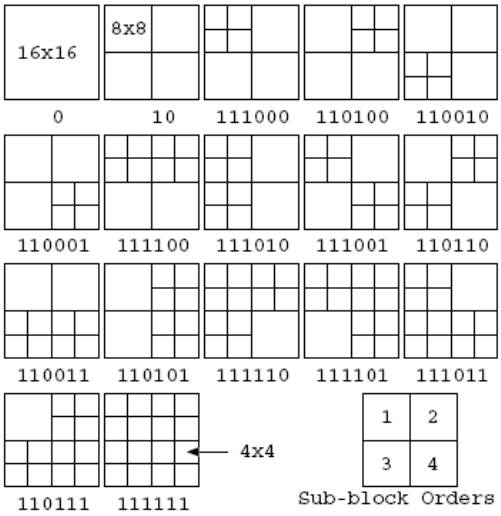

Fig 3: All possible 16 x 16 block partitioning modes and their associated binary representation

best matches its features. The two distinct peaks of the QRH of a 4x4 edge block are represented by two representative intensities. These are the block low and high intensities, denoted by I_{low} and I_{high}. By forcibly clustering all pixels in an edge block into two groups, a bi-level approximation of the block may be obtained. Only two representative intensities and certain binary bits, forming a bit-map are necessary to specify the bi-level representation. Once the representative intensities of an edge block have been determined, a bit- map may be constructed to specify the correspondence between the pixels and the representative intensities. In such a bit-map, each pixel is represented as a '1' or a '0'. The detailed description is given simply as:

$$B_{i,j} = \begin{cases} 1 & x_{i,j} \le I_{mid} \\ 0 & x_{i,j} > I_{mid} \end{cases}$$

Where, $I_{mid} = \dfrac{I_{low} + I_{high}}{2}$

Once the bit-map of an edge block has been formed, the block can be coded by finding the best match for its bit-map from a set of patterns in a look-up table. A set of 30 patterns (Fig. 4) which preserve the location and polarity of edges in four major directions is used for the pattern matching stage. This process determines the index of the matched pattern selected from the set. The matching score P_{score} of each pattern is calculated as:

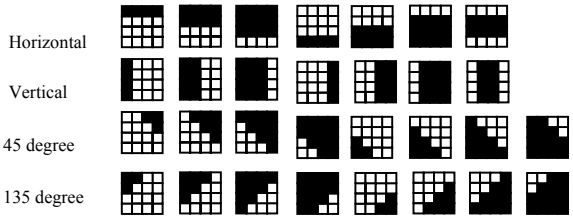

Fig. 4: Set of pre-defined patterns

$$P_{score} = \sum_{i=0}^{3}\sum_{j=0}^{3} \begin{cases} 1 & if \quad P_{i,j} = B_{i,j} \\ 0 & if \quad P_{i,j} \neq B_{i,j} \end{cases}$$

The pattern *p*, which has the maximum score, P_{score} is selected as the pattern with the best match. The transmitted information includes the index P_{best} of the selected pattern in the look-up table as well as the block representative intensities I_{low} and I_{high} for the areas indicated in black and white of the selected pattern, respectively. During the decoding process, the decoder replaces the pixels of a uniform block by the block mean. Whereas, in decoding an edge block, the decoder uses the index of the selected pattern as well as the transmitted intensities to reconstruct the block.

Reconstruction of an edge block is carried out by replacing the 1's and 0's of the selected pattern by I_{high} and I_{low}, respectively. The number of bits to code an edge block *edge B* is computed as:

$$B_{edge} = 2B_{rep} + \log_2^P + 1$$

where, B_{rep} denotes the number of bits required to code one of the representative intensity, *P* is the number of patterns used, $\log_2^p$ is the number of bits required to transmit the index of P_{best}, and the 1 is the overhead to inform the decoder the block is an edge block. If B_{mean} is the number of bits required to code the mean value of a uniform block, the achievable bit rate in bpp for an 8-bit grey level image may then be determined according to:

$$bpp = \frac{(N^{16x16} + N^{8x8} + N_{uni}^{4x4}) * B_{mean} + (N_{edge}^{4x4} * B_{edge})}{MxN * 8}$$

Where, N^{16x16} : number of 16x16 blocks, N^{8x8} : number of 8x8 blocks, N_{uni}^{4x4} : number of 4x4 uniform blocks and N_{edge}^{4x4} : number of 4x4 edge blocks. The 0.003 bpp for the quadtree overhead is added to the above calculation.

3. The Data Hiding Scheme

This section demonstrates how to embed the secrete bits into a gray level host image and how to extract the data. The whole process can be partitioned into two phased: one is the data embedding phase; the other is the data extraction phase.

3.1. Data Embedding Phase

In our proposed scheme, the data are embedded into the compression domain of a cover image. To enhance the hiding capacity of the compression domain of the host image, the secrete messages are sequentially embedded into the bitmap of uniform blocks [5] as well as into the representative intensities of the edge blocks.

In the data embedding phase, the host image is compressed using the quadtree-based compression algorithm, described in the previous section. An ownership file is then constructed according to the splitting modes of 16x16 image blocks. The ownership file is a bit stream where every set of 6 bits represent the splitting mode of the processed 16x16 block.

To lower the distortion, we embed the majority of secrete data into the bitmap of uniform blocks of different sizes, that is 16x16, 8x8 and 4x4 uniform blocks. This way, the bitmap of a 16x16 block is replace by 256 secret bits, whereas the bitmaps of 8x8 blocks and uniform 4x4 blocks are replaced by 64 and 16 secrete bits, respectively. Because the pixel intensities in a uniform block are close to their neighboring pixels, even though the bits in the bitmap are changed, the reconstructed pixel value is still close to its original one.

In the coding algorithm described in the previous section, the pixels of a uniform block are represented by a single value that is the block mean. Therefore the block pattern will not have 1's and 0's. However, in order to embed secrete binary data into uniform blocks the block mean is tuned to produce two intensities as follows : $I_{mean} \pm \delta$, where δ is a small tuning value [6]. The bitmap of a uniform block is then given by :

$$B_{uniform} = \begin{cases} 0 & \text{for } x_{i,j} = I_{mean} - \delta \\ 1 & \text{for } x_{i,j} = I_{mean} + \delta \end{cases}$$

The sequence of embedded positions are from left to right and then up to down, which is in the row-major order. For the sake of data security, secret data should be encrypted before hiding. In our scheme, the population of uniform blocks in an encoded image is an important parameter to balance between the embedding capacity and the image quality of the stego-image. A larger population of the uniform blocks provides high embedding capacity but lower image quality of the stego-image, and vice versa.

The edge blocks can also be used for data embedding. In our approach, the LSB of the two representative intensities of an edge block are replaced by two secrete bits. The embedding capacity *EC* in bits is calculated by :

$$EC = (N^{16x16} * 256) + (N^{8x8} * 64) + N_{uni}^{4x4} * 8) + (N_{edg}^{4x4} * 2)$$

Here, the secret length *L* should be smaller than or equal to *EC*.

Data Embedding Algorithm:

Input : A grey-level host image *H* of *N x M* pixels, Total number of secrete bits *L*.

Output : A compressed image file H' , carrying the ownership file *O* of size $\frac{MxN}{16x16} * 6$ bits

Step 1. Apply the quadtree decomposition to the blocks of on the host image *H*.

Step 2. Construct the ownership file *O* by determining the splitting mode for each block of 16x16 pixels.

Step 3. Calculate the embedding capacity *C*. If *C* < *L*, then decrease the $Score_{\min}$ and go back to step 1.

Step 4. Form the bitmaps of uniform blocks of different sizes by setting a block pixel $I_{i,j}$ to '1' if its value is grater than the block mean I_{mean} ; otherwise, set it to '0'.

Step 5. Embed the secret bits into the bit map of each uniform block.

Step 6. Transmit the block mean and also the tuning value δ.

Step 7. Embed 2 secret bits in the LSB of the pair (I_{low}, I_{high}) of each 4x4 edge block

Step 8 . Transmit the representative intensities and the index of the matched pattern

Step 9. Repeat the above steps until *L* secret bits are totally embedded.

An example of the embedding procedure is shown in Fig. 5.

3.2. Data Extraction Phase

The extraction of data is relatively simpler than the embedding phase. For each 16x16 block, the corresponding bit sequence in the ownership file is examined. All the uniform blocks are then transformed into a bit plane. All the secret bits can then be retrieved from the bit plane. The data extracting order is by row-major policy and is and the process is repeatedly executed for all uniform blocks until *L* secret bits are retrieved. The extracted information need to be decrypted to gain the secret data.

Data Extracting Algorithm:

Input : The compressed image file H' and the ownership file *O*

Output : *L* secret bits

Step 1. Use the ownership file to identify the splitting mode of a processed 16x16 block.

Step 2. Form the bit planes of all the uniform blocks within the splitting mode and retrieve all the 0s and 1s of the bitplanes.

Step 3. Retrieve the LSB of the representative intensities of all the edge blocks.

Step 4. If *L* secrete bits are not retrieved yet , go to step 1.

4. Simulation Results

We have evaluated the performance of the proposed coding scheme through a computer simulation. The computer simulation has been carried out, using a set of 256x256, 8-bit intensity, monochromatic standard images including the images of 'Lena' (Fig.6a).

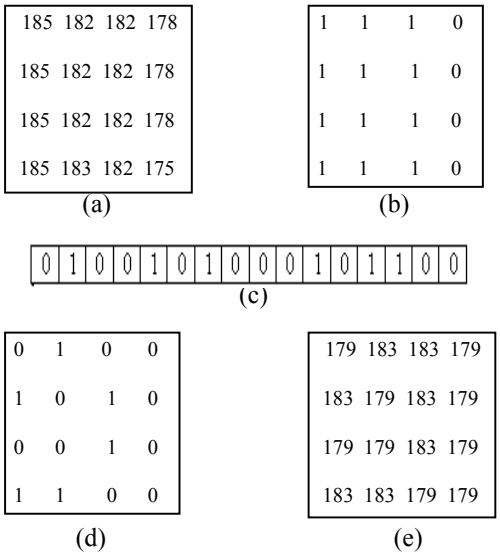

Fig. 5 : (a) original uniform block, mean value :181

(b) block bit map,

(c) the first 16 secret bits

(d) the modified bitmap,

(e) the reconstructed block after embedding the secret bits (mean =181 and $\delta = 2$)

In the implementation used here, the $Score_{\min}$ was set to two different values: 80% and 70% of the total samples in each block. That is for an 80% score case at least 205 samples, for a 16x16 block, 52 samples for an 8x8 block,, and 13 samples for a 4x4 block should form the height of central peak on the histogram of block residuals. Fig. 6.b and Fig. 6d illustrate the quadtree segmented images of 'Lena' for different $Score_{\min}$ values.

The hiding capacity depends highly on the identification of the uniform blocks of different sizes in the host image, which in turn is the important factor for the compression ratio. Table 1 shows the population of different blocks in the quadtree segmented image of Lena for both tested values of. $Score_{\min}$. Table 2 presents the embedding capacity of different types of blocks and the total embedding capacity.

For coding the edge blocks, The index P_{best} was coded by 5 bits (indicating the use of 30 patterns), and 8 bits were used to transmit each of the intensity values, I_{mean}, I_{low}, and I_{high}. Figs. 6c and 5.6

Fig. 6(a) : Original Image of Lena (256x256)

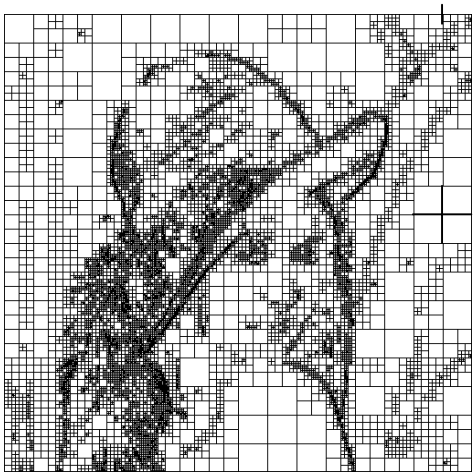

Fig. 6(b) : Segmented Image of Lena, $Score_{min}$ = 80%

Fig. 6(c) : Compressed Stego-Image , $Score_{min}$ = 80%

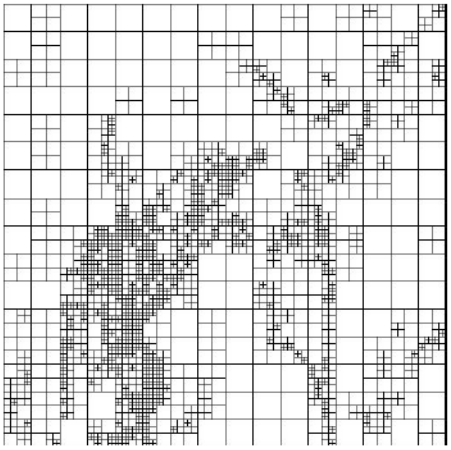

Fig. 6(d) : Segmented Image of , $Score_{min}$ = 70%

Fig. 6(e) : Compressed Stego-Image , $Score_{min}$ = 70%

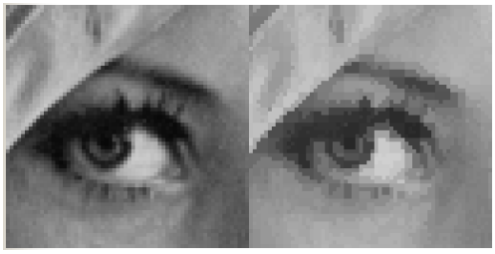

Fig 6(f) : The magnified versions of Figs 6(c) and 6(e)

illustrate the compressed stego-images for both cases. To evaluate the performance of the proposed scheme in terms of quality, we use the peak signal-to-noise ratio (PSNR) in order to measure the distortion between the host image and the processed image embedded with secret bits. It is defined as follows :

$$PSNR = 10x\log_{10}\frac{255^2}{MSE}dB$$

Where, MSE is the mean-square error.

Table 3 shows the compression results, the embedding capacity (EC) and the image quality (PSNR) for the host images 'Lena' The results show that, setting a higher value for $Score_{\min}$ will result in less number of uniform blocks being identified. This in turn leads to lower compression ratio and smaller embedding capacity, but a better image quality. The embedded capacity takes into account the transmission of the ownership file, as the ownership file is transmitted through the overheads.

6. Conclusions

In this paper, we have proposed an information hiding scheme to hide secrete data into compression domain of the host image, generated by a quadtree decomposition and a pattern-based compression algorithm. The majority of secrete data are embedded in the bitmaps of the uniform blocks of variable sizes of the host image. A small portion of the secret data are also embedded in the high detailed area of the host image. The hidden data can be extracted directly without decompressing the stego compressed file. The ownership file which is constructed during the compression phase is the key to extract the information at the receiver end.

6. References

[1] Petitcolas, F.A.P., Anderson, R.J., and Kuhn, M.G.: 'Information hiding – a survey', *Proc. IEEE*, 1999, 87, (7), pp. 1062–1078

[2] Chan, C.K., and Cheng, L.M.: 'Hiding data in images by simple LSB substitution', *Pattern Recognit*., 2004, 37, (3), pp. 469–474.

[3] Bao, P., and Ma, X.: 'Image adaptive watermarking using wavelet domain singular value decomposition', *IEEE Trans. Circuits Syst. Video Technol*., 2005, 15, (1), pp. 96–102.

Table 1
Population of blocks in the segmented image of Lena

$Score_{\min}$	**Total**	P_{16x16}	P_{8x8}	P_{4x4}^{uni}	P_{4x4}^{edg}
80 %	2986	% 1.4	% 5.2	% 33.1	% 60.3
70 %	2761	% 1.9	% 6.5	% 33.7	% 57.9

Table2
Embedding capacity for the segmented image of Lena

$Score_{\min}$	**Embedding Capacity (Kb)**				**Total**
	Uniform			*Edge*	
	16x16	8x8	4x4	4x4	
80 %	1.4	1.2	1.9	0.4	4.9
70 %	1.7	1.4	1.8	0.4	5.3

Table3
Compression and Embedding Capacity results for Lena
Image size : 64 Kb (256 x 256)

$Score_{\min}$	**CR**	**C I**	**EC**	**PSNR**
80 %	10.5 : 1	6.1 (Kb)	4.9 (Kb)	30.14
70 %	11.5 : 1	5.6 (Kb)	5.3 (Kb)	29.43

CR : Compression Ratio, *CI* : Compressed Image, *EC* : Embedding Capacity

[4] Shie SC, Lin SD, Fang CM (2006) Adaptive data hiding based on SMVQ prediction. IEICE Trans Inform Syst E89-D(1):358–362

[5] J-C Chuang and C-C Chang, "Using a simple and fast image compression algorithm to hide secret information", *International Journal of Computers and Applications*, Vol. 28, No. 4, 2006, pp. 329-333

[6] Keissarian, F. :'An information hiding scheme using a pattern-based compression algorithm', *Proceeding of the 3rd IEEE International Conference on Signal Image Technology & Internet Based Systems,* Dec. 2007, Shanghai, China, pp. 890-898.

[7] Keissarian, F.: "Novel quad-tree predictive image coding technique using pattern-based classification, *Proc. SPIE, Visual Communications and Image Processing (VCIP-2003*), vol. 5150, pp. 1481-1490, June 2003, Lugano, Switzerland.

[8] Dai, W.,*et al*. : 'Adaptive block-based image coding with pre-/post-filtering', *Proceeding of the Data Compression Conf.,* March 2005, pp. 73-82.