Listless block-tree set partitioning algorithm for very low bit rate embedded image compression

Article *in* AEU - International Journal of Electronics and Communications · December 2012

DOI: 10.1016/j.aeue.2012.05.001

CITATIONS
23

READS
183

3 authors:

Ranjan Senapati
VNR Vignana Jyothi Institute of Engineering & Technology
52 PUBLICATIONS **149** CITATIONS
SEE PROFILE

Umesh C. Pati
National Institute of Technology Rourkela
90 PUBLICATIONS **405** CITATIONS
SEE PROFILE

Kamalakanta Mahapatra
National Institute of Technology Rourkela
214 PUBLICATIONS **1,269** CITATIONS
SEE PROFILE

Int. J. Electron. Commun. (AEÜ) 66 (2012) 985–995

Contents lists available at SciVerse ScienceDirect

International Journal of Electronics and Communications (AEÜ)

journal homepage: www.elsevier.de/aeue

Listless block-tree set partitioning algorithm for very low bit rate embedded image compression

Ranjan K. Senapati*, Umesh C. Pati, Kamala Kanta Mahapatra

Department of Electronics and Communication Engineering, National Institute of Technology, Rourkela, Orissa 769008, India

ARTICLE INFO

Article history:
Received 8 February 2012
Accepted 8 May 2012

Keywords:
No list SPIHT
Wavelet block tree coding
Listless block tree coder
Listless block tree DCT
Image compression
Embedded coder

ABSTRACT

This paper presents a listless implementation of wavelet based block tree coding (WBTC) algorithm of varying root block sizes. WBTC algorithm improves the image compression performance of set partitioning in hierarchical trees (SPIHT) at lower rates by efficiently encoding both inter and intra scale correlation using block trees. Though WBTC lowers the memory requirement by using block trees compared to SPIHT, it makes use of three ordered auxiliary lists. This feature makes WBTC undesirable for hardware implementation; as it needs a lot of memory management when the list nodes grow exponentially on each pass. The proposed listless implementation of WBTC algorithm uses special markers instead of lists. This reduces dynamic memory requirement by 88% with respect to WBTC and 89% with respect to SPIHT. The proposed algorithm is combined with discrete cosine transform (DCT) and discrete wavelet transform (DWT) to show its superiority over DCT and DWT based embedded coders, including JPEG 2000 at lower rates. The compression performance on most of the standard test images is nearly same as WBTC, and outperforms SPIHT by a wide margin particularly at lower bit rates.

1. Introduction

Recently low bit rate image compression has assumed a major role in applications such as processing and storage on handheld mobile or portable devices, wireless transmission, streaming data on the Internet and transmission in narrow band channels. Contemporary wavelet based coding provides substantial improvement in progressive picture build up qualities at lower rates. In addition, it supports a wide range of functionalities either with increase computational complexities, e.g., JPEG 2000 [1,2] or increase memory requirements, e.g., EZW [3], SPIHT [4], SLCCA [5] and [6].

EZW and SPIHT are spatial tree based, which generates fidelity progressive bit stream by encoding each bit planes of quantized dyadic subband coefficients. Both algorithms perform significant tests using set partitioning procedure. However, SPIHT has additional partitioning steps to distinguish between descendants (type A and type B). This leads to improved performance of SPIHT over EZW. While EZW moves from coarse to fine subbands, it explicitly does a breadth first search on the hierarchical trees. However, though not explicit, SPIHT does a roughly breadth first search as well. This is because, SPIHT appends four new descendant sets at the end of LIS (list of insignificant sets) by partitioning a grand descendant set. It is the appending to the LIS that results in approximate breadth first traversal of the SPIHT.

SPIHT uses list structure to keep track of set partitions. However, the use of lists in these coders causes a variable data dependent memory management as the list nodes are updated on every passes. At high rates, it is possible that the number of list nodes can be more than number of coefficients. This is an undesirable feature for hardware implementations. A variant of SPIHT called No list SPIHT (NLS) [7] which uses a state table with four bits per coefficients to keep track of set partitions is reported.

Though NLS is a low complexity image coding algorithm with performances nearly close to SPIHT, these coders do not fully exploit the coding performances at lower bit rates. By looking at the statistics of transformed images, the number of significant coefficients whose magnitudes are higher than earlier thresholds are very few on earlier bit plane passes. Since NLS does a roughly breadth first search, it codes zeros to each insignificant zero trees as it moves from coarsest to finest subbands. It is also experimentally demonstrated that there could be up to six to seven bit plane passes where NLS codes many zeros, as many trees are likely to be insignificant with respect to early thresholds. Bits indicating insignificance of a coefficient or subband are required, but they do

* Corresponding author. Tel.: +91 661 2464461; fax: +91 661 2462451.
E-mail address: rksphd@gmail.com (R.K. Senapati).
URLs: http://www.nitrkl.ac.in/faculty/ ucpati (U.C. Pati), http://www.nitrkl.ac.in/faculty/ kkm (K.K. Mahapatra).

1434-8411/$ – see front matter.
http://dx.doi.org/10.1016/j.aeue.2012.05.001

not code information that reduces distortion of the reconstructed image. This leads to a reduction of zero distortion for an increase of non zero rate. A block-tree algorithm called wavelet block truncation coding (WBTC) presented in [8,9] exploits both inter and intra subband correlation to improve the coding performance at very low bit rates with a slight increase of the decoder complexity. Pan et al. [10] presented a listless modified SPIHT algorithm which improves the low bit rate performance of SPIHT. However, it's performance drastically reduces from medium to higher bit rates. Our proposed algorithm named as listless block-tree coding (LBTC) which not only reduces the dynamic memory requirement almost by 88–89% but also enhances the low to higher bit rate performance with an average reduction of encoder and decoder complexity.

The proposed algorithm does explicit breadth and depth searches. State information is kept in a fixed size array which corresponds to the array of coefficient values with four bits per coefficient. Special markers are placed on lower levels of insignificant block trees when they are created. These markers are updated during tree partitioning. Efficient skipping of insignificant block tree is accomplished using a morton scan sequencing. In stead of addressing each coefficient using two indices, linear indexing scheme is used; because, this particular format has several computational and organizational advantages.

Although wavelets are capable of more flexible space-frequency resolution trade offs than DCT, DCT is still widely used in many practical applications such as JPEG [11,12], MPEG-4 and H.264 [13] because of its compression performance and computational advantages. Recently, DCT-based coders with innovative data organization strategies and representations of DCT coefficients have been reported with high compression efficiency [14–19]. Therefore, in order to evaluate the coding efficiency over the DCT based coders, we combine the proposed LBTC algorithm with DCT. The proposed new coder named as Listless block-tree DCT '(DCT_LBT)' has some desirable attributes like progressive image compression, precise rate control and lower complexity. This makes DCT_LBT to be an ideal candidate for modern multimedia applications.

The organization of the paper is as follows: Section 2 discusses the proposed LBTC algorithm. In Section 3, DCT_LBT algorithm is presented. Memory analysis is presented in Section 4. Simulation results and performance comparisons are discussed in Section 5. Finally, the paper is concluded in Section 6.

2. The proposed LBTC algorithm

Generally, the encoding process of a wavelet transformed image is performed bit-plane by bit-plane. In order to do this, an appropriate threshold is pre-computed. Considering the transformed image as an indexed set of transformed coefficients c_{ij} located by ith row and jth column, the initial threshold is:

$$T = \left\lfloor log_2(\max_{i,j}\{|c_{i,j}|\}) \right\rfloor \quad (1)$$

The new coder presented here uses eight class of markers for partitioning a block tree. These are defined below:

- BIP: The pixel is insignificant or untested for this bit plane.
- BNP: The pixel is newly significant and it will not be refined for this bit plane.
- BSP: The pixel is significant and it will be refined for this bit plane.
- BDS: The pixel is the first (lowest index) child in a single tree consisting of all descendants of its parents.
- BCP: Like BIP which is applied during partitioning of IS pass, but it will be tested for significance immediately during same IS pass.
- SD: The pixel is the first child (lowest index) in a composite tree consisting of all descendants of its parent block.

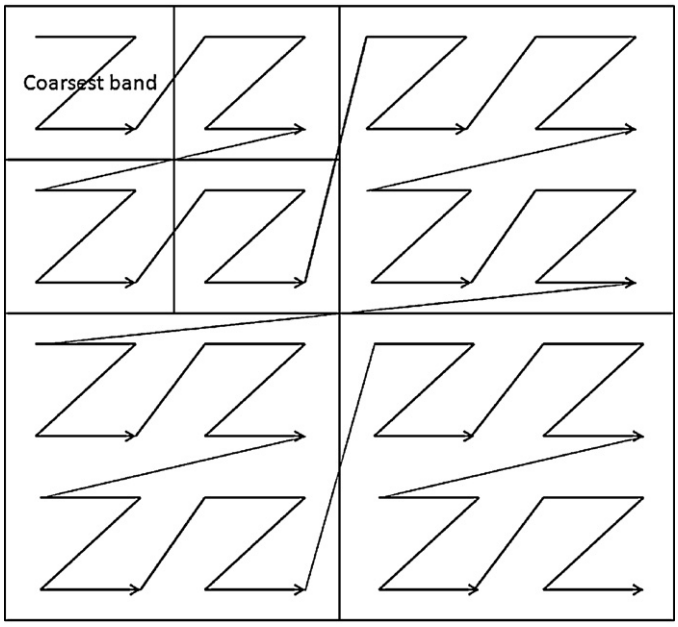

Fig. 1. Two dimensional morton scan sequence of the transformed coefficients.

- SG: The pixel is the first grandchild (lowest index) in a set consisting of all grand descendants of its grandparent block.
- SN*: The following markers are used on the leading nodes of each generations of an insignificant block tree.
  - SN2: The pixel is first child of SD set. This pixel and its 16 neighbors (4×4 blocks) can be skipped.
  - SN3: The pixel is first grandchild of SD or child of SG set. This pixel and its 64 neighbors (8×8 blocks) can be skipped.
  - SN4: The pixel is first great grandchild of SD or grand child of SG set. This pixel and its 256 neighbors (16×16 blocks) can be skipped.

⋮

  - SN6: The pixel is 5th generation descendants of SD or 4th generation descendants of SG set. This pixel and its 4096 neighbors (64×64 blocks) can be skipped.

The two dimensional arrangement of wavelet transformed coefficients are mapped to an one dimensional array of length I using morton scan sequencing shown in Fig. 1, where two levels of wavelet decomposition is shown for illustration. The algorithm scans from top most bit plane to the least bit plane while updating threshold (T) by half every time. In each pass the coefficient array called *val* is examined for significance. A coefficient s is significant, if $s \geq 2^T$. The simplest method to test the significance is by bit wise AND operation. The decoder does exactly the reverse process. At the arrival of first significant bit of a coefficient, it is reconstructed as $\pm 1.5 \times 2^T$. The decoder add or subtract 2^{T-1} to its current reconstructed value depending on whether it inputs significant bit in the current pass or not. There is one to one correspondence between *markers* and *val*. This means markers update its value each time *val* at ith position become significance.

2.1. Initialization

During encoding, two maximum magnitude descendant arrays are computed. These are (a) $d_{max}^{m,n}(i)$ calculates the maximum descendant of a block tree whose root block size is $m \times n$. Index 'i'

is the lowest index of the root block, and (b) $g^{m,n}_{max}(i)$ calculates the maximum grand descendant sets of the root block $m \times n$. Therefore,

$$d^{m,n}_{max}(i) = max[d^{m/2,n/2}_{max}(i),\ d^{m/2,n/2}_{max}(i+(m/2+n/2)), d^{m/2,n/2}_{max}(i+2(m/2+n/2)),\ d^{m/2,n/2}_{max}(i+3(m/2+n/2))] \quad (2)$$

and,

$$g^{m,n}_{max}(i) = max\ [d^{m,n}_{max}(4i),\ d^{m,n}_{max}(4i+(m+n)), d^{m,n}_{max}(4i+2(m+n)),\ d^{m,n}_{max}(4i+3(m+n))]. \quad (3)$$

The descendants of a single tree is represented by:

$$d^{1,1}_{max}(i) = max\ [val(4i),\ val(4i+1),\ val(4i+2),\ val(4i+3),\ g^{1,1}_{max}(i)]. \quad (4)$$

Zero is substituted for $g^{m,n}_{max}(i)$ when $i \geq I/16$. The function *insert* is used to push markers SN* wherever needed by the lower tree levels when descendants are created. We can define the *insert* function as follows:

For 2×2 blocks:

$$\text{mark}[4i] = \text{SN2},\ \ \text{mark}[16i] = \text{SN3},\ \ \text{mark}[64i] = \text{SN4}, \ldots \quad (5)$$

and, for 4×4 blocks:

$$\text{mark}[4i] = 4 \times SN2,\ \ \text{mark}[16i] = 4 \times SN3,\ \ \text{mark}[64i] = 4 \times SN4, \ldots \quad (6)$$

A five levels of hierarchical tree has $1364 \times (m \times n)$ descendants, if the root block is of size $m \times n$. When the block tree is processed, SD marker and 5 SN* markers associated with the tree shall be encountered. Therefore, a large number of predictable insignificant coefficients can be skipped with little effort.

The DC (coarsest) subband block is coded using a slight different procedure than that in NLS (assume the number of coefficients inside DC subband are I_{dc}). NLS codes each coefficients in the DC band; whereas, special markers are used for coding the DC subband. The number of markers required depend on the size of DC band. The pseudo-code for the initialization procedure is shown below.

```
Initialization of DC subband :
    while i ≤ I_dc − 1
        mark[i : I_dc/4 : end(i)] ← M*
    and while i = I_dc, ..., 4I_dc
        mark[I_dc : 16 : 4I_dc] ← SD
```

Markers, M* are updated and the corresponding coefficients inside the subband are coded each time they become significant. Note that markers M* are only used to partition the DC subband. The coefficients from I_{dc} to $4I_{dc}$ will be marked as SD at a step of 16 or (4×4) blocks. This is because these are the offspring of block-trees having their root blocks (whose sizes are 2×2) lies in DC subband. A further compression is achieved as more blocks are likely to be insignificant on earlier passes using this techniques.

A function named 'skip' is used to indicate how many coefficients to be skipped when a marker is encountered during scanning process.

For (2×2) initial root block, if the marker is SD, skip[SD]=16. Similarly, for BIP, BNP and BSP, skip[S*P]=1. For marker SG, skip[SG]=64; For SN*, skip[SN2]=16, skip[SN3]=64,. . ., skip[SN6]=4096.

For (4×4) initial root block, if the marker is SD, skip[SD]=64, skip[S*P]=1, skip[SG]=256, skip[SN2]=64, skip[SN3]=256,. . .skip[SN6]=16385.

It is to be noted that the pseudo-code presented in Section 2.2 is exclusively for LBTC having initial root block size of (2×2). Additional markers are to be inserted into the algorithm during partitioning a block tree having initial root block size (4×4) to (2×2). Then, the block tree of initial root block size (2×2) is to be processed until it becomes a single tree at a bit plane pass. Once it is partitioned into a single tree, the root of the descendant is to be marked as BCP. Then, the algorithm switches to the one proposed for LBTC(2×2).

Therefore, in general, the algorithm uses additional $(n-1)$ markers for each increase of root block size $(2^n \times 2^n)$ blocks, where $n = 1, 2, 3, \ldots$

2.2. The pseudo-code of encoder algorithm

```
Step 1: Insignificant pixel pass
i = 0, while i ≤ I
  if mark[i] ← BIP                          if insignificant
    output(d ← val[i]AND s)                 send the coeff significance
    if d,                                   if significance
      output(sign[i])
      mark[i] ← BNP                         mark as newly significant
      i = i + 1                             move past the coeff
    else,
      i ← i + skip(mark[i])                 skip to the next block/set
  end.
```

Step 2: Insignificant set pass

1. If mark[i] ← SD — set of descendants
   output(d ← $d_{max}^{m,n}\left\lfloor \frac{i}{4} \right\rfloor$ AND s) — send the block tree significance
   if d, — if significance
   mark[i] = mark[i + (m × n)] ← BDS — quad split
   mark[i + 2(m × n)] = mark[i + 3(m × n)] ← BDS
   mark[4i] ← SG — set of grand descendants block tree
   else,
   i = i + 4(m × n) — skip to the next block tree
2. Elseif mark[i] ← SG — if grand descendant block tree
   output(d ← $d_{max}^{m,n}\left\lfloor \frac{i}{16} \right\rfloor$ AND s) — send the significance
   if d,
   mark[i] = mark[i + 4(m × n)] ← SD — quad split and mark
   mark[i + 8(m × n)] = mark[i + 12(m × n)] — as descendant block trees
   insert(i), insert(i + 4(m × n)), — mark boarders of these new sets
   insert(i + 8(m × n)), insert(i + 12(m × n))
   else,
   i = i + 16(m × n) — skip to the next grand descendant block tree
3. Elseif mark[i] ← BDS — if it is a single tree
   output(d ← $d_{max}^{1,1}\left\lfloor \frac{i}{4} \right\rfloor$ AND s) — check for significance
   if d, — if significance
   mark[i] = mark[i + 1] ← BCP — quad split and mark each
   mark[i + 2] = mark[i + 3] ← BCP — coeff as insignificant
   mark[4*i*] ← SD
   else, i = i + 4 — skip to the next tree
4. Elseif mark[i] ← BCP — if insignificant during current pass
   output(d ← val[i]AND s) — send the significance
   if d, output(sign[i]) — if significant, send the sign bit
   mark[i] ← BNP — mark as newly significant
   else, mark[i] ← BIP — else insignificant
   i = i + 1 — go to the next coefficient
5. Else
   i = i + 1 — skip to the next coeff

Step 3: Refinement pass

i = 0, while i ≤ I

1. If mark[i] ← BSP — significant coeff
   output(val[i]AND s) — refine the coeff
   i = i + 1 — move past the coeff
2. Elseif, mark[i] ← BNP — newly significant coeff
   mark[i] ← BSP — significant for next pass
   i = i + 1 — move past the coeff
3. Elseif, mark[i] ≠ BIP — if SN*
   i = i + skip(mark[i]) — skip to the next block/set
4. Else, i = i + 1 — move past the coeff
   End.

3. The proposed DCT_LBT embedded encoder

The block diagram of the proposed DCT_LBT embedded coder is shown in Fig. 2. First, the input image is divided into non-overlapping $N \times N$ blocks. Then each block is transformed using DCT. The DCT coefficients of each block are arranged in a wavelet like hierarchical manner. Though there exists a number of possible arrangement of coefficients, few arrangements are described below.

Case-1: The DCT coefficients are arranged in M levels of wavelet like hierarchical arrangement (e.g., $M = 4$ if $N = 16$; $M = 3$ if $N = 32$ or 8). The decision logic roots the coefficients of coarsest subband to the input. The coarsest subband is further divided into $N \times N$ blocks. Another M_1 level of wavelet like hierarchical rearrangement (e.g., $M_1 = 1$ if $N = 16$; $M_1 = 2$ if $N = 32$ or 8) is made by reapplying DCT to the $N \times N$ blocks of coarsest subband. The resultant rearrangement of DCT coefficients make the coarsest level to be of size 16×16, with an overall five levels of hierarchical arrangement in a standard 512×512 size monochrome image. The coefficients are converted to integers and quantized by proposed LBTC coding algorithm. For image reconstruction, exactly the same reverse process is carried out at the decoder side.

Case-2: The input image is dived into 32×32 blocks. The DCT coefficients of each block are arranged into 5 levels of hierarchical arrangement. Coefficients present in the similar subband of all 32×32 blocks are grouped together. This makes an overall of 5 levels of arrangement in a standard 512×512 size image. This types of arrangement of coefficients has already been reported [14–19]. The function of decision logic is simply to root the overall coefficient arrangement to the proposed LBTC coding algorithm instead of rooting into the feedback path. The rate control logic decides the exact target rate at which the image is to be compressed.

3.1. Relation between transformed coefficients

Fig. 3 shows the arrangement of 8×8 DCT coefficients in a typical 3-level wavelet pyramid structure for illustration. After labeling 64 coefficients in a 8×8 block, the parent child relationship is explained below:

The parent of coefficient i is $\lfloor i/4 \rfloor$ for $1 \leq i \leq 64$, while the set of four children associated with coefficient j is $\{4j, 4j+1, 4j+2, 4j+3\}$ for $1 \leq j \leq 15$. The DC coefficient 0 is the root of DCT coefficients tree, which has only three children: coefficients 1, 2 and 3. In the proposed structure, offspring corresponds to direct descendants in the same spatial location in the next finer band of the pyramid. A tree corresponds to a node having 4 children which always form a group of 2×2 adjacent pixels. In Fig. 3, arrows indicate that the same index coefficients of other 8×8 blocks are grouped together. In the proposed coefficient rearrangement algorithm, further one or two levels (depending upon the block size) of coefficient rearrangement is performed on the coarsest band. Therefore, an overall five levels of DCT coefficients are subjected to be processed by the proposed algorithm.

4. Memory allocation

The number of coefficients in the DC band is $I_{dc} = R_{dc} \times C_{dc}$, where $R_{dc} = R \times 2^{-L}$, $C_{dc} = C \times 2^{-L}$. R is the number of rows, C is the number of columns and L is the number of subband decomposition levels. The number of block trees in a root block of size (2×2), (4×4), and (8×8) required for LBTC is 1/4th, 1/16th and 1/64th of that required by NLS respectively. This reduces the initial cost by 25%, 6.25% and 1.25% respectively compared to NLS.

The coefficients are stored in one dimensional array of length I. If W bits are needed for each sub band coefficients, then the bulk storage memory required is IW for the subband data and $RC/2$ (half byte per pixel) for the eight state table markers. Therefore,

$$M_{LBTC} = IW + \frac{RC}{2} \tag{7}$$

Hence, from Eq. 7, the total memory required by LBTC is 704 kB (assuming $W = 18$ bits/coefficients), which is 22.3% more than the image alone.

4.1. Dynamic memory requirement comparison

In case of SPIHT, assuming total number of list entries (LIP and LSP) is twice the number of coefficients RC and LIS $\simeq RC/4$. At the end of all passes, the total memory required by SPIHT is

$$M_{SPIHT} = \frac{RC}{4} \times (8W + 1) \tag{8}$$

The total number of list entries are less than the number of coefficients because of block nature of WBTC. Therefore, total memory required at any time is less than SPIHT. Though WBTC requires memory approximately 1/3rd of SPIHT for some earlier bit plane passes, by the end of all passes the memory required is almost 85–95% as that required by SPIHT. Therefore, the worst case memory required by WBTC is

$$M_{WBTC} = \sim 0.95 \times M_{SPIHT}. \tag{9}$$

For a 512×512 image with 18 bits per coefficient, this is $(512 \times 512)/1024 \times 1/2$ (bytes/coeff) = 128 kB for LBTC, 1160 kB for SPIHT and 1100 kB for WBTC.

5. Experimental results and performance comparison

The performance of LBTC and DCT_LBT algorithms are compared in terms of coding efficiency and computational complexity with other algorithms such as SPIHT, WBTC, NLS, JPEG 2000, Song [19] and Hou [15] on standard monochrome images. The implementation is done in MATLAB 7.10.0 under Window XP, Intel Core 2 Duo CPU with 3 GHz speed and 4 GB of RAM space.

5.1. Coding efficiency of LBTC embedded wavelet coder

The performance of LBTC algorithm is evaluated on two sets of standard 8 bits/pixel (bpp) monochrome images. The first set includes Lena and Barbara, each of size 512×512. The second set includes higher resolution images from JPEG 2000 test set namely Bike and Woman of size 2560×2048. Five levels of dyadic wavelet decomposition are carried out using 9/7-biorthogonal filter [20] with symmetric extension at the image edges.

In order to evaluate the coding performance, we first compare the cumulative number of bits generated in different passes of NLS and proposed LBTC(2×2) and LBTC(4×4). The results are shown in Table 1 for Lena and Barbara images. It is evident from the Table 1 that, LBTC(2×2) saves 51.3%, 37.7%, 22.3%, 9.8%, 3.6%, 1.4% of bits, and LBTC(4×4) saves 58.9%, 43.1%, 25%, 10.9%, 3.8%, 10%, 1.4% of bits respectively on top six bit plane passes. This give rise to a peak-signal-to-noise-ratio (PSNR) improvement of (1.69–2.01 dB), (1.68–1.79) dB, (0.80–0.88) dB and (0.22–0.24) dB over NLS. On Barbara image LBTC(2×2) saves 64.6%, 42%, 23.4%, 10%, and LBTC(4×4) saves 72.3%, 42%, 26.3% of bits respectively on top four passes. Therefore, the PSNR improvement is (2.08–2.29) dB, (1.62–1.83) dB, (0.47–0.51) dB, and 0.22 dB for the corresponding passes respectively. This indicates that LBTC is more efficient in sorting significant coefficients than NLS. It can be verified that the percentage of bit saving in LBTC is nearly same as WBTC in most of the images.

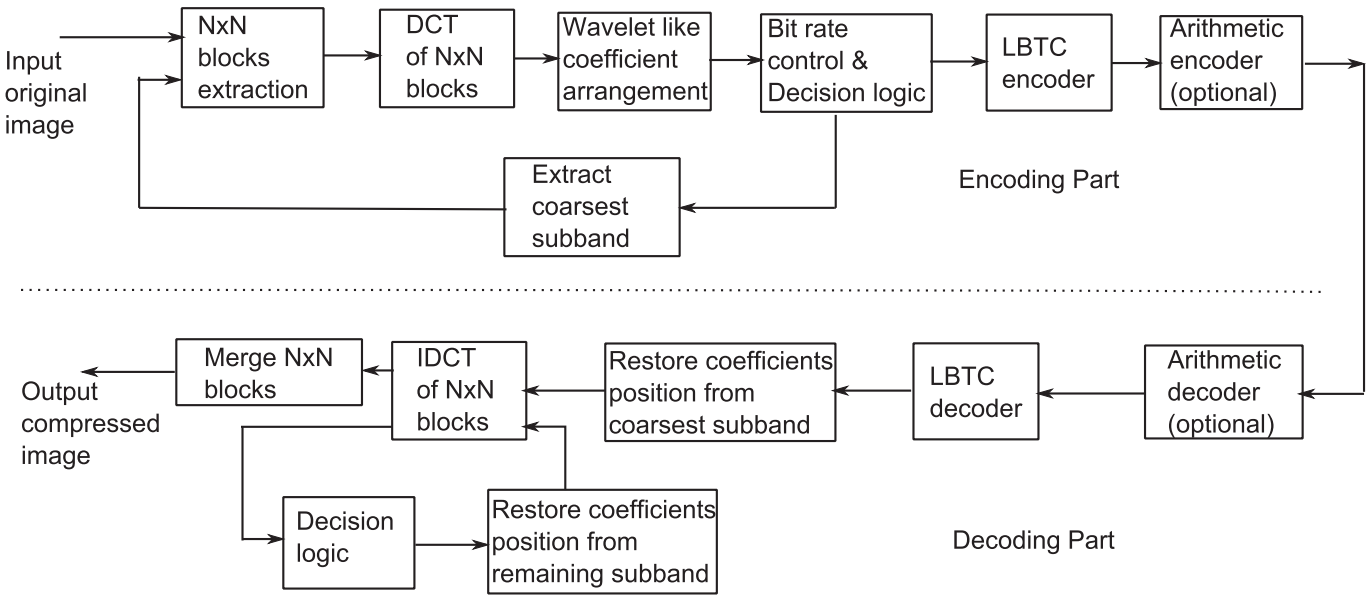

Fig. 2. Block diagram of proposed DCT_LBT embedded image coder.

Table 2 shows comparison of PSNR values between SPIHT, NLS, WBTC(2×2), LBTC(2×2), LBTC(4×4) and LBTC(8×8) for the same set of images at very low bit rates. It can be observed that the average PSNR loss of LBTC(2×2) over WBTC(2×2) is 0.126 dB on Lena image, whereas, the average PSNR gain of LBTC(2×2) over WBTC(2×2) on Barbara image is 0.473 dB over (0.005–0.1) bit rates. However, by increasing the root block size of LBTC, the average rate distortion (R-D) performance increases slightly over WBTC.

In Fig. 4, it is observed that the R-D performance of Lena and Barbara images using LBTC(2×2) algorithm is nearly same as WBTC (2×2) on (0.0–2.0) bpp. However, on Lena image, a slight fall in performance is observed in LBTC at higher bit rates. On the contrary, LBTC shows an improvement in performance over all the considered bit rates in Barbara image.

Table 3 lists out the comparisons of LBTC algorithm with SPIHT, NLS and WBTC algorithms over (0.0625–2.0) bit rates on Woman and Bike images without arithmetic coding. It can be seen that LBTC shows a coding gain of (0.08–1.42) dB with respect to WBTC, and (0.38–1.67) dB with respect to SPIHT over bit rates (0.0625–0.5) bpp on Woman image. However, the gain in LBTC falls by (0.91–1.0) dB and (0.59–0.73) dB over rates (1.0–2.0) bpp on WBTC and SPIHT respectively. Comparing with NLS, LBTC shows a coding gain of

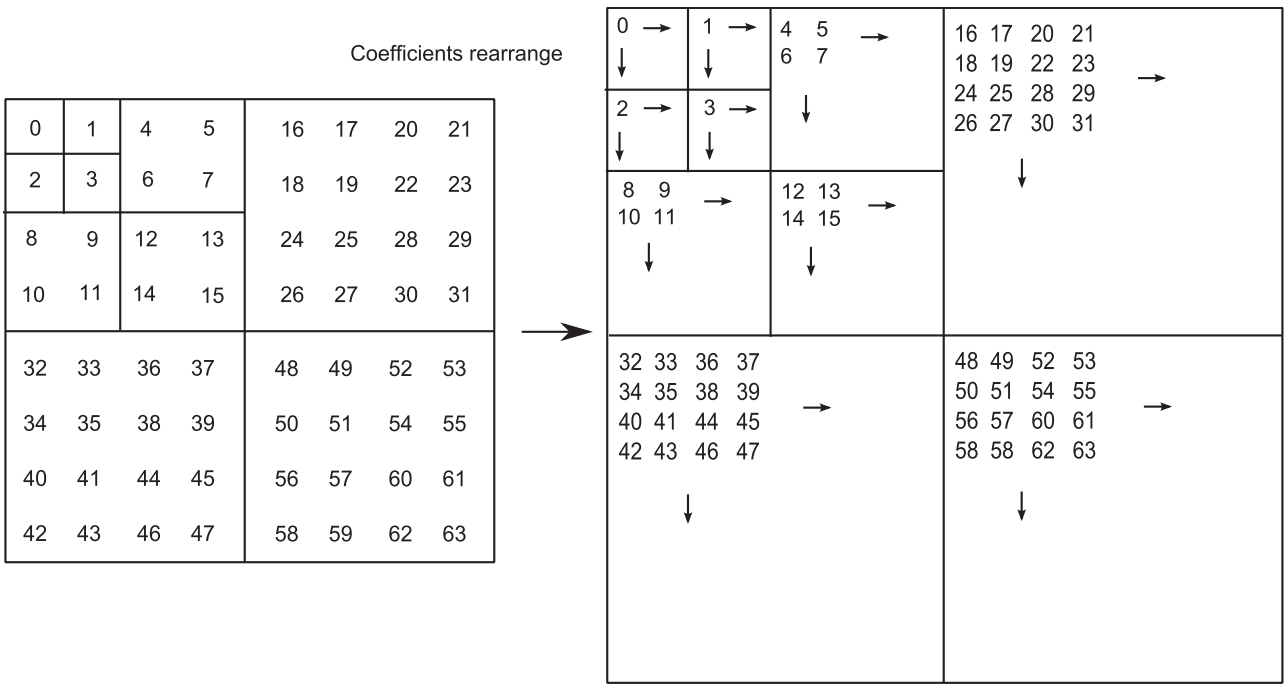

Fig. 3. Rearrangement algorithm of transformed coefficients (a three levels of coefficient arrangement is shown here for illustration).

Table 1
Comparison of encoder output string length between NLS and LBTC for Lena and Barbara images on top six cumulative bit plane passes.

Images	No. of sorting passes	NLS encoding string-length (no. of significant coeff.)	LBTC(2×2) encoding string-length (no. of significant coeff.)	LBTC(4×4) encoding string-length (no. of significant coeff.)	PSNR(dB) improvement LBTC(2×2) vs. NLS	PSNR(dB) improvement LBTC(4×4) vs. NLS
Lena	1	481(29)	234(29)	198(29)	1.69	2.01
	2	1041(106)	649(106)	593(106)	1.68	1.79
	3	2015(262)	1566(262)	1511(262)	0.80	0.88
	4	4296(571)	3874(571)	3825(571)	0.22	0.24
	5	10483(1422)	10105(1422)	10087(1422)	0.17	0.17
	6	23754(3367)	23430(3367)	23420(3367)	0.07	0.07
Barbara	1	469(19)	166(19)	130(19)	2.08	2.29
	2	1023(98)	593(98)	532(98)	1.62	1.83
	3	1952(256)	1496(256)	1440(256)	0.47	0.51
	4	4139(564)	3725(564)	3670(564)	0.22	0.22
	5	11444(1544)	11100(1544)	11045(1544)	0.06	0.07
	6	44476(17756)	44150(17756)	44095(17756)	0.04	0.04

Table 2
Comparison of PSNR(dB) values between SPIHT, NLS, WBTC and LBTC at very low bit rates.

Images	Algorithm	Bit rate (bpp)				
		0.005	0.01	0.03	0.05	0.1
Lena	SPIHT	19.46	21.94	25.22	27.06	29.71
	NLS	19.39	21.97	25.20	26.97	29.61
	WBTC(2×2)	20.74	22.52	25.48	27.19	29.78
	WBTC(4×4)	20.80	22.54	25.49	27.20	29.79
	WBTC(8×8)	20.82	22.55	25.50	27.20	29.79
	LBTC(2×2)	20.60	22.40	25.44	27.00	29.64
	LBTC(4×4)	20.77	22.51	25.45	27.00	29.64
	LBTC(8×8)	20.79	22.51	25.45	27.00	29.64
Barbara	SPIHT	17.99	19.84	21.91	22.58	23.95
	NLS	18.86	20.53	22.42	23.18	23.98
	WBTC(2×2)	18.64	20.12	22.06	22.80	24.23
	WBTC(4×4)	18.82	20.22	22.21	22.89	24.25
	WBTC(8×8)	19.00	20.32	22.29	22.95	24.29
	LBTC(2×2)	19.66	20.88	22.51	23.21	23.98
	LBTC(4×4)	19.71	20.93	22.52	23.22	23.98
	LBTC(8×8)	19.73	20.95	22.52	23.22	23.98

All the results are without arithmetic coding.

(0.01–0.1) dB over the considered bit rates on Woman image. In Bike image, LBTC shows a coding gain of (0.01–0.43) dB on WBTC and (0.26–0.63) dB on SPIHT for bit rates (0.0625–0.25) bpp. However, the coding gain reduces to approximately (0.34–0.65) dB in WBTC and (0.19–0.51) dB in SPIHT over bit rates (0.5–2.0) bpp. LBTC consistently shows a PSNR gain between (0.01–0.09) dB over NLS in all the bit rates on Bike image.

Table 4 shows the PSNR performance comparisons of LBTC with JPEG 2000, SPIHT, NLS and WBTC algorithms using arithmetic coding [21] for the same set of images. Comparing with JPEG 2000, LBTC shows a significant gain (0.28–1.46) dB over (0.0625–0.5) bpp on Woman image. However, LBTC shows a coding gain of 0.12 dB at 0.0625 bpp and a loss of (0.04–0.72) dB over (0.125–2.0) bpp in Bike image. WBTC shows poor performance than LBTC below 0.5 bpp. At higher rates (approx. $\geq$1.0 bpp), WBTC outperforms LBTC by a wide margin. Summarizing all, LBTC outperforms most of the algorithm at lower rates (typically $\leq$1.0 bpp) and under performs at higher rates in most of the images. One possible reason for reduction of PSNR values at higher rates is that, as more and more coefficients became significant with decreasing thresholds, more bits are

Table 3
Comparison of PSNR(dB) values between SPIHT, NLS, WBTC and LBTC algorithm.

Images	Algorithm	Bit rate (bpp)					
		0.0625	0.125	0.25	0.5	1.0	2.0
Woman	SPIHT	25.07	26.91	29.43	32.93	37.73	43.21
	NLS	26.61	28.34	30.56	33.30	36.99	42.60
	WBTC	25.29	27.08	29.71	33.23	37.91	43.62
	LBTC	26.71	28.38	30.59	33.31	37.00	42.62
Bike	SPIHT	22.86	25.29	28.51	32.36	36.98	42.90
	NLS	23.40	25.86	28.75	32.15	36.46	42.53
	WBTC	23.14	25.48	28.76	32.51	37.12	43.13
	LBTC	23.49	25.91	28.77	32.17	36.47	42.55

All the results are without arithmetic coding.

Table 4
Comparison of PSNR(dB) values between JPEG 2000, SPIHT, NLS, WBTC and LBTC algorithm.

Images	Algorithm	Bit rate (bpp)					
		0.0625	0.125	0.25	0.5	1.0	2.0
Woman	JPEG2000	25.59	27.33	29.95	33.57	38.28	43.97
	SPIHT	25.50	27.33	29.94	33.59	38.28	43.99
	NLS	26.89	28.74	30.86	33.77	37.88	43.60
	WBTC	25.67	27.34	30.05	33.73	38.30	44.10
	LBTC	27.05	28.98	30.96	33.85	37.92	43.62
Bike	JPEG2000	23.74	26.31	29.56	33.43	37.99	43.95
	SPIHT	23.44	25.89	29.12	33.01	37.70	43.80
	NLS	23.70	26.20	29.25	32.75	37.22	43.23
	WBTC	23.61	25.97	29.29	33.17	37.82	43.93
	LBTC	23.86	26.27	29.31	32.91	37.27	43.27

All the results are with arithmetic coding.

required to indicate the significance of block-tree as well as each tree inside a block-tree. This leads to a cumulative increase of bit string length at the encoder output. Bits indicating significance of block-trees are required, but these bits do not carry information anything related to PSNR. This leads to reduction of zero distortion with non zero increase of bit rate. This effect is more dominant in images of higher dimensions. This is because, higher dimension images having more block-trees compared to lower dimension images.

5.2. Coding efficiency of DCT_LBT embedded coder

Table 5 shows a comparison of cumulative number of bits generated on top six bit plane passes of Lena and Barbara images using DCT_LBT coder. It is observed that DCT_LBT(4×4) encoder saves 6.0–82.8% of bits on Lena and 3.8–92.5% of bits on Barbara images for the considered range of bit plane passes. This give rise to a PSNR gain of (0.27–2.85) dB and (0.17–2.66) dB on Lena and Barbara image respectively.

Input image partitioned into (8×8) blocks, (16×16) blocks, and (32×32) blocks-I use the coefficient rearrangement algorithm as explained in Case-1; whereas, (32×32) blocks-II uses the coefficient arrangement as explained in Case-2 of Section 3. Fig. 5 shows the coding performance of DCT_LBT(4×4) algorithm on (a) Lena, (b) Barbara, (c) Boat and (d) Mandrill images using the above block sizes associated with their respective coefficient arrangements. It is clearly evident from the results that (16×16) block sizes show a good trade-off of coding performance over the range of considered bit rates on most of the images. (32×32) blocks-II shows slightly better performance at lower rates. However, its performance is seriously affected at higher rates, typically $\geq$1.0 bpp in most of the standard images.

Fig. 6(a)–(d) shows the decoded cropped portions of Woman image when coded using DCT_SPIHT, DCT_NLS, DCT_LBT(2×2) and DCT_LBT(4×4) at 0.05 bpp, while Fig. 7(a)–(d) displays the decoded cropped portions of Bike image at 0.1 bpp using same set of coding schemes. It is observed that the decoded images using DCT_LBT coding schemes have better perceptual qualities than other coding schemes. It is to be noted that blocking effect is manifested in the above simulations. These effects vanishes gradually with the increase of bit rate.

Tables 6 and 7 show a comparison of R-D performance of DCT_LBT(4×4) algorithm with other state-of-the-art DCT coders presented in [15,19], including JPEG 2000, for (512×512) size test images (Peppers, Boat, Mandrill, Lena and Barbara) and (2560×2048) size images (Woman, Bike and Cafe). The proposed algorithm uses input block size (16×16). These binary coded

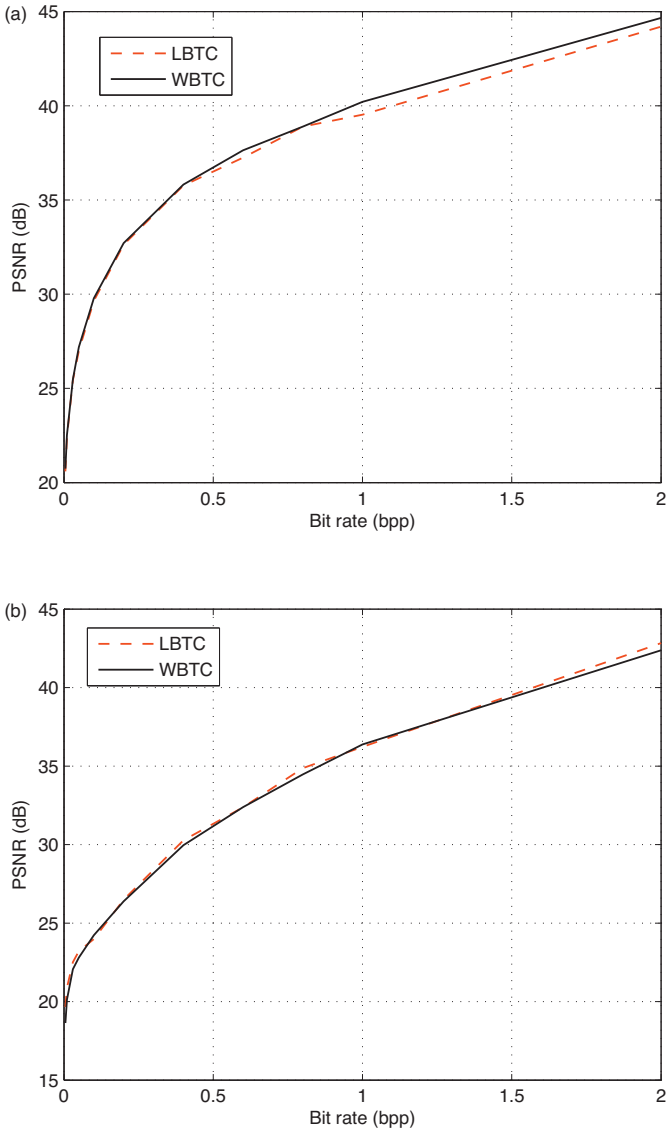

Fig. 4. Rate distortion performance comparison of WBTC(2×2) and LBTC(2×2) algorithms on (a) Lena and (b) Barbara images (no back-end arithmetic coding employed).

Table 5
Comparison of encoder output string length between DCT_NLS and DCT_LBT for Lena and Barbara image on top six cumulative bit plane passes.

Images	No. of sorting passes	DCT_NLS encoding string-length (no. of significant coeff.)	DCT_LBT(2×2) encoding string-length (no. of significant coeff.)	DCT_LBT(4×4) encoding string-length (no. of significant coeff.)	% of bit saving DCT_LBT(4×4) vs. DCT_NLS	PSNR(dB) improvement
Lena	1	458(6)	115(6)	79(6)	82.8	2.85
	2	925(25)	301(25)	229(25)	75.3	2.46
	3	1473(76)	653(76)	561(76)	62.0	1.43
	4	2799(243)	1949(243)	1853(243)	33.8	0.74
	5	6544(761)	5708(761)	5623(761)	14.1	0.54
	6	15082(1963)	14268(1963)	14188(1963)	6.0	0.27
Barbara	1	453(1)	70(1)	34(1)	92.5	2.66
	2	926(21)	240(21)	172(21)	81.4	2.31
	3	1452(70)	554(70)	461(70)	68.3	1.32
	4	2873(252)	1980(252)	1884(252)	34.4	0.75
	5	8280(822)	7469(822)	7376(822)	10.9	0.34
	6	23375(2486)	22577(2486)	22484(2486)	3.8	0.17

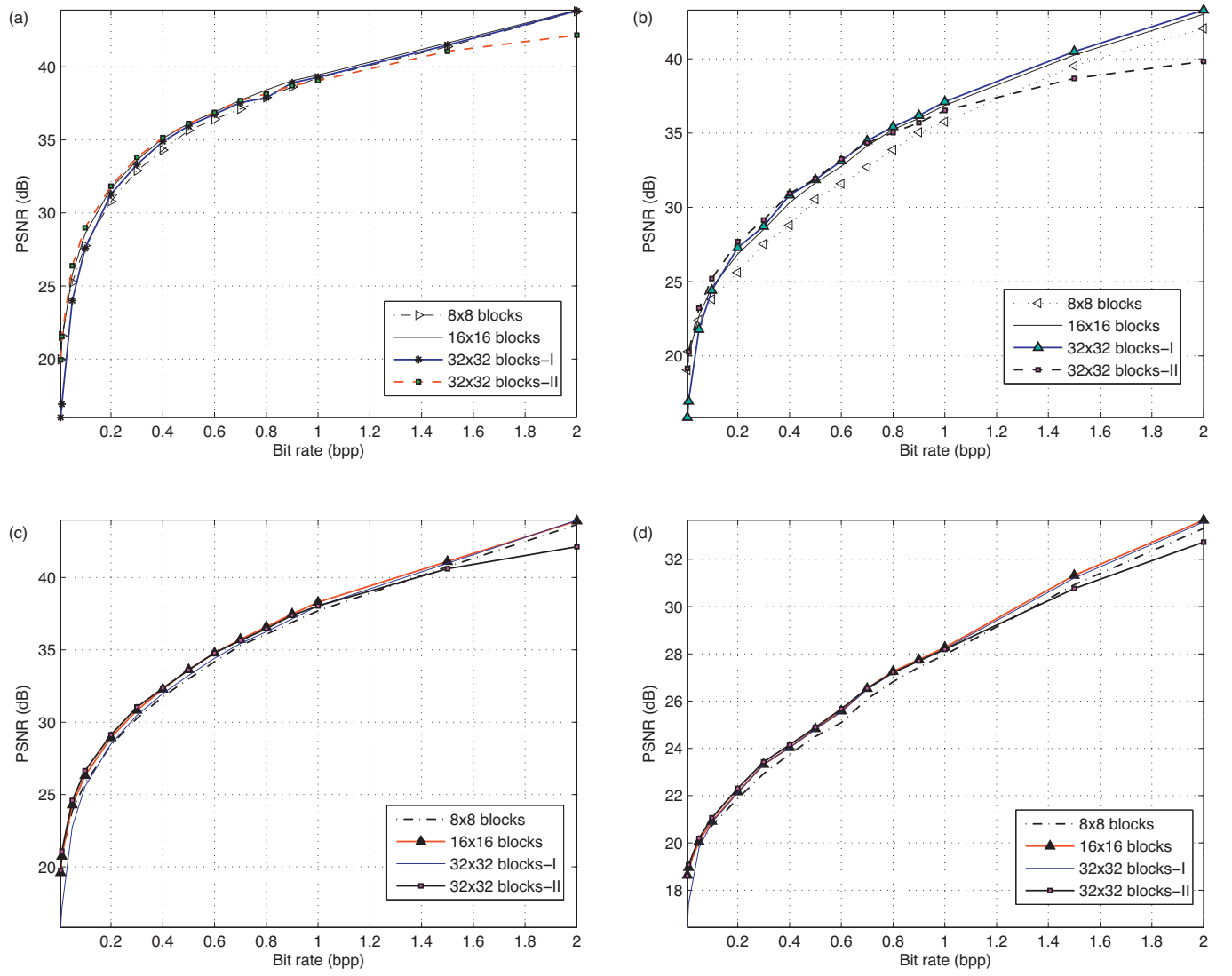

Fig. 5. Rate distortion performance comparison of DCT_LBT (4×4) algorithm for input block sizes of (8×8), (16×16), (32×32)blocks-I, (32×32)blocks-II on (a) Lena, (b) Barbara, (c) Boat and (d) Mandrill images, without using arithmetic coding.

version results (Arithmetic coding) of other algorithms are taken directly from [15,19].

When compare with the results by Song [19] in Table 6, it is clear that the proposed algorithm shows a PSNR gain of (0.3–1.33) dB between (0.0625–0.5) bpp and a loss of (0.37–0.76) dB between rates (1.0–2.0) bpp on Woman image. On bike image, the gain is (0.06–1.27) dB over (0.0625–0.5) bpp and a loss of (0.9–0.98) dB above 0.5 bpp is observed. Similarly, a PSNR gain of (0.37–1.56) dB is achieved over (0.0625–2.0) bit rates in Cafe image. While comparing with JPEG 2000, the proposed technique shows approximately

Table 6
Comparison of PSNR(dB) values between Hou [15], JPEG 2000, Song [19] and DCT_LBT(4×4) algorithms on 2560×2048 size images.

Images	Algorithm	Bit rate (bpp)					
		0.0625	0.125	0.25	0.5	1.0	2.0
Woman	Hou [15]	24.90	26.69	28.63	31.70	35.98	41.34
	JPEG2000	25.59	27.35	29.99	33.62	38.42	43.99
	Song [19]	25.60	27.29	29.84	33.38	37.89	43.77
	DCT_LBT	26.85	28.62	30.87	33.68	37.52	43.01
Bike	Hou [15]	22.18	24.47	27.35	30.73	34.77	39.91
	JPEG2000	23.80	26.36	29.62	33.51	38.10	43.98
	Song [19]	23.24	25.75	28.60	32.08	36.57	42.57
	DCT_LBT	23.65	26.32	29.52	32.65	36.86	42.72
Cafe	Hou [15]	18.21	19.61	21.50	24.49	28.66	34.30
	JPEG2000	19.05	20.76	23.13	26.81	32.03	39.07
	Song [19]	18.73	20.46	22.83	25.87	30.50	37.21
	DCT_LBT	19.74	21.52	23.91	27.43	31.12	37.58

All the results are with arithmetic coding.

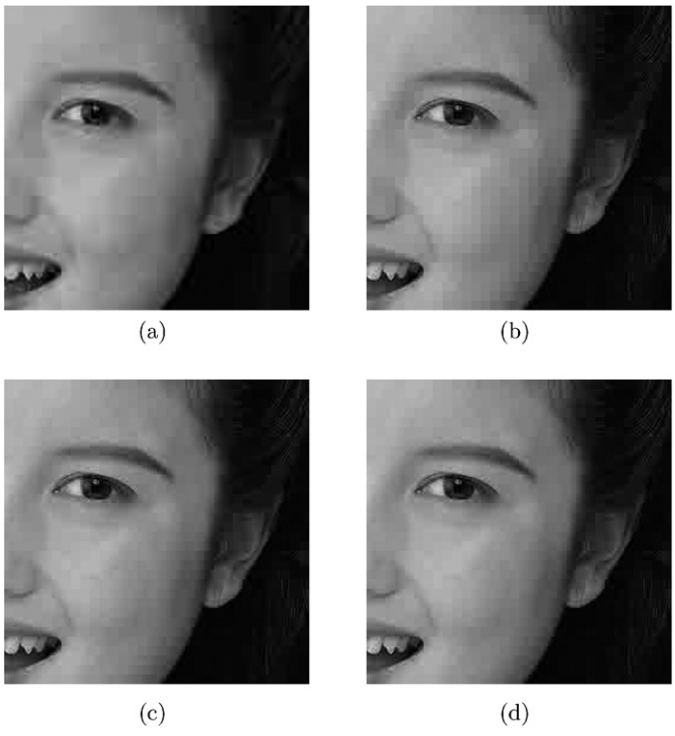

Fig. 6. Decoded Woman images compressed at 0.05 bpp using (a) DCT_SPIHT, (b) DCT_NLS, (c) DCT_LBT(2 × 2), and (d) DCT_LBT(4 × 4) respectively.

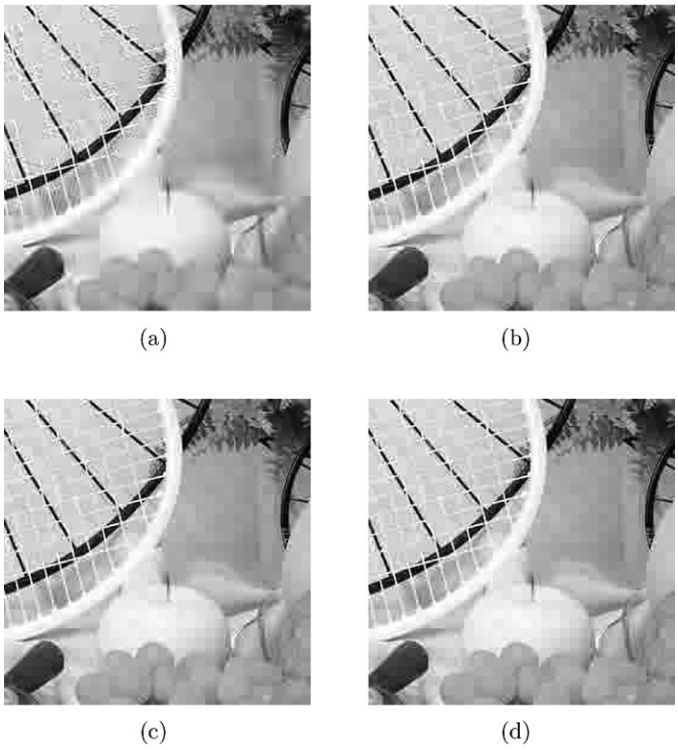

Fig. 7. Decoded Bike images compressed at 0.1 bpp using (a) DCT_SPIHT, (b) DCT_NLS, (c) DCT_LBT(2 × 2), and (d) DCT_LBT(4 × 4) respectively.

same or better results at lower bit rates (≤0.5 bpp) and worst results on higher rates (≥1.0 bpp) in most of the images. Improvement at lower rates is a desirable feature for browsing images over wireless lines where a significant amount of information is required at the earlier stages of transmission. Though the coding performances are lower at higher rates, the subjective quality of the image does not show any significant change even for a PSNR loss over 1.0 dB.

In Table 7, DCT_LBT(4 × 4) shows (0.3–1.28) dB PSNR improvement over Hou [15], PSNR loss of (0.04–0.8) dB with respect to JPEG 2000 and a gain of (0.03–0.50) dB with respect to Song [19] in Pepper image. Similarly, when comparing with Hou [15] the proposed algorithm shows a coding gain of (0.03–1.97) dB, (0.08–0.41) dB, (0.15–0.55) dB and (0.19–0.53) dB respectively on Lena, Barbara, Mandrill and Boat images over (0.0.0625–2.0) bit rates. It can be calculated from Table 7 that the proposed techniques show an average reduction of 0.004 dB with [19] and 0.2 dB with JPEG 2000 over the five set of images considered. It is also worth mentioning that DCT_LBT coder shows a significant improvement of PSNR values at lower bit rates than most of the DCT based embedded coders reported in [14–18].

Table 7
Comparison of PSNR(dB) values between Hou [15], JPEG 2000, Song [19] and DCT_LBT(4 × 4) algorithms on 512 × 512 size images.

Images	Algorithm	Bit rate (bpp)					
		0.0625	0.125	0.25	0.5	1.0	2.0
Peppers	Hou [15]	26.31	29.16	31.89	34.52	37.28	41.84
	JPEG2000	26.93	30.48	33.37	35.84	38.32	43.07
	Song [19]	26.67	27.47	32.09	34.67	37.61	42.49
	DCT_LBT	26.61	30.44	32.57	35.17	37.67	42.52
Boat	Hou [15]	25.17	27.47	30.08	33.70	38.40	44.26
	JPEG2000	25.45	27.81	30.92	34.50	39.21	44.69
	Song [19]	25.54	27.71	30.47	34.09	38.60	44.51
	DCT_LBT	25.52	27.66	30.52	34.12	38.93	44.66
Mandrill	Hou [15]	20.40	21.25	22.99	25.03	28.48	34.03
	JPEG2000	20.60	21.60	23.12	25.53	29.00	34.73
	Song [19]	20.59	21.66	23.11	25.56	28.97	34.78
	DCT_LBT	20.55	21.56	23.21	25.43	28.66	34.58
Lena	Hou [15]	–	29.42	32.88	36.37	39.68	–
	JPEG2000	27.80	30.79	33.97	37.24	40.34	–
	Song [19]	27.57	30.28	33.36	36.64	39.93	–
	DCT_LBT	27.49	30.17	33.41	36.72	40.08	44.82
Barbara	Hou [15]	–	25.43	28.54	32.29	37.05	–
	JPEG2000	23.17	25.25	28.30	32.17	37.12	–
	Song [19]	24.06	26.43	29.27	32.82	37.52	–
	DCT_LBT	23.52	25.84	28.62	32.39	37.41	43.81

All the results are with arithmetic coding.

Table 8
Comparison of encoding and decoding times (s) for Lena image.

Algorithm	Bit rate (bpp)					
	0.0625	0.125	0.25	0.5	1.0	2.0
Encoding time						
SPIHT	0.67	1.26	2.76	7.27	25.14	139.07
NLS	5.26	6.30	8.06	9.69	11.82	14.89
LBTC(2×2)	7.02	8.23	10.13	11.83	14.13	17.33
LBTC(4×4)	7.19	8.43	10.40	12.18	14.55	17.63
LBTC(8×8)	7.47	8.50	10.49	12.36	14.71	17.94
Decoding time						
SPIHT	0.15	0.45	1.47	5.74	21.55	44.35
NLS	4.97	5.75	7.20	8.19	9.39	11.22
LBTC(2×2)	6.71	7.73	9.25	10.43	11.79	13.67
LBTC(4×4)	6.54	7.46	8.98	10.07	11.44	13.29
LBTC(8×8)	6.30	7.06	8.49	9.56	10.87	12.80

Wavelet transform/DCT transforms are not included.

5.3. Computational complexity

The execution time determines the complexity of an algorithm. It is observed from the Table 8 that execution time of list based coders (e.g. SPIHT) increases exponentially with respect to coders without lists (viz. NLS and LBTC). During encoding, SPIHT shows 1.3–7.8 times faster below 0.5 bpp and 2–9 times slower above 0.5 bpp in Lena image. Similar kind of performance is also noticed during decoding. In other words the average encoding time in SPIHT is 29.36 s and decoding time is 12.29 s, whereas, it is 10.26 s and 9.18 s respectively in NLS over the considered bit rates. The proposed LBTC algorithm is slightly more complex than NLS. The reason could be additional block partitioning steps where a significant block tree will be recursively quad partitioned to find significant coefficients for each pass. A slight increase of encoding time and a slight decrease of decoding time is observed for larger root block sizes. One possible reason is that larger blocks consume more time during encoding, whereas, reconstructing a pixel from a larger set is a less time consuming operation because of efficient skipping of predictable insignificance sets/blocks. Similar kind of performance is noticed in other popular images.

6. Conclusion

In this paper, a listless block tree coder (LBTC), which is a no list variant of WBTC algorithm is proposed. WBTC uses two dimensional block trees, whereas, LBTC uses one dimensional block tree structures to exploit inter and intra scale correlation. This in effect, enhances the low bit rate performance of most of the wavelet and DCT based embedded coders, including listless SPIHT (NLS). The memory requirement of the proposed coder is fixed which is almost 22.3% more than the image alone. The proposed coders (LBTC and DCT_LBT) manifest performance degradation at higher bit rates, typically $\geq$1.0 bpp. This does not impose any serious problem, as at higher rates, the subjective quality of an image remains indistinguishable to slight reduction of PSNR values. It is to be noted that, the proposed coders are little complex than NLS, but much lesser than JPEG 2000 (as it uses multiple coding passes within each bit plane, context adaptive arithmetic coding and rate-distortion optimization) and embedded DCT coder proposed by Song and Cho (It couples with a complex kind of coefficient sorting method with context based adaptive arithmetic coding).

Therefore, the proposed coders are suitable for low memory hand held devices (e.g. digital camera, PDAs, mobile phones, etc.) in particular and image communications in general, where a significant amount of information is to be transmitted within the available bandwidth.

References

[1] Santa-cruz D, Grosbois R, Ebrahimi T. JPEG 2000 performance evaluation and assessment. Signal Process: Image Commun 2002;17(1):113–30.
[2] Taubman D. High performance scalable image compression with EBCOT. IEEE Trans Image Process 2000;9(7):1158–70.
[3] Shapiro JM. Embedded image coding using zerotrees of wavelet coefficients. IEEE Trans Signal Process 1993;41(12):3445–62.
[4] Said A, Pearlman WA. A new fast and efficient image codec based on set partitioning in hierarchical trees. IEEE Trans Circuits Syst Video Technol 1996;6(3):243–50.
[5] Liu G, Zeng X, Tian F, Chaibou K, Zheng Z. A novel direction-adaptive wavelet based image compression. Int J Electron Commun, Elsevier 2010;64(6):531–9.
[6] Chai BB, Vass J, Zhuang X. Significance-linked connected component analysis for wavelet image coding. IEEE Trans Image Process 1999;8(6):774–84.
[7] Wheeler F, Pearlman WA. SPIHT image compression without lists. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 4. 2000. p. 2047–50.
[8] Moinuddin AA, Khan E, Ghanbari M. Efficient algorithm for very low bit rate embedded image coding. IET Image Process 2008;2(2):59–71.
[9] Moinuddin AA, Khan E, Ghanbari M. Low complexity efficient and embedded color image coding technique. IEEE Trans Consum Electron 2008;54(2):787–94.
[10] Pan H, Siu WC, Law NF. A fast and low memory image coding algorithm based on lifting wavelet transform and modified SPIHT. Signal Process: Image Commun 2008;23(3):146–61.
[11] Pennebakr WB, Mitchell JL. JPEG still image compression standard. New York: Chapman Hall; 1993.
[12] Douak F, Benzid R, Benoudjit N. Color image compression algorithm based on DCT transform combined to an adaptive block scanning. Int J Electron Commun, Elsevier 2011;65(1):16–26.
[13] Richardson IE. H.264 and MPEG-4 video compression. John Wiley and Sons; 2003.
[14] Davis GM, Chawla S. Image coding using optimised significance tree quantization. In: IEEE data compression conference. 1997. p. 387–96.
[15] Hou X, Liu G, Zou Y. Embedded quadtree-based image compression in DCT domain. In: Proc. IEEE int. conf. acoustics, speech and signal processing, vol. 3. 2003. p. 277–80.
[16] Junqiang L, Zhuang X. Embedded image compression using DCT based subband decomposition and slcca data organisation. In: IEEE workshop multimedia signal processing. 2002. p. 81–4.
[17] Monoro DM, Dickson GJ. Zerotree coding of DCT coefficients. Proc IEEE Int Conf Image Process 1997;2:625–8.
[18] Servetto S, Ramchandran K, Orchand MT. Image coding based on morphological representation of wavelet data. IEEE Trans Image Process 1999;8(9):1161–74.
[19] Song HS, Cho NL. DCT-based embedded image compression with a new coefficient sorting method. IEEE Signal Process Lett 2009;16(5):410–3.
[20] Antonini M, Barlaud MM, Mathieu P, Daubechies I. Image coding using wavelet transform. IEEE Trans Image Process 1992;1(2):205–20.
[21] Witten IB, Neal RL, Cleary JG. Arithmetic coding for data compression. Commun ACM 1987;30(6):520–40.