

# Compressed Pattern Matching in JPEG Images

Shmuel T. Klein  
Bar Ilan University, Israel  
tomi@cs.biu.ac.il

Dana Shapira  
Ashkelon Academic College, Israel  
shapird@ash-college.ac.il

We concentrate here on two-dimensional compressed matching in which the given encoded text  $\mathcal{E}(T)$  is an  $n \times k$  pixel image encoded by the standard JPEG baseline scheme and the pattern consists of a given  $m \times \ell$  pixel image fragment we are looking for. Baseline JPEG uses a static Huffman code, without which compressed matching would not always be possible, since our underlying assumption is that all occurrences of the pattern are encoded by the same binary sequence. This is not the case for dynamic Huffman coding or for arithmetic coding.

The compressed matching starts by encoding the pattern using the same JPEG algorithm as the one used for the original image. Even then we cannot assure that a pattern can be located, as the  $8 \times 8$  blocks of the pattern are not necessarily aligned with those of the image. The search process has therefore to be repeated 64 times, positioning, for each matching attempt, the leftmost uppermost pixel of the first  $8 \times 8$  block in the pattern at the  $i$ th pixel in the  $j$ th row,  $1 \leq i, j \leq 8$ .

The compressed matching paradigm raises then several problems. First, suppose that the binary sequence encoding the first part of the pattern is indeed located. This does not necessarily mean that an occurrence of the encoded elements is found, as the beginning of the Huffman codewords might not be synchronized. The second problem is that the encoded pattern does not appear consecutively in the encoded image (unless  $k = \ell$ ), but with gaps corresponding to the encoding of  $(k - \ell)/8$  blocks. The third problem relates to the fact that there are possibly many occurrences of the pattern, perhaps even overlapping ones. We therefore conclude that compressed matching in JPEG files is hard to achieve, unless we keep some auxiliary data.

The task would be much easier if one would know, for a given position in the JPEG encoded file, the index of the corresponding  $8 \times 8$  block in the original file. One could construct a table  $S$ , acting as an index, that would be stored in addition to the original compressed file.  $S(i)$  would be the bit-position, within the JPEG image, of the beginning of the encoding of the AC sequence in the  $i$ th block, that is the index of the bit following the DC value. The size of each entry in  $S$  would be  $\lceil \log |\mathcal{E}(T)| \rceil$ , so that a 3 byte entry could accommodate a compressed image of size up to 2MB.

The construction of such an index has to be done in a preprocessing stage, which can be justified in certain applications, for example when one large image will be used many times for searches with different patterns. The index  $S$  can be used to solve some of the problems mentioned above. Once the encoding of the first row of the pattern image has been located in  $\mathcal{E}(T)$  at bit offset  $y$ , a binary search for  $y$  in  $S$  can decide in  $\lceil \log n + \log k \rceil - 6$  comparisons whether the match is a true one. Similar searches for the following rows of the pattern can locate all the rows, without decoding.