Numerical Analysis

Carmen Arévalo

Lund University

carmen@maths.lth.se

Huffman Coding

Lossless compression

Lossy compression occurs because of quantization of the transform matrix.

There is a tradeoff between the amount of compression and the amount of accuracy. The smaller the resulting file, the bigger the loss of accuracy.

After quantization, we can further compress without losing any more accuracy.

This is done by efficient coding of the transformed image.

Huffman coding

Shannon information

$$\boxed{I = -\sum_{i=1}^{k} p_i \log_2 p_i}$$

Huffman tree

Assign shorter codes to symbols with higher probabilities. From bottom up, join symbols with smallest probabilities. Assign a 0 to left branches, 1 to right branches.

Huffman coding for JPEG

It needs:

1. a Huffman tree for the DC components
2. a Huffman tree for the AC components
3. an integer identifier table

Huffman coding for the DC component

The code for y_{00} (DC component) has two parts, the first is obtained from the DPCM (Differential Pulse Code Modulator) tree, and the second part from the integer identifying table. The DC coefficient is a binary formed by the concatenation of these two parts.

Example: $y_{00} = 18$. From the table we see that $L = 5$. From the DPCM tree, we get 110. From the table, the extra digits are 10010. The entire code is 11010010.

Huffman coding for the AC components

For an 8×8 matrix, the remaining 63 components are stored in a zigzag fashion:

$$\left(\begin{array}{cccccccc} & 1 & 5 & 6 & \longrightarrow & & & \\ 2 & 4 & 7 & \nearrow & \swarrow & & & \\ 3 & 8 & \nearrow & \swarrow & & & & \\ 9 & \nearrow & \swarrow & & & & & \\ 10 & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{array} \right)$$

AC components are coded in a run-length pair (n,L), where n is the length of a run of zeros and L is the length of the next nonzero entry. Then a Huffman AC tree is used to code these pairs. After that comes the integer identifying code.

Example for AC components

Coding of

$$\begin{pmatrix} * & 3 & 0 & 1 & 0 \\ 0 & -2 & 8 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(0,2)3(2,2)-2(1,1)1(0,4)8(4,1)1EOB
(01)(11)(11111001)(01)(1100)(1)(1011)(1000)(11111)(1)(1010)

Note that we need 34 bits for the 24 pixels:37/24=1.54 bits/pixel. At 8 bits per pixel in the original matrix, the compression is greater than 5:1