

# Using Multiple Huffman Code Tables for Optimal Coding of DCT Blocks

Gopal Lakhani, Texas Tech University  
(lakhani@cs.ttu.edu)

It is a well-known observation that when a DCT block is traversed in zigzag order, the AC coefficients generally decrease in size and the run-length of zero coefficients increase in number. Therefore, use of a single AC Huffman code table in the JPEG baseline algorithm leads to sub-optimal coding, and it is desirable to use multiple code tables, one for each DCT coefficient position, if necessary. It creates a problem, because a non-zero coefficient,  $X$ , and the run-length,  $Z$ , of zero coefficients that precede  $X$ , are coded as one element  $(Z,X)$ , and therefore, the decoder may not know which table to use to decode the next  $X$ . To solve this problem, we made a minor modification to the JPEG Huffman coding algorithm. During the run-length coding phase, instead of pairing  $X$  with the run-length of zero coefficients that precede it, our encoder pairs  $X$  with the run of zeros that follow  $X$ . This change requires our codec to handle two possible zero runs for each block, one that precede the first non-zero coefficient, and other that follow the last non-zero coefficient differently. To accommodate these changes, our code tables contain a small number of additional entries, but the table indices,  $0 \leq Z \leq 15$  and  $0 \leq X \leq 10$  of our code tables are same as for the JPEG AC code Table K.5. A side advantage of this change is that no end-of-block marker is needed for most DCT blocks. Our encoder includes 63 AC code tables with the image code, and therefore, we developed an efficient algorithm for coding the AC code tables. It is based on following observations. (1) It is sufficient to specify the code string size, instead of actual code strings. (2) The code string size of any pair  $(Z,X)$ , in the tables for any two consecutive AC positions in the zigzag order is nearly the same, and hence it can be coded differentially. (3) Most of non-trivial size entries are for smaller values of  $Z$  and  $X$  (other entries are 16). For each AC code table, our algorithm encodes for each  $X$ ,  $X = 0, \dots, 10$ , the number of changes in the code size set  $\{(Z, X), Z = 0, \dots, 15\}$  w.r.t. to the previous table and then records the code size of each  $(Z,X)$  differentially. A separate Huffman code table is used to encode code size.

To evaluate reduction in the code size using our method, we compressed the luminance component of ten well-known test images at default quality level and computed AC Huffman code size. Experimental results show that if images are coded using a single AC custom code table, the average reduction over JPEG Table K.5 is only 1.38% (i.e., Table K.5 is indeed very efficient), where as the reduction using our 63 AC code tables over Table K.5 is 14.6%, i.e., use of multiple code tables produces significant reduction. Further, the average code size of our 63 AC code tables is 6606 bits, where as the same for Table K.5 alone is 1432. If we include the Table code size in this comparison, the average reduction by our method is 12.1%. It is emphasized that our method does not add any complexity to the DCT coefficient decoding process. The extra burden on our decoder is that it must decode 63 AC code tables and one table used for coding the AC code tables. This is a one time cost.