SN Computer Science (2020) 1:309
https://doi.org/10.1007/s42979-020-00322-9

ORIGINAL RESEARCH

Recursive Quad-Tree Block Partitioning for Data Embedding in Images

Tamer Shanableh[1]

Received: 16 June 2020 / Accepted: 5 September 2020 / Published online: 24 September 2020

Abstract
In this paper, we propose to embed messages in raw images using a recursive block partitioning technique adopted from HEVC video coding technology. This work introduces a quad-tree partitioning solution in which square blocks of pixels are partitioned recursively according to a homogeneity criterion. As such, large block sizes are allocated for homogenous areas of the image and smaller block sizes are used elsewhere. For this to work, the variable block sizes should cover the whole image without gaps. Additionally, the criterion used for determining the block partitioning should work for both message embedding and message extraction. As such, the extractor process can reproduce the same variable block sizes. Matrix encoding is used for message embedding and extraction, as such, a maximum of one-pixel value is changed per block. The proposed solution is assessed in terms of percentage of changed pixels, PSNR, SSIM, histogram changes and blind steganalysis. Comparison with existing work reveals that the proposed solution reduces both the image distortions and pixel change rates. With embedding rates ranging from 2.5 to 38%, the average PSNR of the cover images ranged from 72 to 60 dB, and the average normalized histogram differences ranged from 3 to 12 pixels. It is also shown that the proposed solution is less detectible when tested with blind steganalysis in comparison to existing solutions where the average detection accuracy is 49.8%.

Keywords Image processing · Data embedding · Video coding

Introduction

Data embedding in visual multimedia content has a number of important applications including digital rights management, quality assessment and error correction of streamed visual content, hiding patient records in medical images and convert conversations in general.

Data embedding can be applied to raw and compressed images, video and audio files. It can also be applied to encrypted multimedia files and in all cases the data can be embedded in plain or in encrypted formats.

The challenge in the message and data embedding solutions is to minimize both detectability and distortions caused to the cover visual content due to data embedding. If the cover content is compressed such as JPEG and MPEG files, then data embedding shall be implemented such that both the quality and the file size of the cover content is not affected much.

Additionally, data embedding can be applied in the spatial, frequency or wavelet domains of multimedia content. It can also be applied to syntax elements of compressed content such as quantization scales, motion vectors and coding modes.

The work reported in [1] showed that data can be embedded in compressed video by modifying the values of motion vectors. The work in [2] extended this solution to multilayer video coding where the motion vectors of the base and enhancement layers are modified to carry message data. As such, the data embedding rate is substantially increased with minimal distortions caused to the cover video. Data can also be embedded in compressed video by modifying the quantization scales of the frequency domain coefficients [3]. Likewise, partitioning modes and split decisions of HEVC videos, are used for data embedding [4] and [5].

Compressed JPEG images are also used as a cover for data embedding as reported in [6] and [7], so are compressed JPEG2000 images [8].

The solution proposed in this work deals with uncompressed natural images. A dominant work focusing on this

✉ Tamer Shanableh
tshanableh@aus.edu

[1] Department of Computer Science and Engineering, College of Engneering, American University of Sharjah, Sharjah, UAE

research aspect appeared in [9] where a novel edge detection solution is proposed such that data can be embedded in edge and around areas. Uncompressed or raw images were divided into non-overlapping blocks of 3×3 pixels where corner pixels are used for edge detection resulting in embedding message bits in up to 40% of image pixels. A similar solution was used for embedding encrypted patient records in MRI images [10].

Similar solutions exist for gray and binary images, for instance, a binary image steganographic solution based on syndrome-trellis code is reported in [11]. In [12], values of pixel pairs are used to search for a coordinate in the neighborhood set according to a given message digit, replacing the pixel pair by the searched coordinate is then used for message embedding. In [13], a tree-based parity check method is presented. It was proposed to find stego objects using a majority vote approach with the aim of reducing image distortion.

Adaptive data embedding in grayscale images is also reported in the literature. For instance, [14] proposed an adaptive k-bit LSB steganographic solution that uses pixel-value differencing where more message bits are embedded in edge areas. Similarly, [15] proposed an edge adaptive scheme that selects embedding areas according to both the existence of sharp edges and the length of the message to be embedded. The work in [16] proposed to partition the cover image into non-overlapping 1×3 blocks. The difference between the middle pixel value and other pixel values in the block are used to determine the number of secret bits to be embedded in the two pixels.

Steganalysis was also proposed for detecting data embedding in grayscale images. For example, [17] proposed an adaptive steganalytic solution where pixels are assigned different weights during feature extraction with the aim of focusing on the areas that have been possibly modified. And [18] proposed a steganalytic solution based on B-Spline fitting that achieves excellent results for detecting stego images with low embedding rates.

Recently, many image-based steganography solutions have been reported in the literature. For example, in [19] a data hiding scheme using a mini-Sudoku matrix is proposed in which each non-overlapping pixel pair can embed four bits. The first two bits are mapped based on a specific element in the mini-Sudoku matrix and the other two are represented by the horizontal and vertical coordinates of this element with the modulus function. In [20], data is embedded in red and blue channels of color images. Consequently, the green channel is adjusted to counter the offsets of the gray version caused by data embedding. The adjusted green channel is recovered using error-correcting bits. In [21], a cover image is segmented into rough and smooth areas based on local complexity then three bits are embedded into each pixel belonging to smooth areas and one bit is embedded into each pixel belonging to the rough area.

In [22], a steganographic solution for spatial color images is proposed by considering color pixel vectors of the same spatial location. The embedding payload is adaptively assigned to all color channels. An adapted clustering modification directions approach is used to resist steganalytic methods. The work in [23], proposed a data embedding solution based on the multi-matrix structure of turtle shell where weighting is used to divide the cover image into smooth, texture and edge regions. The smaller turtle shell matrix is used to ensure lower embedding in the pixels of smooth regions, and the larger turtle shell matrix is used for the edge region. The work in [24] proposed a hybrid steganography method based on least significant bit replacement and Hamming code. Consequently, sharp areas of the image are used for higher data embedding rates and vice-a-versa. In [25], a pixel value difference based high capacity methodology is proposed in which data embedding varies depending on the local contrast of the pixel pairs, which are chosen in a non-sequential manner. In the work published in [26], a histogram of oriented gradient solution is proposed to find the dominant edge direction for each 2×2 block of pixels. Blocks used for data embedding are then determined based on the gradient magnitude and angle of the cover image. Consequently, data is embedded in the dominant edge direction using pixel value differences. And least significant bit substitutions are used for data embedding in the remaining pixels of the block.

Finally in [27], secret data is encrypted into a sequence of two-digit decimal numbers. Each digit is embedded in a pixel by changing the pixel's digits to the nearest pixel value. It is shown that such a digit-based embedding solution outperforms the well-known least significant bit and pixel value difference approaches.

In this paper, we propose the use of recursive quad-tree block partitioning with block sizes ranging from 2×2 to 16×16 pixels. The block size is chosen according to the underlying image content. The range of control pixels values is used to estimate the spread of pixel intensities in a given block. This measure is used to make a decision on applying recursive quad-tree block partitioning. Message embedding and extraction are then implemented using matrix encoding [28]. It is shown that, in comparison to the fixed-size block approach, this solution reduces image distortion and results in less detectability when blind steganalysis is applied.

The rest of the paper is organized as follows; Sect. 2 introduces the main contribution of this paper which is the recursive quad-tree partitioning for variable-size block-based data embedding. Section 3 reviews matrix encoding and its application to data embedding and extraction specific to this work. Section 4 discusses security considerations using

blind steganalysis. Section 5 introduces the experimental results and Sect. 6 concludes the paper.

Recursive quad-tree block partitioning solution

In this section, we propose the use of recursively partitioned blocks for data embedding. Similar partitioning is used in the HEVC digital video compression algorithm [29]. The objective in inter-image video compression is to partition an image block to enhance the efficiency of motion compensation to meet a given rate-distortion criteria. In this work, the objective is to vary the block size and consequently vary the embedding rate according to the content of the blocks. The embedding rate is defined as the number of message bits divided by the total number of pixels in an image. It makes sense to use large block sizes and lower embedding rates in homogenous image areas and vice versa. For that to work, there are two conditions to be met. The first condition is that the variable block sizes should cover the whole image without gaps. The second condition is related to defining a criterion for determining the size of each block that works for both data embedding and extraction. As such, the extractor process can reproduce the same variable block sizes.

To meet the first condition, we propose the use of recursive quad-tree block partitioning where a cover image is partitioned into NxN block of pixels, an example is 16×16 pixels. A splitting criterion is used to recursively partition the NxN block into 4 sub blocks until the minimum block size of 2×2 is researched. This recursive partitioning is referred to as quad-tree block partitioning. We illustrated this petitioning with an example in Fig. 1.

The resultant block sizes can be 16×16 if no partitioning is applied, or a mix of 8×8, 4×4 and 2×2. By following such a block partitioning, the first condition is met where no gaps exist between the image blocks.

For the second condition to be met, we propose the use of a simple partitioning criterion that can be replicated for message extraction. The top-left pixels of each 2×2 block in the NxN block are used to estimate the dispersion or spread of the pixels in the underlying block. This is achieved using the range operator, which is simply the difference between the maximum and minimum control pixel values. If the range is less than a TH then we assume that the pixel block is homogenous and, therefore, do not apply partitioning. Otherwise, the block is divided into 4 partitions and the procedure is repeated for each N/2xN/2 partition. That is, the control pixels of the N/2xN/2 block are used to compute the range operator. If the range is greater than or equal to the TH then further partitioning takes place. The procedure stops when a minimum size of 2×2 is reached or when no further partitioning takes place, which is determined by the partitioning

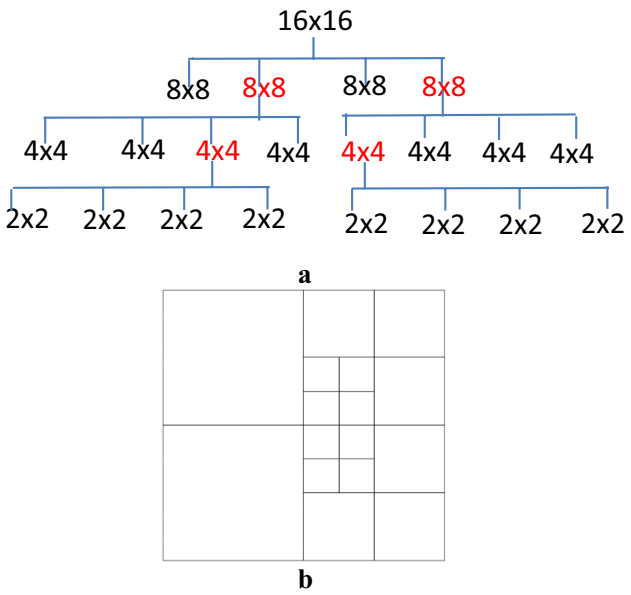


Fig. 1 **a** Example quad-tree block partitioning, **b** corresponding blocks with variable sizes.

criterion. Clearly, decreasing the range TH results in a higher percentage of block partitions and, therefore, higher embedding rates. In the experimental results, the TH.

varies from 5 to 150, with a step of 5, to cover embedding rates from 2.5 to 50%.

In the example given in Fig. 1, the range of control pixel values of a 16×16 block was greater than the range TH, which indicates that this block of pixels is not homogenous. Therefore, the block is split into four 8×8 blocks. The first and third 8×8 blocks contain homogenous pixel values and, therefore, are not further split. The second and fourth 8×8 blocks in this example have a range of pixel values greater than the range TH and, therefore, both blocks are further split into 4×4 blocks. Finally, the third 4×4 block of the second 8×8 block and the first 4×4 block of the fourth 8×8 blocks are further split into 2×2 blocks. The splitting stops when the range of a given block is smaller than the range TH or when a block size of 2×2 is reached. Part (b) of the figure provides a visual description of the locations of the split blocks within the 16×16 block for this particular example.

The aforementioned proposed solution of recursive quad-tree block partitioning is further illustrated in Fig. 2. The overall data embedding algorithm is listed in Fig. 3 and the split algorithm is listed in Fig. 4.

The extraction process starts by splitting the image into non-overlapping blocks of size 16×16. Each block is then recursively partitioned using the proposed quadtree partitioning algorithm. At this point, the range TH used in partitioning should be made available to the extractor. One solution is to dedicate the first 16×16 block to embed the range

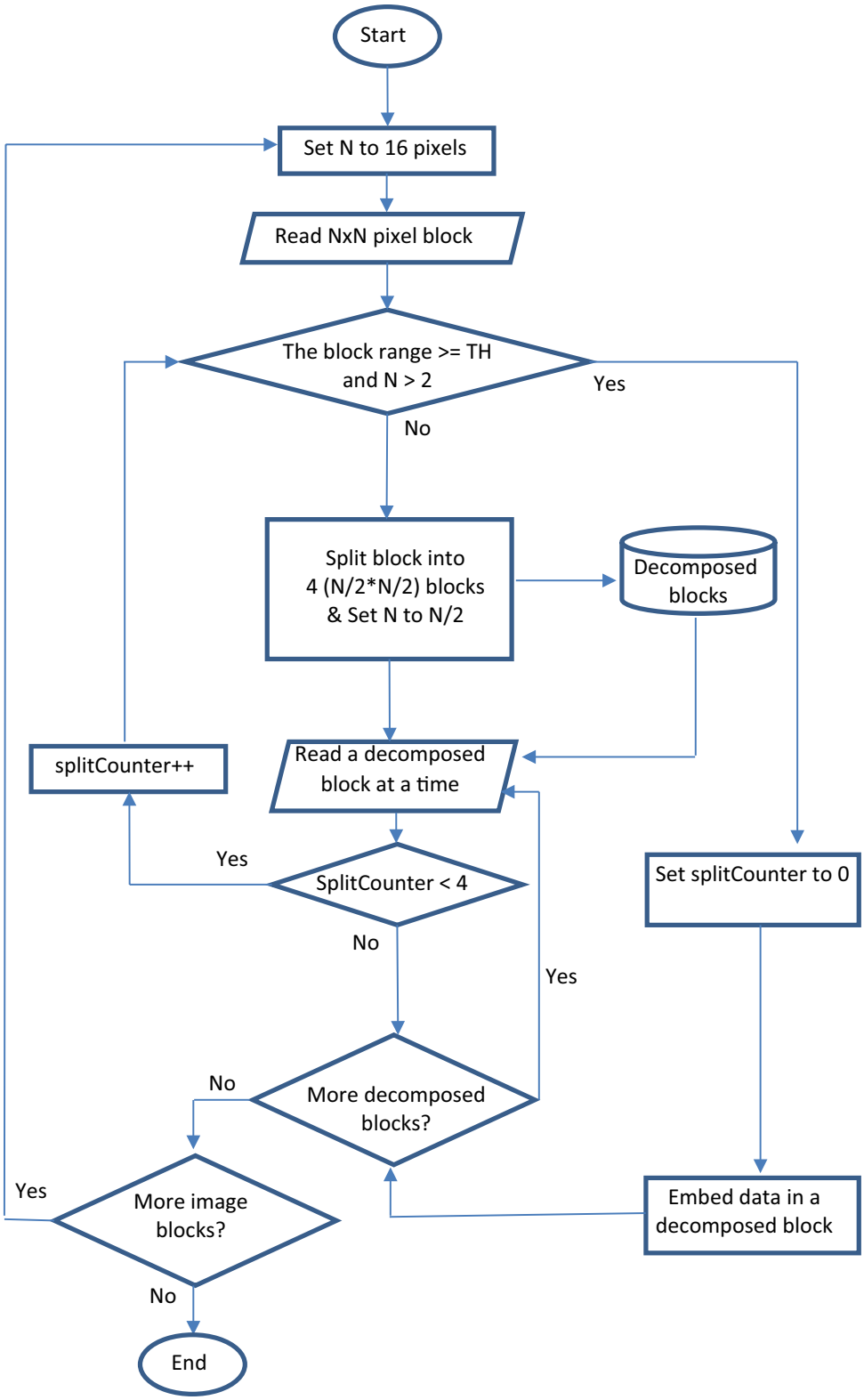


Fig. 2 Proposed solution of recursive quad-tree block partitioning

```
---------------------------------------------------------------------
ALGORITHM data embedding in recursively split blocks is:
---------------------------------------------------------------------
INPUT:
Sequence of NxN block of pixels
RangeTH for recursive block splitting
OUTPUT:
Split sub blocks with embedded data

for each NxN block in block sequence
  blk = org_block(1:2:end, 1:2:end)//control pixels
  splitLevel = 0 // 0=>NxN, 1=>N/2xN/2, 2=>N/4xN/4
  result_splits=findSplit(rangeTH, blk, splitLevel,
                          result_splits)
  subBlocks=getSubBlocks(from: org_block, according_to:
                         result_splits)
  for each subBlock in subBlocks
        embedData(message bits, subBlock)
end
```

Fig. 3 Pseudocode for the message embedding algorithm

```
---------------------------------------------------------------------
ALGORITHM recursive split of block is:

---------------------------------------------------------------------
INPUT:
rangTH: TH used for recursive block splitting
blk: Block of control pixels
splitLevel:
Level of split, 1=>NxN, 2=>N/2xN/2, 3=>N/4xN/4
OUTPUT:
result_splits: a list of split flags

result_splits = findSplit(rangeTH, blk, splitLevel,
                          result_splits)
  len = (length of blk)
  if len >= 2
    %--No split case (stop recursion)
    if max(blk(:))-min(blk(:)) < rangeTH
      if   splitLevel==0,
        result_splits = [no split in NxN block]
      elseif splitLevel==1,
        result_splits = [result_splits]
                append [no splits in N/2xN/2 block];
      elseif splitLevel==2,
        result_splits = [result_splits]
                append [no splits in N/4xN/4 block];
      end
    %--split case (recursive calls)
    else
      result_splits = [result_splits SplitFlag];
      for each subBlk in blk
        result_splits = findSplit(rangeTH, subBlk, len/2,
                                  splitLevel+1, result_splits);
    end
  end
end
```

Fig. 4 Pseudocode for the recursive split algorithm

TH. It can be agreed between the embedding and extraction processes that the first block is always partitioned into exactly four 8×8 blocks to facilitate embedding and extracting the range TH. Embedding can be performed using matrix encoding as well.

After that, the extraction process can resume splitting the remaining blocks using quadtree partitioning. Message embedding and extraction are implemented using matrix encoding as explained in the next section. It is noteworthy to mention that the implementation of the proposed solution raised a case that caused a minor difference between the block partitioning at the embedder and the extractor. Assume that by chance, a control pixel is to be modified for message embedding. This might happen as the size of the underlying block might be greater than 2×2 and the control pixels are located at the top left corner of each 2×2 block as mentioned previously. The pixel shall be modified such that it remains in the [min, max] range of control pixel values.

However, if the control pixel to be modified has the minimum value in the block, then it can only be modified by incrementing it by 1. Such increment reduces the actual pixel range of control pixels. If the new range is less than the TH used for message embedding, then the same block which was partitioned at the embedder will not be partitioned at the extractor.

Although this is a subtle and a rare case, nonetheless, it has to be investigated further. In Table 1, we use an experimental setup that is explained in the experimental section where 3,334 images are used from the BOWS2 [30] dataset. The first column of the table shows the embedding rate i.e., the number of message bits divided by the total number of pixels in an image, the second, third, fourth and fifth show the average percentage of block sizes as a result of partitioning. More importantly, the last columns show the accuracy of the block partitioning. This refers to the average percentage of blocks that are partitioned exactly the same at the embedding and extraction processes.

It is shown in the table that in all embedding rates, the average percentage of blocks partitioned exactly the same at both the embedder and the extractor is between 99.2% and 99.94%. At a low embedding rate, the range TH is high which results in a high percentage of 16×16 blocks. In other words, the lower the embedding rate, the lower is the rate of block partitioning. As shown in the table, for the embedding rate of 5%, the percentage of 16×16 blocks without splitting is 92.2%, and, therefore, the accuracy of partitioning the blocks at the extractor is very high (99.94%). On the other extreme, at the highest embedding rate of 38%, the percentage of 16×16 is 5.9% and the percentage of fully partitioned

Table 1 Average partitioning accuracy at the extractor

Embedding rate%	2×2 blk%	4×4 blk%	8×8 blk%	16×16 blk%	Partitioning accuracy (%)
5	0.7	4.2	10.1	92.2	99.94
10	5.9	14.7	17.9	67.5	99.85
15	11	19.4	18.6	55.4	99.79
20	25.7	23.7	15.6	38.4	99.68
25	36.3	22.7	13.6	30.1	99.6
30	46.6	20.7	12.3	22.5	99.49
38	64.6	18.4	11.1	5.9	99.2

Table 2 Percentage of erroneous message bits if image partitioning in embedding and extraction processes differ

Embedding rate%	Partitioning accuracy (%)	Erroneous message bits (%)
38	99.2	0.010
30	99.49	0.018
25	99.6	0.022
20	99.68	0.027
15	99.79	0.033
10	99.85	0.041
5	99.94	0.042

blocks, i.e. 2×2, is around 65%. Therefore, the percentage of blocks partitioned exactly the same at the extractor is 99.2%.

The partitioning accuracy reported in Table 1 can be used to compute the error percentage in the extracted image using Eq. 1.

$$\text{msgErrRate\%} = 100 * 1/\text{len(msg)} * \sum_{\substack{iin \\ \{2x2,4x4,8x8,16x16\}}} (1 - \text{PartAcc}) * \text{ratio}\left(blk_i\right)\text{count}\left(\text{ErrBits}_i\right) \tag{1}$$

where len(msg) is the length of the embeded message, msg, in bits, PartAcc is the partitioning accuracy as reported in the last column in Table 1, ratio(blk_i) is the ratio of 2×2, 4×4, 8×8 and 16×16 blocks, *blks*, as given in Table 1 as well. The count(ErrBits$_i$) is the number of message bits in error per block given that the partitioning was not replicated correctly at the extractor. For 2×2 blocks, the maximum number of message bits embedded are 2, hence if the block is not replicated correctly at the extractor 2 bits are lost. For 4×4 blocks the number of error bits in the worst-case scenario is 8. This scenario occurs if the block was split during embedding but not during extraction. For 8×8 blocks, the maximum number of message error bits is 32 and for 16×16 blocks it is 96 bits. Again, all counts are based on the worst-case scenario.

Table 3 Message segments in bits, the required number of cover pixels and corresponding pixel block sizes

Message segment, *n* bits	Number of required cover pixels	Required block pixel size
2	3	2×2
4	15	4×4
6	63	8×8
8	255	16×16

According to the equation above, the actual percentage of erroneous message bits is given in Table 2. As seen in the table, the percentage of erroneous message bits are insignificant, nonetheless there is room for further research to perfect it. The error percentage increases as the percentage of 16×16 blocks increases. This is expected as 16×16 blocks if not recovered correctly can result in the worst-case scenario as mentioned above.

Matrix Encoding

Matrix encoding is an efficient message embedding solution [28]. The efficiency of the solution stems from the fact that the number of changes required to embed a given message are reduced. Matrix encoding was successfully used by the author for message embedding in compressed MPEG videos as reported in [31].

In this work, message bits are divided into n bits message segments, where the parameter n is known as the matrix encoding parameter. Table 3 lists typical message segment lengths in bits and the required number of cover pixels such that a maximum of 1-pixel value is modified according to matrix encoding. Each message segment is embedded using 2^n-1 pixels. The table also lists the

Table 4 Example dependencies between message bits and cover pixel

	Msg bit$_1$	Msg bit$_2$	Msg bit$_3$
P_1	√		
P_2		√	
P_3	√	√	
P_4			√
P_5	√		√
P_6		√	√
P_7	√	√	√

Table 5 Matrix encoding example with 3 message bits ($n=3$)

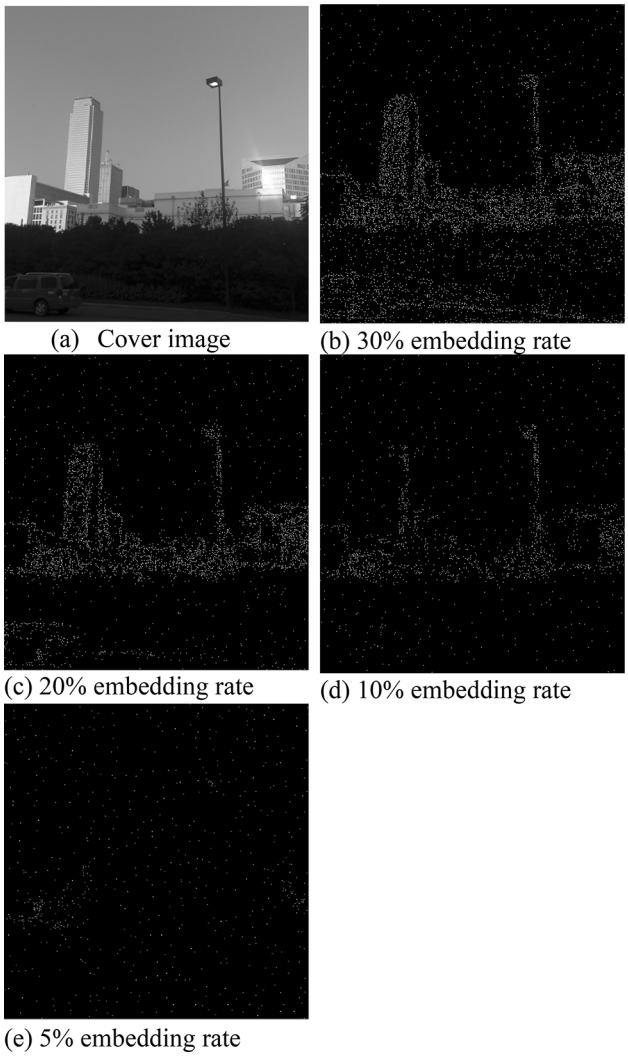
Parity tests			Action
$m_1=t_1$	$m_2=t_2$	$m_3=t_3$	Cover pixels are not modified
$m_1\neq t_1$	$m_2\neq t_2$	$m_3\neq t_3$	Modify p_7
$m_1\neq t_1$	$m_2\neq t_2$	$m_3=t_3$	Modify p_3
$m_1\neq t_1$	$m_2=t_2$	$m_3\neq t_3$	Modify p_5
$m_1=t_1$	$m_2\neq t_2$	$m_3\neq t_3$	Modify p_6
$m_1\neq t_1$	$m_2=t_2$	$m_3=t_3$	Modify p_1
$m_1=t_1$	$m_2\neq t_2$	$m_3=t_3$	Modify p_2
$m_1=t_1$	$m_2=t_2$	$m_3\neq t_3$	Modify p_4

required pixel block size according to the proposed quad-tree block partitioning solution. For example, to embed 4 bits, 15 (or 2^4–1) pixels from the cover image are needed and the corresponding smallest block size that contains 15 pixels is 4×4 pixels.

For a particular example of $n=3$, the dependencies between the message bits m_i; $i=1\ldots n$ and the cover pixel p_j; $j=1.0.2^n$-1are shown in Table 4.

To embed a message of 3 bits, a message bit is compared against the parity of the corresponding cover pixels of the columns of Table 4. If the parity is different than the message bit value, then one of the cover pixels is modified. For example, if $\text{MsgBit}_1\neq p_1\oplus p_3\oplus p_5\oplus p_7$ (i.e. if $\text{MsgBit}_1\neq\text{mod}(p_1+p_3+p_5+p_7, 2)$) then p_1 is either increased or decreased by a value of 1. Note that for embedding any message segment, a maximum of one pixel will be modified. With the case of $n=3$ and according to the dependencies in Table 4. let t_1, t_2 and t_3 denote the parities of the cover pixel values. According to Table 4, $t_1=p_1\oplus p_3\oplus p_5\oplus p_7$, $t_2=p_2\oplus p_3\oplus p_6\oplus p_7$ and $t_3=p_4\oplus p_5\oplus p_6\oplus p_7$. Consequently, embedding message 3-bits requires one of the actions listed in Table 5.

Figure 5 shows an example image where data embedding is performed using the proposed recursive quad-tree block partitioning with various embedding rates. The embedding rates vary according to the range TH used.



(a) Cover image (b) 30% embedding rate (c) 20% embedding rate (d) 10% embedding rate (e) 5% embedding rate

Fig. 5 Locations of data embedding using the proposed quad-tree partitioning solution

In this work, the range TH is varied to generate a message embedding rate up to 50%, where the embedding rate is measured by dividing the number of message bits by the number of pixels in a cover image. In the figure, pixel blocks belonging to homogenous areas of the image are not split and remain with the maximum block size of 16×16. On the other hand, less homogenous area of the image are further split all the way down to 2×2 pixels. Using a low range TH results in more block splits and, therefore, higher embedding rates. Consequently, the location of data embedding reveals the structure of the cover image as shown in parts *a* and *b* of the figure. On the other hand, using a high range TH results in less block splits and larger block sizes with lower embedding rates. Consequently, the location of data embedding does not reveal the structure of the cover image as shown in parts *d* and *e*.

Steganalysis Consideration

It is required to prevent detecting the existence of embedded messages in raw images using blind staganalysis. In this work, we use two well-known spatial domain steganalysis methods to assess the detectability of message embedding using the proposed solution. Namely; Li-11D [23] and SPAM [24], the former uses normalized histograms of linear transformations as feature vectors and the latter uses nonlinear markov-based features. In the experimental results section we extract features from clear and stego images to train two classifiers using SVM. We show that blind steganalysis does not distinguish between stego images and clear images based on the proposed recursive quad-tree partitioning message embedding solution.

Experimental Results

In all the experiments to follow, we mainly compare our work against the recent work proposed in [32] and [9] and, therefore, we use similar experimental setup. The messages to embed in the images are generated using a uniform random number generator. The image dataset used is BOWS2 [30], which contains 10,000 natural images with a spatial resolution of 512×512. Systematic sampling is used in which every third image is used in our experiments, resulting in a total of 3334 images.

The proposed solution is assessed in terms of message embedding capacity, the effect on image quality and rate of pixels' value change. The message embedding capacity is measured by dividing the number of message bits by the number of pixels in a cover image. The pixel change rate on the other hand, is measured by dividing the number of modified pixels by the total number of image pixels. For assessing image distortion, PSNR, SSIM [33] and image histogram differences are used. Image histogram differences are measured by computing the Sum of Absolute Differences (SAD) between the histograms of the original and modified images. The SAD is then normalized by dividing it by 256 which is the number of histogram bins. We report the results of applying blind staganalysis to assess where or not data embedding is detectible.

The results to follow apply the embedding solution to variable block sizes using the proposed quad-tree block decomposition.

In all the figures to follow, the embedding rate for the recursivel partitioned blocks solution is changed by varying the TH of the range operator as introduced in Sect. 2.

In Fig. 6, the pixel change rate against the embedding capacity is plotted.

In [34], 2 message bits are embedding in a 2×2 block of pixels. Therefore, the embedding rate in 2×2 pixels is the highest as shown in Fig. 6. This comes at the expense of a higher rate of pixel changes.

It is shown in the figure, that using the recursively partitioned blocks solution, results in the lowest percentage of pixel changes up to 10% embedding rate. After that, increasing the embedding rate implies increasing the percentage of smaller block sizes including 8×8, 4×4 and 2×2. Therefore, the number of changed pixels starts to increase. However, it remains below that of the fixed block size of 2×2 as a mix of block sizes is still used.

In Fig. 7, the average PSNRs of the proposed and reviewed solutions are plotted.

The figure shows that the PSNR of the proposed solution using the recursively-partitioned blocks results in low image distortions at low embedding rates. This is so, as larger block sizes are used at low embedding rates. Having increased the embedding rates, however, results in lower PSNR as the

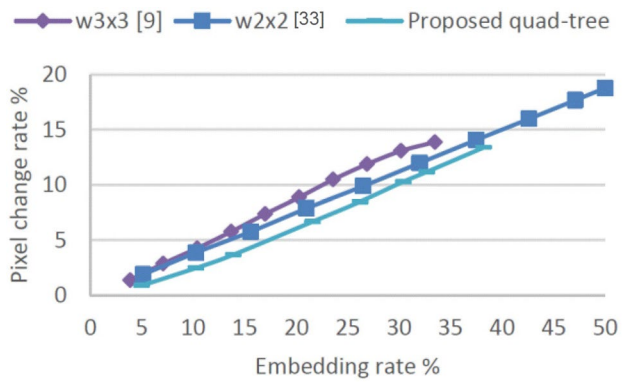


Fig. 6 Message embedding rate versus rate of pixel change

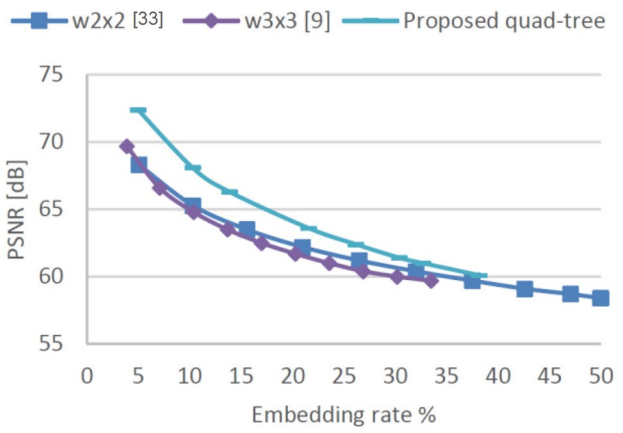


Fig. 7 Normalized sum of absolute histogram differences between the cover and stego images

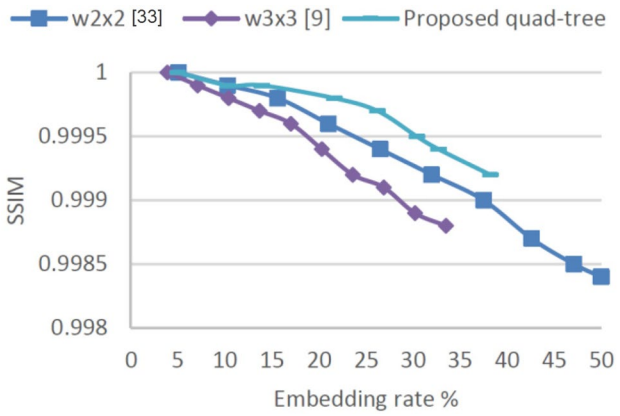


Fig. 8 Average PSNR at difference message embedding rate

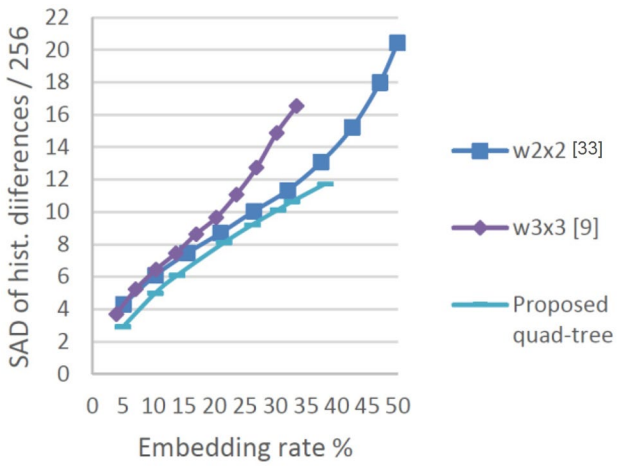


Fig. 9 Average SSIM as a function of the embedding rate

percentage of smaller block sizes increase. In all cases, the PSNR remains higher than the 2×2 block size solution and the reviewed solution.

The structural similarity index, SSIM, complements PSNR as it considers changes in structural information whilst taking image contrast into consideration. In Fig. 8, the average SSIM results are reported.

As shown in the figure, at low embedding rates (below 10%), nearly all solutions result in similar SSIM. As the embedding rate increases, the reviewed solutions start to embed more message bits into high-frequency regions which affects the structure of the image. With recursively-partitioned blocks, the structure of the modified image is not affected much as the use of recursively-partitioned blocks results in embedding message bits in and around high-frequency areas. This is a very clear advantage of the proposed solution in comparison to existing work.

Table 6 Average PSNR result in dB with a comparison with existing work

Embedding rate%	Ref. [13]	Ref. [15]	Proposed solution
5	63.82	63.07	72.4
10	61.94	60.23	68.1
20	59.04	57.85	63.6
25	58.99	56.87	62.4
30	57.23	55.73	61.4
40	55.95	54.38	60.1
Avg	**59.5**	**58.0**	**64.7**

Table 7 Classification results using Li-110D steganalysis

Embedding rate%	Ref. [9] (%)	Ref. [32] (%)	Proposed solution (%)
5	49.58	49.66	50.61
10	51.27	50.19	50.04
20	54.37	49.94	49.97
25	55.69	50.9	49.73
30	57.69	52.04	49.9
40	60.23	54.8	50.3
Avg	**54.78**	**51.25**	**50.1**

In Fig. 7, the normalized sum of absolute histogram differences between the cover and modified images is presented.

Reducing the differences between the histogram of the original images and the modified ones is preferred. The results in Fig. 7 are consistent with the pixel change rate of Fig. 6 and the average PSNR values of Fig. 9. The work in [9] also reported the average PSNR results of [13] and [15] on the same BOWS2 image dataset. In Table 6, we list these results and provide a comparison with our proposed solution. Again, the average PSNR results indicate the advantage of the proposed solutions. It is shown in the table that the proposed solution results in higher PSNR values at low and high embedding rates.

Table 7 presents the results of blind steganalysis with comparison to existing work using the same BOWS2 dataset. We also report the results for the proposed recursively-partitioned blocks solution. The aim is to train a classifier to distinguish between clean and stego images based on features extracted using various techniques.

As mentioned previously, we extract feature using the Li-110D technique which is designed for spatial domain steganalysis [31]. Half the data is used for training and the other half for testing. The model generation is done through SVM with a linear kernel. Tenfold round robin is used for

Table 8 Classification results using SPAM steganalysis

Embedding rate%	Ref. [32] (%)	Proposed solution (%)
5	48.73	49.51
10	49.49	49.54
20	50.22	49.45
25	51.6	49.67
30	53.12	49.81
40	57.12	50.79
Avg	**51.7**	**49.8**

training and testing and the average classification results are reported in Table 7. The classification results in the table indicate that the proposed solution is less detectable than the reviewed work, it also indicate that the use of Li-110D blind steganalysis cannot be used to detect stego images using the proposed solutions. Clearly, using recursively-partitioned blocks for data embedding makes the detection challenging as illustrated in the average classification results in the table.

We also experiment with a more sophisticated blind steganography technique referred to as SPAM [2]. Note that the use of SPAM was not reported in [9]. Nonetheless, we report classification results using SPAM with the same experimental setup as reported above. Table 8 presents the classifications results for both of the proposed solution and the reviewed work of [32] using the same BOWS2 dataset.

From the results in Tables 7 and 8, it seems that SPAM is more suitable for detecting steganography with fixed block sizes. However, it cannot detect steganography with recursively partitioned blocks as the average classification accuracy is below 50%.

Conclusion

The paper proposed a novel data embedding solution in row images by recursively-partitioning pixel blocks based on techniques adopted from the HEVC video coding. Data embedding and extraction were implemented using matrix encoding. It was proposed to use varying block sizes from 2×2 to 16×16 pixels. Two conditions were identified for varying block sizes within a cover image, namely, avoiding gaps between blocks and being able to reproduce the same partitioning at the extraction process. The first condition is met using a recursive quadtree partitioning approach while for the second condition, a detailed analysis was presented which examined the accuracy of the extracted messages and true percentage of identically partitioned blocks at both the embedder and extractor. The experimental results revealed that such a solution is less detectible by blind steganography and results in less image distortions. It was also shown that varying the block sizes using the proposed solution results in the lowest rate of pixel change, best PSNR, SSIM and histogram matching results. In the reviewed solutions where edges are targeted for data embedding, the SSIM is affected negatively as the structure of the cover image is distorted. Lastly, it was shown that using two different blind steganalysis approaches, the classification accuracy was around 50%, which makes it very difficult to detect data embedding using the proposed solutions.

Compliance with Ethical Standards

Conflict of Interest Author T. Shanableh declares that he has no conflict of interest.

References

1. Aly H. Data hiding in motion vectors of compressed video based on their associated prediction error. IEEE Trans Inf Forensics Secur. 2011;6(1):14–8. https://doi.org/10.1109/TIFS.2010.2090520.
2. Shanableh T. Matrix encoding for data hiding using multilayer video coding and transcoding solutions. Signal Process Image Commun Elsevier. 2012;27(9):1025–34.
3. Shanableh T. Data hiding in MPEG video files using multivariate regression and flexible macroblock ordering. IEEE Trans Inf Forensics Secur. 2012;7(2):455–64. https://doi.org/10.1109/TIFS.2011.2177087.
4. Tew Y, Wong K. Information hiding in HEVC standard using adaptive coding block size decision. IEEE Int Conf Image Process. ICIP, Paris, 2014
5. Shanableh T. 'Altering split decisions of coding units for message embedding in HEVC', multimedia and applications. Springer. 2017. https://doi.org/10.1007/s11042-017-4787-6,May.
6. Guo L, Ni J, Su W, Tang C, Shi YQ. Using statistical image model for JPEG steganography: uniform embedding revisited. IEEE Trans Inform Foren Secur. 2015;10(12):2669–800.
7. Guo JM, Le TN. Secret communication using JPEG double compression. IEEE Signal Process Lett. 2010;17(10):879–82.
8. Zhang L, Wang H, Wu R. A high-capacity steganography scheme for JPEG2000 baseline system. IEEE Trans Image Process. 2009;18(8):1797–803.
9. Al-Dmour H, Al-Ani A. A steganography embedding method based on edge identification and XOR coding. Expert Syst Appl. 2016. https://doi.org/10.1016/j.eswa.2015.10.024,vol.46.
10. Al-Dmour H, Al-Ani A. Quality optimized medical image information hiding algorithm that employs edge detection and data coding. Comput Methods Prog Biomed. 2016;127:24–43.
11. Feng B, Lu W, Sun W. Secure binary image steganography based on minimizing the distortion on the texture. IEEE Trans Inform Forens Secur. 2014;10(2):243–55.
12. Hong W, Chen TS. A novel data embedding method using adaptive pixel pair matching. IEEE Trans Inform Forens Secur. 2012;7(1):176–84.
13. Hou CL, Lu C, Tsai SC, Tzeng WG. An optimal data hiding scheme with tree-based parity check. IEEE Trans Image Process. 2011;20(3):880–6.

14. Yang CH, Weng CY, Wang SJ, Sun HM. Adaptive data hiding in edge areas of images with spatial LSB domain systems. IEEE Trans Inform Forens Secur. 2008;3(3):488–97.
15. Luo W, Huang F, Huang J. Edge adaptive image steganography based on LSB matching revisited. IEEE Trans Inform Forens Secur. 2010;5(2):201–14.
16. Khodaei M, Faez K. New adaptive steganographic method using least-significant-bit substitution and pixel-value differencing. IET Image Process. 2012;6(6):677–86.
17. Tang W, Li H, Luo W, Huang J. Adaptive steganalysis based on embedding probabilities of pixels. IEEE Trans Inform Forens Secur. 2016;11(4):734–45.
18. Luo W, Huang F, Huang J. Edge adaptive image steganography based on LSB matching revisited. IEEE Trans Inform Forens Secur. 2012;5(2):201–14.
19. He M, Liu Y, Chang CC, He M. A mini-sudoku matrix-based data embedding scheme with high payload. IEEE Access. 2019;7:141414–25.
20. Dongdong H, Weiming Z, Kejiang C, Sian-Jheng L, Nenghai Y. Reversible data hiding in color image with grayscale invariance. IEEE Trans Circuits Syst Video Technol. 2019;29(2):363–74.
21. Fang C, Bowen A, Heng Y, Zhenjun T. Local complexity based adaptive embedding mechanism for reversible data hiding in digital images. Multim Tools Appl. 2019;78(7):7911–26.
22. Qin X, Li B, Tan S, Zeng J. A novel steganography for spatial color images based on pixel vector cost. IEEE Access. 2019;7:8834–46.
23. Li L, Lifang W, Chin-chen C. Data embedding scheme based on multi-matrix structure of turtle shell to avoid human eye perception. Multim Tools Appl. 2019;78(8):10473–90.
24. Wang Y, Tang M, Wang Z. High-capacity adaptive steganography based on LSB and hamming code. Optik. 2020;26:164685.
25. Mukherjee N, Paul G, Saha SK, Burman D. A PVD based high capacity steganography algorithm with embedding in non-sequential position. Multim Tools Appl. 2020;29:1–31.
26. Hameed MA, Hassaballah M, Aly S, Awad AI. An adaptive image steganography method based on histogram of oriented gradient and PVD-LSB techniques. IEEE Access. 2019;17(7):185189–204.
27. Verma V, Muttoo SK, Singh VB. Enhanced payload and trade-off for image steganography via a novel pixel digits alteration. Multim Tools Appl. 2020;79:7471–90. https://doi.org/10.1007/s11042-019-08283-9.
28. Crandall R. Some notes on steganography. 1998. http://dde.binghamton.edu/download/Crandall_matrix.pdf. Accessed Sept 2020.
29. Sullivan GJ, Ohm JR, Han WJ, Wiegand T. Overview of the high efficiency video coding (HEVC) standard. IEEE Trans Circ Syst Video Technols. 2012;22(12):1649–68.
30. Bas, P., Furon, T.: 'Bows-2', 2007.
31. Li B, Huang J, Shi Y. Textural features based universal steganalysis, Electronic imaging, International Society for Optics and Photonics, 2008.
32. Pevny T, Bas P, Fridrich J. Steganalysis by subtractive pixel adjacency matrix. IEEE Trans Inform Foren Secur. 2010;5(2):215–24.
33. Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: From error visibility to structural similarity. IEEE Trans Image Process. 2004;13(4):600–12.
34. Shanableh T. Adaptive message embedding in raw images. In Fourth International Congress on Information and Communication Technology. Advances in Intelligent Systems and Computing, London, 2019.