

STATISTICAL IMAGE COMPRESSION

EUGENE AGEENKO

The aim of compression is to remove redundancy of the data. Statistical image compression consists of two distinct phases: *statistical modeling* and *coding* [RL81]. In the modeling phase, we construct the probability distribution for the occurrence of the symbols to be compressed. The coding process assigns the variable length code words to the symbols according to the probability model, so that shorter codes are assigned to symbols that are more probable and vice versa. The main problem of the compression is to find a good model, describing the data with high precision. The coding can be efficiently performed using *arithmetic coding*, which is an optimal coding for a given probability model [RL79].

1. Statistical Modeling

A binary image can be considered as a message, generated by an information source. The idea of statistical modeling is to describe the message symbols (pixels) according to the probability distribution of the source alphabet (binary alphabet, in our case). Shannon has shown in [Sh48] that the information content of a single symbol (pixel) in the message (image) can be measured by its *entropy*:

$$H_{pixel} = -\log_2 p, \quad (1)$$

where p is the probability of the pixel. Entropy of the entire image can be calculated as the average entropy of all pixels:

$$H_{image} = -\frac{1}{n} \sum_{i=1}^n \log_2 p_i, \quad (2)$$

where p_i is the probability of i -th pixel and n is the total number of pixels in the image. If the probability distribution of the source alphabet (black and white pixels) is a priori known, the entropy of the probability model can thus be expressed as:

$$H = -p_W \log_2 p_W - p_B \log_2 p_B, \quad (3)$$

where p_W and p_B are the probabilities of the white and black pixels, respectively.

Entropy gives the optimal number of bits required for encoding a single pixel with a given model. A model with skewed probability distribution will have low entropy, see Figure 1. Respectively, the codes with the lengths equal to the entropy values will provide an optimal compression in respect of the model.

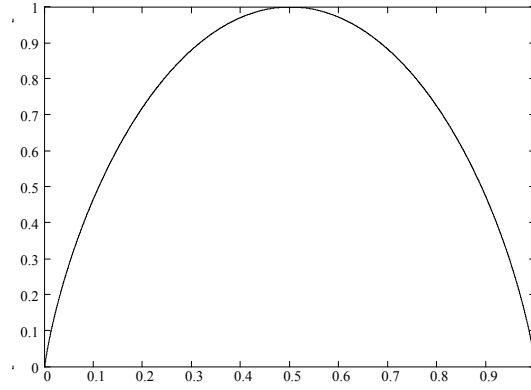


Figure 1. Entropy of the binary probability model, as function of white pixel probability.

2. Static, Semi-adaptive and Adaptive Approaches

The modeling schemes can be classified as *static*, *semi-adaptive* or *adaptive (dynamic)*. In the static modeling, the probability distribution of a source alphabet is a priori known (or suggested) and the same, non-changing model is applied to every pixel during the compression. The advantage of static modeling is its simplicity, and that no side information has to be passed to the decoder.

Semi-adaptive modeling uses a preliminary pass on the input data to gather statistics and construct the model. The model is passed to the encoder, which performs data compression as in the static variant. The model must also be passed to the decoder to make decompression possible.

Dynamic (adaptive) modeling takes one step further and eliminates the need for an extra pass over the image to construct the model, and no model overhead is required. Both encoder and decoder dynamically estimate the model during the compression/decompression adapting to the preceding data. Usually an equal initial probability distribution for black and white pixels is assumed. The time-dependent cumulative pixel counts n_B^t and n_W^t are therefore initialized to 1, and are subsequently incremented by 1 each time black or white pixel value appears, respectively.

The more sophisticated *Bayesian sequential estimator* calculates probability of the pixel on the basis of the observed pixel frequencies as follows [MF98]:

$$p(x^t) = \begin{cases} p_W^t = \frac{n_W^t + \delta}{n_W^t + n_B^t + 2\delta}, & \text{if } x^t \text{ is white} \\ p_B^t = 1 - p_W^t, & \text{if } x^t \text{ is black} \end{cases} \quad (4)$$

where n_W^t , n_B^t are the time-dependent counters, p_W^t , p_B^t are the probabilities for white and black colors respectively, and δ is a constant. Counters n_W^t and n_B^t start from zero and are updated after the pixel has been coded (decoded). As in [JBIG1], we use $\delta = 0.45$. The

cumulative equation for entropy (2.2) is used to estimate the average bit rate and calculate the ideal code length.

Dynamic modeling is inefficient at early stage of compression, since it takes time to adapt to the correct model, but highly applicable for compression large volumes of data, such as document images. The loss in compression rate caused by the model adaptation is known as the *learning cost*.

3. Context Modeling

The pixels in an image form geometrical structures with appropriate spatial dependencies. The dependencies can be localized to a limited neighborhood, and described by a *context-based statistical model* [LR81]. In this model, the pixel probability is conditioned on the *context* C , which is defined as distinct black-white configuration of neighboring pixels within the local template. For binary images, the pixel probability is calculated by counting the number of black (n_B^C) and white (n_W^C) pixels appeared in that context in the entire image:

$$p(x) = \begin{cases} p_W^C = \frac{n_W^C}{n_W^C + n_B^C}, & \text{if } x = \text{white} \\ p_B^C = 1 - p_W^C, & \text{if } x = \text{black} \end{cases}, \quad (5)$$

Here, p_B^C and p_W^C are the corresponding probabilities of the black and white pixels. The entropy $H(C)$ of a context C is defined as the average entropy of all pixels within the context:

$$H(C) = -p_W^C \cdot \log_2 p_W^C - p_B^C \cdot \log_2 p_B^C \quad (6)$$

A context with skew probability distribution has smaller entropy and therefore smaller information content. The entropy of an N -level context model is the weighted sum of the entropies of individual contexts:

$$H_N = -\sum_{j=1}^N p(C_j) \cdot (p_W^{C_j} \cdot \log_2 p_W^{C_j} + p_B^{C_j} \cdot \log_2 p_B^{C_j}). \quad (7)$$

In principle, a skewed distribution can be obtained through conditioning of larger regions by using larger context templates. However, this implies a larger number of parameters of the statistical model and, in this way, increases the model cost, which could offset the entropy savings. Another consequence is the “context dilution” problem occurring when the count statistics are distributed over too many contexts, thus affecting the accuracy of the probability estimates.

4. Arithmetic Coding

Arithmetic coding is a statistical compression method that assigns one long code to the entire input stream, instead of assigning codes to the individual symbols [RL79, WNC87]. It is an optimal coding method for a given probability model, because it can achieve a bit-rate approximately equal to the entropy value.

The basic idea of arithmetic coding is to represent the entire input data as a small sub-interval in range $[0,1)$. The coding process starts by dividing the interval $[0,1)$ into two sub-intervals according to the probability distribution of the black and white pixels. Depending on the pixel color, the upper or lower sub-interval is chosen, and the process is repeated for the next symbols, resulting in smaller and smaller intervals. The final interval describes the source uniquely. The length of this interval, L , is the cumulative product of the probabilities of the coded symbols:

$$L_{\text{final}} = p_1 \cdot p_2 \cdot p_3 \cdot \dots \cdot p_n = \prod_{i=1}^n p_i, \quad (8)$$

and it can be coded by the following number of bits:

$$\text{Codesize}(L_{\text{final}}) = -\log_2 \prod_{i=1}^n p_i = -\sum_{i=1}^n \log_2 p_i \quad (9)$$

The implementation aspects of the binary arithmetic coding follow [Sal97]. The encoding process starts by defining two variables, *Low* and *High*, in order to describe the coding interval. The *Low* is initialized to 0, and *High* to an infinite fraction $.999\dots$, since it has to be interpreted as a fraction less than 1. Usually, *Low* and *High* are represented as integer (binary) variables holding the most significant part of the real numbers. After the pixel has been coded, the interval is reduced by a factor that equals the pixel probability, and *Low* and *High* are updated accordingly. Very soon the interval becomes too small to be expressed by the two variables, and the interval scaling procedure is therefore applied. When the interval falls below or above the *half point*, the codeword is known to start with the bit 0 or 1, respectively. In both cases, the starting bit can be shifted out of the interval variables and output to the compressed stream, and the interval is rescaled, see Figure 2.

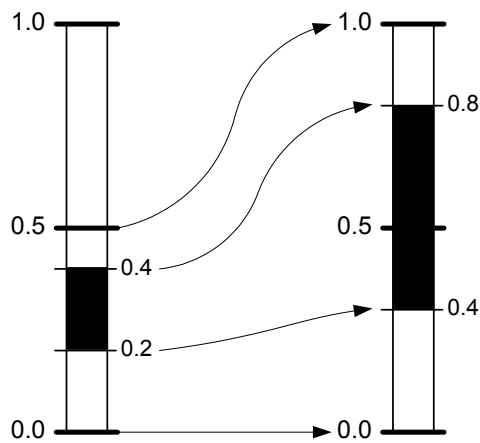


Figure 2. Example of *half point scaling* [RL79].

Underflow can occur when the size of the interval becomes too small, but the interval still covers the half point. To solve this problem, *quarter point scaling* is applied, see Figure 3. In this case, neither bit is output. Later, when the half point scaling occurs, an appropriate bit will be added to the code stream, see [RL79] for details.

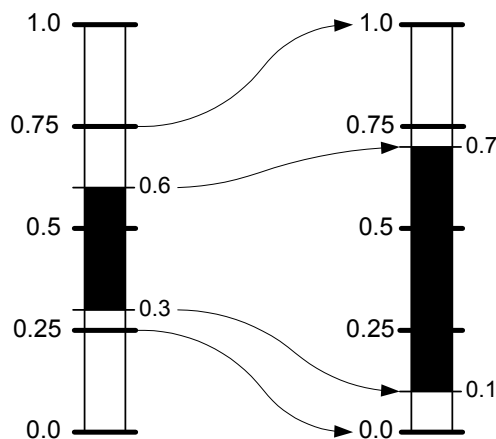


Figure 3. Example of *quarter point scaling* [RL79].

5. QM-coder

The QM-coder is the arithmetic coder used in JBIG1 [PMLA88, PM93]. It is an approximate implementation of arithmetic coding tailored for binary data. Its sub-optimality is compensated by the sophisticated automaton-based probability estimation (see Section 6), providing fast adaptation to the source data.

The QM-coder uses the following variables to describe the interval: *interval base* and *interval size*. If the encoded pixel value (color) is the one with higher probability, it is denoted as *most probable symbol* (MPS), when the opposite value is denoted as the *least probable symbol* (LPS). The interval is always divided so that the LPS sub-interval is above and MPS sub-interval is below as shown in Figure 4. Here C is the *interval base*, A is the *interval size*, and Q_e is the LPS probability estimate [PMLA88].

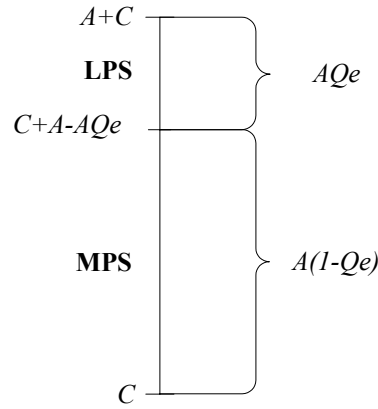


Figure 4. Interval subdivision of the QM-coder.

Altering the interval size involves multiplication. The QM-coder accepts the element of approximation by replacing the interval multiplication by suitable scaling. It assumes that the interval size is roughly constant and equals to 1. In this case, the coding of a pixel changes the interval as follows.

After MPS:

$$\begin{aligned}
 &C \text{ is unchanged} \\
 &A \leftarrow A \cdot (1 - Qe) = A - A \cdot Qe \approx A - Qe
 \end{aligned} \tag{10}$$

After LPS:

$$\begin{aligned}
 &C \leftarrow C + A \cdot (1 - Qe) = C + A - A \cdot Qe \approx C + A - Qe \\
 &A \leftarrow A \cdot Qe \approx Qe
 \end{aligned} \tag{11}$$

The interval size is maintained between 0.75 and 1.5, centered on 1. When the interval size falls below lower bound, the interval is renormalized by a series of consecutive duplications performed by bit-shifting operations. The renormalization occurs always after the LPS, and if necessary, after the MPS is encountered. At each renormalization, the encoder generates output bits (0 or 1) regarding to MPS or LPS and number of duplications in the renormalization process.

6. Automaton-based Probability Estimation

Probability estimation can be derived from arithmetic coder renormalization, as in the QM-coder [PM88]. Instead of maintaining pixel counts, the estimation process is implemented as a state automaton consisting of 226 states. Each context has its own 8-bit pointer to the automaton, where one bit indicates the color of MPS. The automaton has mirror symmetry about the change in the sense of MPS color, and we therefore consider only 113 states, see Figure 5. The automaton is a Markov-chain containing one state for each probability estimate. The states are organized in rows that are ordered by the level of adaptation. The states in the upper rows are more sparsely distributed throughout the probability range and therefore they allow faster adaptation.

The adaptation process starts from the zero-state. In each state, the automaton can perform a transition to two other states, see Figure 5. After each MPS renormalization, a transition is made to the next state situated to the right in the same row, having a smaller LPS probability. After each LPS renormalization, a transition is made to the state with a larger LPS probability, which is the appropriate state in the row at the next level in the case of the *transient* state, or to the preceding state in the same row in the case of *non-transient* states. *Transient* states are, therefore, visited only during the learning stage, and the pointers stabilize eventually to the *non-transient* states. If the statistics change later, the non-transient states can be re-entered from other non-transient states, making local adaptation possible.

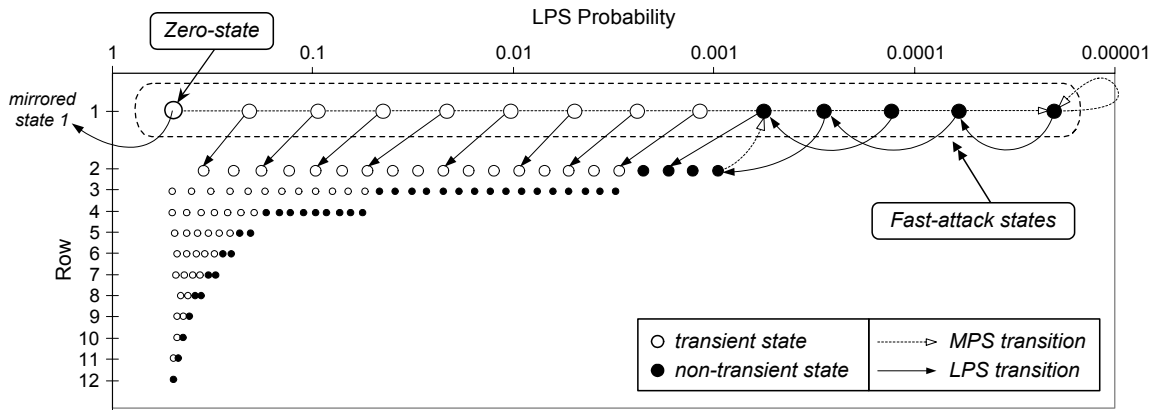


Figure 5. Spatial organization of the QM-coder state automaton and transition sketch for the fast-attack states. Because of the mirror symmetry regarding the change in sense of MPS, only half of the states are depicted.