Available online at www.elixirpublishers.com (Elixir International Journal)

Digital Processing

Elixir Digital Processing 94 (2016) 40368-40374

ISSN: 2229-712X

ZJICD algorithm for JPEG image compression/decompression

Ziad A. Alqadi[1], Majed Omar Al-Dwairi[1,2] ,Hatim Zaini[2] , and Mohamed S. Soliman[2,3]

[1]Albalqa Applied University/Faculty of Engineering Technology, Amman-Jordan.

[2]Taif University / Department of Electrical Engineering, Taif , Kingdom of Saudi Arabia.

[3]Department of Electrical Engineering / Faculty of Energy Engineering, Aswan University, Aswan, Egypt

ARTICLE INFO

Article history:

Received: 6 April 2016;

Received in revised form:

9 May 2016;

Accepted: 14 May 2016;

Keywords

ZJICD Ziad JPG image compression decompression, JPEG image, DCT, IDCT, Compression ratio, NOISE to signal ratio.

ABSTRACT

The algorithm of the JPEG image compression based on DCT and IDCT was developed and implemented . The proposed algorithm can cope with the problem of impairing dependency of noised digital image. The result indicates that the algorithm is an effective way for gray scale image compression and the image rebuilt is acceptable. The practical experiments were done with MATLAB7.0. The algorithm doing experiments with MATLAB is simple and with little error, and it can improve the efficiency and precision of the image compression greatly.

I. Introduction

Many compression algorithms have been implemented in the past years. Some of which are of general use, *i.e.*, can be used to compress files of different types (e.g., text files, image files, video files, etc.). Others are developed to compress efficiently a particular type of files. It has been realized that, according to the representation form of the data at which the compression process is performed, below is reviewing some of the literature review in this field.

Lossless image compression [1] with four modular components: pixel sequence, prediction, error modeling, and coding. They used two methods that clearly separate the four modular components. These method are called Multi-Level Progressive Method (MLP), and Partial Precision Matching Method (PPMM) for lossless compression, both involving linear predictions, modeling prediction errors by estimating the variance of a Laplace distribution (symmetric exponential), and coding using arithmetic coding applied to pre-computed distributions [1].

A composite modeling method (hybrid compression algorithm for binary image) is used to reduce [2], the number of data coded by arithmetic coding, which code the uniform areas with less computation and apply arithmetic coding to the areas. The image block is classified into three categories: all-white, all-black, and mixed, then image processed 16 rows at a time, which is then operated by two global and local stages [2].

An algorithm that works by applying a reversible transformation on the fourteen commonly used files of the Calgary Compression Corpus.[3] It does not process its input sequentially, but instead processes a block of texts as a single unit, to form a new block that contains the same characters, but is easier to compress by simple compression algorithms, group characters together based on their contexts. This technique makes use of the context on only one side of each character so that the probability of finding a character closer to another instance of the same character is increased substantially. The transformation does not itself com-press the data, but reorder it to make it easy to compress with simple algorithms such as move-to-front coding in combination with Huffman or arithmetic coding [3].

Lossless grayscale image compression method—TMW—is based on the use of linear predictors and implicit segmentation.[4] The compression process is split into an analysis step and a coding step. In the analysis step, a set of linear predictors and other parameters suitable for the image are calculated in the analysis step in a way that minimizes the length of the encoded image which is included in the compressed file and subsequently used for the coding step. To do the actual encoding, obviously, the chosen parameter set has to be considered as a part of the encoded image and has to be stored or transmitted alongside with the result of the Coding Stage [4].

A lossless compression scheme for binary images which consists of a novel en-coding algorithm and uses a new edge tracking algorithm.[5] The proposed scheme consists of two major steps: the first step encodes binary image data using the proposed encoding method that encodes image data to only characteristic vector information of objects in image by using a new edge tracing method. Unlike the existing approaches, our method encodes information of edge lines obtained using the modified edge tracing method instead of directly encoding whole image data. The second is com-pressing the encoded image Huffman and Lempel-Ziv-Welch (LZW) [5].

An algorithm for lossless binary image compression which consists of two modules, called Two Modules Based Algorithm (TMBA), the first module: direct redundancy exploitation and the second: improved arithmetic coding [6].

Tele:

E-mail address: majeddw@gmail.com

a two-dimensional dictionary-based on lossless image compression scheme for grayscale images is introduced.[7] The proposed scheme reduces a correlation in image data by finding two-dimensional blocks of pixels that are approximately matched throughout the data and replacing them with short code words.

In [7], the two-dimensional Lempel-Ziv image compression scheme (denoted GS-2D-LZ) is proposed. This scheme is designed to take advantage of the 2 dimensional signal correlations in the image data. It relies on three different compression strategies, namely: two-dimensional block matching, prediction, and statistical encoding.

A lossless image compression method that is based on Multiple-Table's Arithmetic Coding (MTAC) method to encode a gray-level image, [8]first classifies the data and then encodes each cluster of data using a distinct code table. The MTAC method employs a median edge detector (MED) to re-duce the entropy rate of f. The gray levels of two adjacent pixels in an image are usually similar. A base-switching transformation approach is then used to reduce the spatial redundancy of the image. The gray levels of some pixels in an image are more common than those of others. Finally, the arithmetic encoding method is applied to re-duce the coding redundancy of the image [8].

A lossless method of image compression and decompression is proposed. It uses a simple coding technique called Huffman coding. A soft-ware algorithm has been developed and implemented to compress and decompress the given image using Huff-man coding techniques in a MATLAB platform. They concern with compressing images by reducing the number of bits per pixel required to represent it, and to de-crease the transmission time for images transmission. The image is reconstructed back by decoding it using Huff-man codes [9].

ZJICD algorithm can be implemented into 2 phases :

1. Compression phase
2. Decompression phase

Compression phase

This phase contains the following sequence of procedures which are to be implemented in order to create a JPEG encryptor which can be used to compress the image

a) Capture the original image.
b) Original image transformation to approximate color space.
c) Generate 8x8 image blocks
d) DCT computation
e) Zigzag Scanning and Block Scrambling
f) Convert to symbols
g) Coding the 8x8 blocks

Decompression phase

This phase creates JPEG decoder which can be used to decompress the image and it can be implemented by applying the following sequence:

a) Variable length decoding using DC decoding and AC decoding
b) Reverse Zigzag

ZJICD algorithm implementation

A matlab code was built for the proposed algorithm taking the following in assumption::

- Coding: Compression
- Codeword: A binary string representing either the whole coded data or one coded data symbol
- Coded Bit stream: the binary string representing the whole coded data.
- Lossless Compression: 100% accurate reconstruction of the original data
- Lossy Compression: The reconstruction involves errors which may or may not be tolerable
- Bit Rate: Average number of bits per original data element after compression

$$bitrate = \frac{number\ of\ bits\ in\ the\ coded\ bitstream}{number\ of\ samples\ (or\ pixels)}$$

- $CompressionRatio(CR) = \frac{size\ of\ the\ uncompressed\ signal}{size\ of\ the\ coded\ bitstream}$
- Signal-to-Noise Ratio (SNR) in the case of lossy compression.

Let I be an original signal (e.g., an image), and R be its lossily reconstructed counterpart. SNR is defined to be:

$SNR=10\ \log_{10}(\|I\|^2/\|I-R\|^2) = 20\ \log_{10}(\|I\|/\|I-R\|)$

where for any vector/matrix/set of number $E=\{x_1, x_2, ..., x_N\}$, $\|E\|^2=x_1^2 + x_2^2 + ... + x_N^2$.

- The unit of SNR is "decibel" (or dB for short).
- So, if SNR = 23, we say the SNR is 23 dB.
- Mean-Square Error (MSE): $\|I-R\|^2/N$
- Relative Mean-Square Error (RMSE): $\|I-R\|^2/\|I\|^2$
- Therefore, $SNR = -10\ \log_{10}$ RMSE.
- So the smaller the error, the higher the SNR. In particular, the higher the SNR, the better the quality of the reconstructed data.
- Exercise: Prove that if RMSE is decreased by a factor of 10, then SNR increases by 10 decibels.
- It is this nice fact that justifies the multiplicative factor in the definition of the SNR.

Appendix A contains The structures were used in the code

The program was tested and run and it gave an acceptalble results as shown in figures 1 and 2.

The objective of the proposed algorithm in this paper is to design an efficient and effective lossless image compression scheme. This section deals with the design of a loss- less image compression algorithm.

ZJICD compression algorithm, which is DCT – Discrete cosine transform and IDCT (Inverse Discrete Cosine Transform).[10,11] DCT is discrete-time version of Fourier-Cosine series. It is the class of the mathematical operations that include the Discrete Fourier Transform (DFT), and the Fast Fourier Transform (FFT), as well as many others. It is very closely related to the DFT- Discrete Fourier transform, a technique in signal processing for converting the elementary signal in to frequency component. Thus, DCT can be computed mathematically using the FFT (Fast Fourier Transform).

The 8x8 image block uses a set of 64 two-dimensional cosine basis function that are created by multiplying a horizontally oriented set of one-dimensional 8 point cosine basis function by vertically oriented set of same functions. The horizontally oriented set of cosine coefficient represents the horizontal frequencies and the other set of coefficients represents the vertical frequencies.

The DCT coefficients generated by the above equations can thus be regarded as the relative amount of the two dimensional spatial frequencies contained in the 64 point input signal. The coefficient with zero frequency in the both dimensions is called the "DC coefficient" and the remaining 63 coefficients are called "AC coefficients".

The proposed algorithm is based on creating JPEG encryptor for compression and standard JPEG decoder for image decryption. in order to improve the compression ratio of

the image comparing to other compression techniques in the literature review.

Figure 1. The original imag.

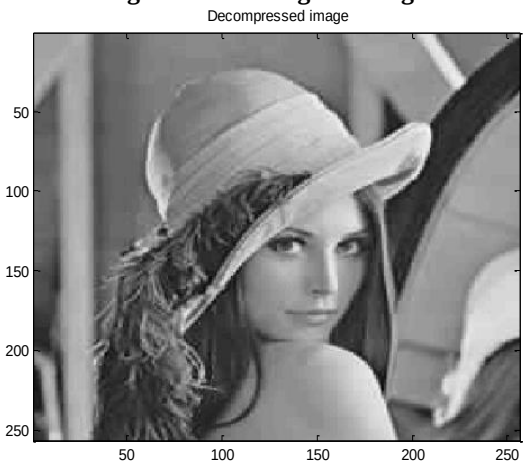

Figure 2. The decompressed image.

The code was implemented several times using various images. Table 1 shows the implemetaiom results

Results discussion

We can see from the implementation results that the proposed ZJICD algorithm can be used as an efficient algorithm to compress-decompress JPEG images and it needs only 0,5 Mbyte to be implemented and an average running time of 2.5 seconds for compression and 4,8 seconds for decompression. Because, PSNR = -10 $\log_{10}$ RMSE.

- So the smaller the error, the higher the PSNR. In particular, the higher the PSNR, the better the quality of the reconstructed data.
- It is this nice fact that justifies the multiplicative factor in the definition of the SNR. The results in table 1 show the high values for PSNR which means a minimum loss of information we decompressing the image.

Using ZJICD algorithm gives a higher compression ratio comparing with other algorithms of image compression-decompression and this is shown in figure 3.

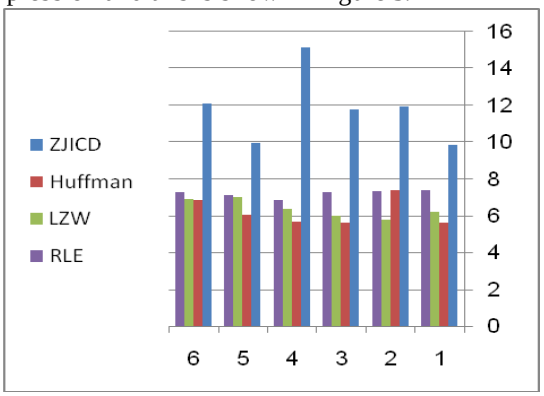

Figure 3. Compression ratio for ZJIDC algorithm and other algorithms.

Conclusion

A novel ZJIDC algorithm for image compression-decompression was proposed and tested. The proposed algorithm requires minimum hardware and it minimizes the information losses and maximizes the compression ratio.

Appendix A

The following structures where used in the code:

symbol_order=[0,3,9,13,17,21,15,27,1,2,4,6,8,10,12,14,5,7,11,23,19,25,16,20,18,22,24,26,28,30,29,33,32,35,34,40,47,50,31,43,51,53,57,45,54,63,39,41,55,60,37,36,38,42,44,46,49,52,48,59,61,56,62,58];

block_order=[441,732,366,695,859,429,726,363,181,602,813,406,203,613,306,665,844,934,467,745,884,954,477,750,375,187,605,302,663,843,421,722,361,180,90,557,278,651,837,418,209,616,820,922,461,742,371,185,604,814,919,971,997,1010,505,764,382,703,863,431,727,875,437,730,365,182,91,45,22,523,261,130,65,32,528,776,900,450,225,624,824,924,462,231,627,313,156,78,551,275,137,580,802,913,968,484,242,121,60,542,783,391,707,865,432,216,108,566,795,397,710,355,177,600,812,918,459,741,882,953,988,494,247,635,317,158,79,39,19,9,4,514,769,384,192,96,560,792,908,454,227,625,312,668,334,679,851,425,724,362,693,858,941,982,491,757,890,957,990,495,759,891,445,734,367,183,603,301,150,75,37,18,521,260,642,321,160,80,552,788,906,453,738,369,184,92,558,791,395,709,866,945,984,492,246,123,61,30,527,263,131,577,288,656,328,676,338,681,852,938,469,746,373,186,93,46,535,267,133,578,801,400,200,100,562,793,396,198,99,49,24,52

Table 1. Algorithm Implementation Results.

Image	Compression time(sec)	Decompression time(sec)	Compression ratio	MSE	PMSE	PNSR (dB)
Baboon	3.292	9.407	6.9444	321.8682	0.006	23.054
Pepper	2.294	4.727	11.7503	58.3044	0.247	30.473
Shannon	1.965	4.103	15.1262	27.977	0.015	33.662
Lena	2.324	5.257	11.9246	35.8221	0.032	32.589
Barbara	2.480	6.646	9.8588	194.4746	0.036	25.471
Camerman	2.122	4.868	12.1036	73,2593	0.024	29.482

The proposed algorithm was compared with other famous algorithms of compression as shown in table 2

Table 2.Comparison Results

Image	RLE	LZW	Huffman	Proposed ZJICD
Barbara	7.359	6.204	5.638	9.8588
Lena	7.344	5.792	7.411	11.9246
Pepper	7.293	6.010	5.638	11.7503
Shannon	6.825	6.349	5.672	15.1262
Baboon	7.120	7.012	6.032	9.9444
Camerman	7.263	6.901	6.854	12.1036

```
4,774,899,961,992,496,248,124,574,799,399,711,867,433,728
,364,694,347,173,598,811,405,714,357,178,89,44,534,779,38
9,706,353,176,88,556,790,907,965,994,497,760,380,702,351,
175,599,299,149,586,805,402,201,612,818,921,972,486,243,6
33,316,670,335,167,595,297,148,74,549,274,649,836,930,465
,744,372,698,349,174,87,43,21,10,517,258,641,832,928,464,2
32,116,570,797,398,199,611,305,152,76,550,787,393,708,354
,689,856,940,470,235,629,314,669,846,935,979,1001,1012,50
6,253,638,831,415,719,871,435,729,876,950,475,749,886,955
,989,1006,503,763,893,446,223,623,311,155,589,294,659,841
,420,210,105,52,538,781,390,195,609,304,664,332,678,339,1
69,596,810,917,970,485,754,377,188,94,559,279,139,581,290
,657,840,932,466,233,628,826,925,974,487,755,889,444,222,
111,55,27,13,6,515,257,128,64,544,784,904,452,226,113,56,5
40,782,903,963,993,1008,504,252,126,575,287,143,583,291,1
45,584,804,914,457,740,370,697,860,942,471,747,885,442,22
1,622,823,411,717,870,947,985,1004,502,251,637,318,671,84
7,423,723,873,436,218,109,54,539,269,134,67,33,16,520,772,
898,449,736,368,696,348,686,343,171,597,298,661,842,933,9
78,489,756,378,701,862,943,983,1003,1013,1018,509,766,38
3,191,1023,607,303,151,587,293,146,73,36,530,777,388,194,
97,48,536,780,902,451,737,880,952,476,238,119,59,29,14,51
9,259,129,576,800,912,456,228,114,569,284,654,327,163,593
,296,660,330,677,850,937,980,490,245,634,829,414,207,615,
307,153,588,806,915,969,996,498,249,636,830,927,975,999,1
011,1017,1020,510,255,639,319,159,591,295,147,585,292,65
8,329,164,82,553,276,650,325,162,81,40,532,778,901,962,48
1,752,376,700,350,687,855,427,725,874,949,986,493,758,379
,189,606,815,407,715,869,434,217,620,822,923,973,998,499,
761,892,958,479,751,887,443,733,878,951,987,1005,1014,50
7,765,894,959,991,1007,1015,1019,1021,1022,511,767,895,4
47,735,879,439,731,877,438,219,621,310,667,845,422,211,61
7,308,666,333,166,83,41,20,522,773,386,193,608,816,920,46
0,230,115,57,28,526,775,387,705,864,944,472,236,118,571,2
85,142,71,35,17,8,516,770,897,960,480,240,120,572,798,911,
967,995,1009,1016,508,254,127,63,31,15,7,3,1,0,512,768,896
,448,224,112,568,796,910,455,739,881,440,220,110,567,283,
141,582,803,401,712,356,690,345,172,86,555,277,138,69,34,
529,264,644,322,673,848,936,468,234,117,58,541,270,647,83
5,417,720,360,692,346,685,854,939,981,1002,501,762,381,19
0,95,47,23,11,5,2,513,256,640,320,672,336,680,340,682,341,
170,85,42,533,266,645,834,929,976,488,244,122,573,286,655
,839,419,721,872,948,474,237,630,827,413,718,359,179,601,
300,662,331,165,594,809,404,202,101,50,537,268,646,323,16
1,592,808,916,458,229,626,825,412,206,103,51,25,12,518,77
1,385,704,352,688,344,684,342,683,853,426,213,618,821,410
,205,614,819,409,716,358,691,857,428,214,107,53,26,525,26
2,643,833,416,208,104,564,794,909,966,483,753,888,956,478
,239,631,315,157,590,807,403,713,868,946,473,748,374,699,
861,430,215,619,309,154,77,38,531,265,132,66,545,272,648,
324,674,337,168,84,554,789,394,197,610,817,408,204,102,56
3,281,140,70,547,273,136,68,546,785,392,196,98,561,280,65
2,326,675,849,424,212,106,565,282,653,838,931,977,1000,50
0,250,125,62,543,271,135,579,289,144,72,548,786,905,964,4
82,241,632,828,926,463,743,883];
```

% % % Zigzag Scanning order.

```
RowArray=[0,0,1,2,1,0,0,1,2,3,4,3,2,1,0,0,1,2,3,4,5,6,5,4,3,2,1
,0,0,1,2,3,4,5,6,7,7,6,5,4,3,2,1,2,3,4,5,6,7,7,6,5,4,3,4,5,6,7,7,6,
5,6,7,7];
ColumnArray=[0,1,0,0,1,2,3,2,1,0,0,1,2,3,4,5,4,3,2,1,0,0,1,2,3
,4,5,6,7,6,5,4,3,2,1,0,1,2,3,4,5,6,7,7,6,5,4,3,2,3,4,5,6,7,7,6,5,4,
5,6,7,7,6,7];
```

% % % Quantization Table.

```
Q=[[16,11,10,16,24,40,51,61];
  [12,12,14,19,26,58,60,55];
  [14,13,16,24,40,57,69,56];
  [14,17,22,29,51,87,80,62];
  [18,22,37,56,68,109,103,77];
  [24,35,55,64,81,104,113,92];
  [49,64,78,87,103,121,120,101];
  [72,92,95,98,112,100,103,99]];
```

% % % AC Coefficients

```
sac(1)=struct('run',0,'cat',0,'accode','1010','aclength',4);          %
0,0,"1010",4,
sac(2)=struct('run',0,'cat',1,'accode','00','aclength',2);            %
0,1,"00",2,
sac(3)=struct('run',0,'cat',2,'accode','01','aclength',2);            %
0,2,"01",2,
sac(4)=struct('run',0,'cat',3,'accode','100','aclength',3);           %
0,3,"100",3,
sac(5)=struct('run',0,'cat',4,'accode','1011','aclength',4);          %
0,4,"1011",4,
sac(6)=struct('run',0,'cat',5,'accode','11010','aclength',5);         %
0,5,"11010",5,
sac(7)=struct('run',0,'cat',6,'accode','1111000','aclength',7);
% 0,6,"1111000",7,
sac(8)=struct('run',0,'cat',7,'accode','11111000','aclength',8);
% 0,7,"11111000",  8,
sac(9)=struct('run',0,'cat',8,'accode','1111110110','aclength',
10);
sac(10)=struct('run',0,'cat',9,'accode','1111111110000010','acle
ngth',16);
sac(11)=struct('run',0,'cat',10,'accode','1111111110000011','acl
ength',16);
sac(12)=struct('run',1,'cat',1,'accode','1100','aclength',4);         %
1,1,"1100",4,
sac(13)=struct('run',1,'cat',2,'accode','11011','aclength',5);        %
1,2,"11011",5,
sac(14)=struct('run',1,'cat',3,'accode','1111001','aclength',7);
% 1,3,"1111001",7,
sac(15)=struct('run',1,'cat',4,'accode','111110110','aclength',9);
% 1,4,"111110110",9,
sac(16)=struct('run',1,'cat',5,'accode','11111110110','aclength',
11);
sac(17)=struct('run',1,'cat',6,'accode','1111111110000100','acle
ngth',16);
sac(18)=struct('run',1,'cat',7,'accode','1111111110000101','acle
ngth',16);
sac(19)=struct('run',1,'cat',8,'accode','1111111110000110','acle
ngth',16);
sac(20)=struct('run',1,'cat',9,'accode','1111111110000111','acle
ngth',16);
sac(21)=struct('run',1,'cat',10,'accode','1111111110001000','acl
ength',16
sac(22)=struct('run',2,'cat',1,'accode','11100','aclength',5);        %
2,1,"11100",5,
sac(23)=struct('run',2,'cat',2,'accode','11111001','aclength',8);
% 2,2,"11111001",8,
sac(24)=struct('run',2,'cat',3,'accode','1111110111','aclength',1
0);
sac(25)=struct('run',2,'cat',4,'accode','111111110100','aclength'
,12);
sac(26)=struct('run',2,'cat',5,'accode','1111111110001001','acle
ngth',16);
sac(27)=struct('run',2,'cat',6,'accode','1111111110001010','acle
ngth',16);
sac(28)=struct('run',2,'cat',7,'accode','1111111110001011','acle
ngth',16);
```

```
sac(29)=struct('run',2,'cat',8,'accode','1111111110001100','aclength',16);
sac(30)=struct('run',2,'cat',9,'accode','1111111110001101','aclength',16);
sac(31)=struct('run',2,'cat',10,'accode','1111111110001110','aclength',16);
sac(32)=struct('run',3,'cat',1,'accode','111010','aclength',6);  % 3,1,"111010",6,
sac(33)=struct('run',3,'cat',2,'accode','111110111','aclength',9);  % 3,2,"111110111",9,
sac(34)=struct('run',3,'cat',3,'accode','111111110101','aclength',12);
sac(35)=struct('run',3,'cat',4,'accode','1111111110001111','aclength',16);
sac(36)=struct('run',3,'cat',5,'accode','1111111110010000','aclength',16);
sac(37)=struct('run',3,'cat',6,'accode','1111111110010001','aclength',16);
sac(38)=struct('run',3,'cat',7,'accode','1111111110010010','aclength',16);
sac(39)=struct('run',3,'cat',8,'accode','1111111110010011','aclength',16);
sac(40)=struct('run',3,'cat',9,'accode','1111111110010100','aclength',16);
sac(41)=struct('run',3,'cat',10,'accode','1111111110010101','aclength',16);
sac(42)=struct('run',4,'cat',1,'accode','111011','aclength',6);  % 4,1,"111011",6,
sac(43)=struct('run',4,'cat',2,'accode','1111111000','aclength',10);
sac(44)=struct('run',4,'cat',3,'accode','1111111110010110','aclength',16);
sac(45)=struct('run',4,'cat',4,'accode','1111111110010111','aclength',16);
sac(46)=struct('run',4,'cat',5,'accode','1111111110011000','aclength',16);
sac(47)=struct('run',4,'cat',6,'accode','1111111110011001','aclength',16);
sac(48)=struct('run',4,'cat',7,'accode','1111111110011010','aclength',16);
sac(49)=struct('run',4,'cat',8,'accode','1111111110011011','aclength',16);
sac(50)=struct('run',4,'cat',9,'accode','1111111110011100','aclength',16);
sac(51)=struct('run',4,'cat',10,'accode','1111111110011101','aclength',16);
sac(52)=struct('run',5,'cat',1,'accode','1111010','aclength',7);  % 5,1,"1111010",7,
sac(53)=struct('run',5,'cat',2,'accode','11111110111','aclength',11);
sac(54)=struct('run',5,'cat',3,'accode','1111111110011110','aclength',16);
sac(55)=struct('run',5,'cat',4,'accode','1111111110011111','aclength',16);
sac(56)=struct('run',5,'cat',5,'accode','1111111110010000','aclength',16);
sac(57)=struct('run',5,'cat',6,'accode','1111111110010001','aclength',16);
sac(58)=struct('run',5,'cat',7,'accode','1111111110010010','aclength',16);
sac(59)=struct('run',5,'cat',8,'accode','1111111110010011','aclength',16);
sac(60)=struct('run',5,'cat',9,'accode','1111111110010100','aclength',16);
sac(61)=struct('run',5,'cat',10,'accode','1111111110010101','aclength',16);
sac(62)=struct('run',6,'cat',1,'accode','1111011','aclength',7);  % 6,1,"1111011",7,
sac(63)=struct('run',6,'cat',2,'accode','111111110110','aclength',12);
sac(64)=struct('run',6,'cat',3,'accode','1111111110100110','aclength',16);
sac(65)=struct('run',6,'cat',4,'accode','1111111110100111','aclength',16);
sac(66)=struct('run',6,'cat',5,'accode','1111111110101000','aclength',16);
sac(67)=struct('run',6,'cat',6,'accode','1111111110101001','aclength',16);
sac(68)=struct('run',6,'cat',7,'accode','1111111110101010','aclength',16);
sac(69)=struct('run',6,'cat',8,'accode','1111111110101011','aclength',16);
sac(70)=struct('run',6,'cat',9,'accode','1111111110101100','aclength',16);
sac(71)=struct('run',6,'cat',10,'accode','1111111110101101','aclength',16);
sac(72)=struct('run',7,'cat',1,'accode','11111010','aclength',8);  % 7,1,"11111010",8,
sac(73)=struct('run',7,'cat',2,'accode','111111110111','aclength',12);
sac(74)=struct('run',7,'cat',3,'accode','1111111110101110','aclength',16);
sac(75)=struct('run',7,'cat',4,'accode','1111111110101111','aclength',16);
sac(76)=struct('run',7,'cat',5,'accode','1111111110110000','aclength',16);
sac(77)=struct('run',7,'cat',6,'accode','1111111110110001','aclength',16);
sac(78)=struct('run',7,'cat',7,'accode','1111111110110010','aclength',16);
sac(79)=struct('run',7,'cat',8,'accode','1111111110110011','aclength',16);
sac(80)=struct('run',7,'cat',9,'accode','1111111110110100','aclength',16);
sac(81)=struct('run',7,'cat',10,'accode','1111111110110101','aclength',16);
sac(82)=struct('run',8,'cat',1,'accode','111111000','aclength',9);  % 8,1,"111111000",9,
sac(83)=struct('run',8,'cat',2,'accode','111111111000000','aclength',15);
sac(84)=struct('run',8,'cat',3,'accode','1111111110110110','aclength',16);
sac(85)=struct('run',8,'cat',4,'accode','1111111110110111','aclength',16);
sac(86)=struct('run',8,'cat',5,'accode','1111111110111000','aclength',16);
sac(87)=struct('run',8,'cat',6,'accode','1111111110111001','aclength',16);
sac(88)=struct('run',8,'cat',7,'accode','1111111110111010','aclength',16);
sac(89)=struct('run',8,'cat',8,'accode','1111111110111011','aclength',16);
sac(90)=struct('run',8,'cat',9,'accode','1111111110111100','aclength',16);
sac(91)=struct('run',8,'cat',10,'accode','1111111110111101','aclength',16);
sac(92)=struct('run',9,'cat',1,'accode','111111001','aclength',9);
```

```
sac(93)=struct('run',9,'cat',2,'accode','1111111110111110','aclength',16);
sac(94)=struct('run',9,'cat',3,'accode','1111111110111111','aclength',16);
sac(95)=struct('run',9,'cat',4,'accode','1111111111000000','aclength',16);
sac(96)=struct('run',9,'cat',5,'accode','1111111111000001','aclength',16);
sac(97)=struct('run',9,'cat',6,'accode','1111111111000010','aclength',16);
sac(98)=struct('run',9,'cat',7,'accode','1111111111000011','aclength',16);
sac(99)=struct('run',9,'cat',8,'accode','1111111111000100','aclength',16);
sac(100)=struct('run',9,'cat',9,'accode','1111111111000101','aclength',16);
sac(101)=struct('run',9,'cat',10,'accode','1111111111000110','aclength',16);
sac(102)=struct('run',10,'cat',1,'accode','111111010','aclength',9);
sac(103)=struct('run',10,'cat',2,'accode','1111111111000111','aclength',16);
sac(104)=struct('run',10,'cat',3,'accode','1111111111001000','aclength',16);
sac(105)=struct('run',10,'cat',4,'accode','1111111111001001','aclength',16);
sac(106)=struct('run',10,'cat',5,'accode','1111111111001010','aclength',16);
sac(107)=struct('run',10,'cat',6,'accode','1111111111001011','aclength',16);
sac(108)=struct('run',10,'cat',7,'accode','1111111111001100','aclength',16);
sac(109)=struct('run',10,'cat',8,'accode','1111111111001101','aclength',16);
sac(110)=struct('run',10,'cat',9,'accode','1111111111001110','aclength',16);
sac(111)=struct('run',10,'cat',10,'accode','1111111111001111','aclength',16);%
sac(112)=struct('run',11,'cat',1,'accode','1111111001','aclength',10);
sac(113)=struct('run',11,'cat',2,'accode','1111111111010000','aclength',16);
sac(114)=struct('run',11,'cat',3,'accode','1111111111010001','aclength',16);
sac(115)=struct('run',11,'cat',4,'accode','1111111111010010','aclength',16);
sac(116)=struct('run',11,'cat',5,'accode','1111111111010011','aclength',16);
sac(117)=struct('run',11,'cat',6,'accode','1111111111010100','aclength',16);
sac(118)=struct('run',11,'cat',7,'accode','1111111111010101','aclength',16);
sac(119)=struct('run',11,'cat',8,'accode','1111111111010110','aclength',16);
sac(120)=struct('run',11,'cat',9,'accode','1111111111010111','aclength',16);
sac(121)=struct('run',11,'cat',10,'accode','1111111111011000','aclength',16);
sac(122)=struct('run',12,'cat',1,'accode','1111111010','aclength',10);
sac(123)=struct('run',12,'cat',2,'accode','1111111111011001','aclength',16);
sac(124)=struct('run',12,'cat',3,'accode','1111111111011010','aclength',16);
sac(125)=struct('run',12,'cat',4,'accode','1111111111011011','aclength',16);
sac(126)=struct('run',12,'cat',5,'accode','1111111111011100','aclength',16);
sac(127)=struct('run',12,'cat',6,'accode','1111111111011101','aclength',16);
sac(128)=struct('run',12,'cat',7,'accode','1111111111011110','aclength',16);
sac(129)=struct('run',12,'cat',8,'accode','1111111111011111','aclength',16);
sac(130)=struct('run',12,'cat',9,'accode','1111111111100000','aclength',16);
sac(131)=struct('run',12,'cat',10,'accode','1111111111100001','aclength',16);
sac(132)=struct('run',13,'cat',1,'accode','11111111000','aclength',11);
sac(133)=struct('run',13,'cat',2,'accode','1111111111100010','aclength',16);
sac(134)=struct('run',13,'cat',3,'accode','1111111111100011','aclength',16);
sac(135)=struct('run',13,'cat',4,'accode','1111111111100100','aclength',16);
sac(136)=struct('run',13,'cat',5,'accode','1111111111100101','aclength',16);
sac(137)=struct('run',13,'cat',6,'accode','1111111111100110','aclength',16);
sac(138)=struct('run',13,'cat',7,'accode','1111111111100111','aclength',16);
sac(139)=struct('run',13,'cat',8,'accode','1111111111101000','aclength',16);
sac(140)=struct('run',13,'cat',9,'accode','1111111111101001','aclength',16);
sac(141)=struct('run',13,'cat',10,'accode','1111111111101010','aclength',16);
sac(142)=struct('run',14,'cat',1,'accode','1111111111101011','aclength',16);
sac(143)=struct('run',14,'cat',2,'accode','1111111111101100','aclength',16);
sac(144)=struct('run',14,'cat',3,'accode','1111111111101101','aclength',16);
sac(145)=struct('run',14,'cat',4,'accode','1111111111101110','aclength',16);
sac(146)=struct('run',14,'cat',5,'accode','1111111111101111','aclength',16);
sac(147)=struct('run',14,'cat',6,'accode','1111111111110000','aclength',16);
sac(148)=struct('run',14,'cat',7,'accode','1111111111110001','aclength',16);
sac(149)=struct('run',14,'cat',8,'accode','1111111111110010','aclength',16);
sac(150)=struct('run',14,'cat',9,'accode','1111111111110011','aclength',16);
sac(151)=struct('run',14,'cat',10,'accode','1111111111110100','aclength',16)
sac(152)=struct('run',15,'cat',1,'accode','1111111111110101','aclength',16);
sac(153)=struct('run',15,'cat',2,'accode','1111111111110110','aclength',16);
sac(154)=struct('run',15,'cat',3,'accode','1111111111110111','aclength',16);
sac(155)=struct('run',15,'cat',4,'accode','1111111111111000','aclength',16);
sac(156)=struct('run',15,'cat',5,'accode','1111111111111001','aclength',16);
```

```
sac(157)=struct('run',15,'cat',6,'accode','1111111111111010','a
clength',16);
sac(158)=struct('run',15,'cat',7,'accode','1111111111111011','a
clength',16);
sac(159)=struct('run',15,'cat',8,'accode','1111111111111100','a
clength',16);
sac(160)=struct('run',15,'cat',9,'accode','1111111111111101','a
clength',16);
sac(161)=struct('run',15,'cat',10,'accode','1111111111111110','
aclength',16);
sac(162)=struct('run',15,'cat',0,'accode','11111111001','aclengt
h',11);
```

% % % DC Coefficients

```
sdc(1)=struct('dccode','00','codelen',2,'dclength',2);          %
"00",2,2,
sdc(2)=struct('dccode','010','codelen',3,'dclength',4);         %
"010",3,4,
sdc(3)=struct('dccode','011','codelen',3,'dclength',5);         %
"011",3,5,
sdc(4)=struct('dccode','100','codelen',3,'dclength',6);         %
"100",3,6,
sdc(5)=struct('dccode','101','codelen',3,'dclength',7);         %
"101",3,7,
sdc(6)=struct('dccode','110','codelen',3,'dclength',8);         %
"110",3,8,
sdc(7)=struct('dccode','1110','codelen',4,'dclength',10);       %
"1110",4,10,
sdc(8)=struct('dccode','11110','codelen',5,'dclength',12);      %
"11110",5,12,
sdc(9)=struct('dccode','111110','codelen',6,'dclength',14);     %
"111110",6,14,
sdc(10)=struct('dccode','1111110','codelen',7,'dclength',16);
% "1111110",7,16,
sdc(11)=struct('dccode','11111110','codelen',8,'dclength',18);
% "11111110",8,18,
sdc(12)=struct('dccode','111111110','codelen',9,'dclength',20);
% "111111110",9,20};
```

References

[1] P. G. Howard and V. J. Scott, “New Method for Lossless Image Compression Using Arithmetic Coding,” *Informa-tion Processing & Management*, Vol. 28, No. 6, 1992, pp. 749-763. doi:10.1016/0306-4573(92)90066-9

[2] P. Franti, “A Fast and Efficient Compression Method for Binary Image,” 1993.

[3] M. Burrows and D. J. Wheeler, “A Block-Sorting Loss-less Data Compression Algorithm,” *Systems Research Center*, Vol. 22, No. 5, 1994, pp.

[4] B. Meyer and P. Tischer, “TMW—A New Method for Lossless Image Compression,” Australia, 1997.

[5] M. F. Talu and İ. Türkoğlu, “Hybrid Lossless Compres-sion Method for Binary Images,” University of Firat, Ela-zig, Turkey, 2003.

[6] L. Zhou, “A New Highly Efficient Algorithm for Lossless Binary Image Compression,” Master Thesis, University of Northern British Columbia, Prince George, 2004.

[7] N. J. Brittain and M. R. El-Sakka, “Grayscale True Two-Dimensional Dictionary-Based Image Compression,” *Jour-nal of Visual Communication and Image Representation*, Vol. 18, No. 1, pp. 35-44.

[8] R.-C. Chen, P.-Y. Pai, Y.-K. Chan and C.-C. Chang, “Lossless Image Compression Based on Multiple-Tables Arithmetic Coding,” *Mathematical Problems in Engi-neering*, Vol. 2009, 2009, Article ID: 128317. doi:10.1155/2009/128317

[8] J. H. Pujar and L. M. Kadlaskar, “A New Lossless Me-thod of Image Compression and Decompression Using Huffman Coding Technique,” *Journal of Theoretical and Applied Information Technology*, Vol. 15, No. 1, 2010.

[10] M. Van Droogenbroeck, R. Benedet. Techniques for a selective encryption of uncompressed and compressed images. *Advances Concepts for Intelligent Vision Systems (ACIVS)*:90-97, 2002. URL http://www.ulg.ac.be/telecom/publi/publications/mvd/acivs2002mvd/index.html.

[11] M. Van Droogenbroeck. Partial encryption of images for real-time applications. *Fourth IEEE Signal Processing Symposium*:11-15, 2004. URL http://www.ulg.ac.be/telecom/publi/publications/mvd/sps-2004/index.html. Invited presentation.