

# EMBEDDED BLOCK CODING IN JPEG2000

*David Taubman*

The University of NSW, Sydney

*Erik Ordentlich  
Marcelo Weinberger  
Gadiel Seroussi*

HP Labs, Palo Alto

*Ikuro Ueno  
Fumitaka Ono*

Mitsubishi Electric Corp., Tokyo

## ABSTRACT

This paper describes the embedded block coding algorithm at the heart of the JPEG2000 image compression standard. The algorithm achieves excellent compression performance, usually somewhat higher than that of SPIHT with arithmetic coding, but in some cases substantially higher. The algorithm utilizes the same low complexity binary arithmetic coding engine as JBIG2. Together with careful design of the bit-plane coding primitives, this enables comparable execution speed to that observed with the simpler variant of SPIHT without arithmetic coding. The coder offers additional advantages including memory locality, spatial random access and ease of geometric manipulation.

## 1. INTRODUCTION

The purpose of this paper is to describe the embedded block coding algorithm which is central to the JPEG2000 image compression standard, being developed under the auspices of ISO/IEC JTC 1/SC29 WG1 (commonly known as the JPEG working group). The block coding concept and the embedded coder itself draw heavily from the EBCOT algorithm [1], which itself builds upon the contributions of other works; however, there are some notable differences as well as a number of mode variations which can have significant practical implications. Our goal here is to provide the reader with an appreciation for the salient features of the algorithm, as well as some of the considerations which have contributed to its development.

## 2. ROLE OF THE BLOCK CODER

JPEG2000 involves three key steps: a Discrete Wavelet Transform (DWT) first decomposes the image pixels into spatial frequency subbands; each subband is partitioned into relatively small blocks<sup>1</sup> known as code-blocks, which are independently coded into embedded block bit-streams; finally, the embedded block bit-streams are packed into “quality layers” in the codestream.

Ideally, an embedded coder generates a bit-stream such that every prefix of the stream can be decoded to reconstruct the original code-block with a fidelity approaching that of an “optimal” coder, tailored to produce the same bit-rate as the prefix. The bit-streams generated by the JPEG2000 block coder may be truncated at any of a large number of points. The compressor is free to assign incremental contributions from each code-block to each quality layer in any desired fashion. One approach is to collect rate and distortion information,  $(R_i^n, D_i^n)$ , for each candidate truncation point,  $n$ , in the embedded bit-stream generated for code-block  $B_i$ , using this information to assign code-block contributions to quality layers in such a way that layers  $1, 2, \dots, \lambda$  contain a rate-distortion optimal representation of the image, for each  $\lambda$ . This strategy may

be applied to various distortion measures, as described in [1]. The JPEG2000 standard, however, places no restriction on the interpretation adopted by the compressor in its construction of quality layers.

We stress the fact that each code-block is coded entirely independently, without reference to other blocks in the same or other subbands, which is contrary to the approach adopted by the well-known zero-tree schemes [2, 3]. The adoption of independent embedded block coding for JPEG2000 has significant benefits. The blocks may be coded or decoded in parallel and in any order. This, in turn, permits a degree of spatial random access to the image content, and facilitates the efficient implementation of geometric manipulations such as rotation. The approach is appealing for applications requiring a degree of error resilience, since errors are confined spatially while the embedding lends itself to differential protection strategies. Most notably, independent embedded block coding allows users of the JPEG2000 standard to arbitrarily select the contributions made by each code-block to each quality layer and this permits improvements in rate-distortion performance which more than compensate for the cost of restarting the coding process at each block.

## 3. EMBEDDED BLOCK CODING

First, we discuss bit-plane coding, a natural technique for embedded coding. Let  $\chi[\mathbf{n}] \in \{1, -1\}$  denote the sign of the sample (transform coefficient)  $s[\mathbf{n}]$  at location  $\mathbf{n} = [n_1, n_2]$ , and let  $\nu[\mathbf{n}]$  denote the quantized magnitude, i.e.

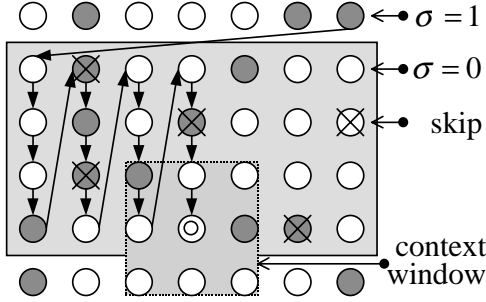
$$\nu[\mathbf{n}] = \left\lfloor \frac{|s[\mathbf{n}]|}{\delta} \right\rfloor$$

where  $\delta$  is the step-size of the relevant “deadzone” quantizer. The JPEG2000 standard supports separate quantization step-sizes for each subband; however, the rate-distortion properties of the embedded representation are usually insensitive to the choice of  $\delta$ .

The value of any individual sample,  $s[\mathbf{k}]$ , is represented by progressively coding its magnitude bits, from most to least significant. Let  $P$  denote the number of bit-planes required to represent all samples in the block, so that  $\nu[\mathbf{n}] < 2^P, \forall \mathbf{n}$ ; then the embedded bit-stream identifies the value of the  $p$ 'th magnitude bit,  $\nu_p[\mathbf{k}]$ , for each  $p = P - 1, \dots, 0$ . The sign bit,  $\chi[\mathbf{k}]$ , is coded immediately after the first “significant” bit position,  $\nu_p[\mathbf{k}] = 1$ . As a result, truncation of the embedded representation is equivalent to discarding some number,  $p_k$ , of least significant magnitude bits from each sample,  $s[\mathbf{k}]$ ; this, in turn, is equivalent to employing a larger quantization step size,  $2^{p_k}\delta$ , for that sample.

Thus, for any given sample, the available quantization levels are quite coarsely spaced. However, an important feature of the embedded coder in JPEG2000, over plain bit-plane coding, is the adaptive nature of the sequence in which bits from different samples are coded, which tends to encode the most valuable informa-

<sup>1</sup>The maximum block size is 4096 samples.



**Fig. 1.** Stripe-based coding scan in significance propagation pass. Significance state indicated by shading. Samples skipped in pass indicated by cross.

tion (in the sense of reducing the distortion of the reconstructed image the most) as early as possible. The embedded block coder uses context modeling to address both the ordering and the coding of the events. Section 3.1 describes the bit-plane coding primitives, while Section 3.2 describes the adaptive coding sequence.

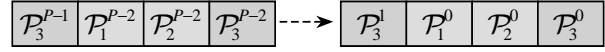
### 3.1. Bit-Plane Coding Primitives

We briefly describe the primitive coding operations which form the foundation of the embedded block coding strategy. The primitives are used to code new information for sample  $s[\mathbf{k}]$ , at the point prescribed by the coding sequence described in Section 3.2. The new information corresponds to some bit-plane  $p$ , all more significant magnitude bits of the sample having already been coded. We define a binary state variable,  $\sigma[\mathbf{k}]$ , which identifies whether or not a non-zero magnitude bit has already been coded for  $s[\mathbf{k}]$ , i.e.  $\nu[\mathbf{k}] \geq 2^{p+1}$ . If the sample is not yet significant ( $\sigma[\mathbf{k}] = 0$ ), a ‘‘Significance Coding’’ primitive is used to code whether or not the symbol becomes significant in bit-plane  $p$ , i.e. whether or not  $\nu_p[\mathbf{k}] = 1$ . If so, the significance state is flipped to  $\sigma[\mathbf{k}] = 1$  and the ‘‘Sign Coding’’ primitive must also be invoked to identify the sign,  $\chi[\mathbf{k}]$ . If the sample is already significant, the ‘‘Refinement’’ primitive is used to encode the value of  $\nu_p[\mathbf{k}]$ .

In each case, a single binary-valued symbol is encoded using the adaptive arithmetic coder discussed in Section 4. The probability models used by the coder evolve within 18 different contexts: 10 for significance coding; 5 for sign coding; and 3 for refinement coding. The specific context to be used is determined from the current significance state and sign of the 8 neighbouring samples, as illustrated in Figure 1. Notice that the causality relations that are relied upon to determine the context model for coding are affected by the adaptive ordering induced by the coding passes. The stripe-based coding scan shown in Figure 1 is discussed in Section 3.2; however, we note here that when all four samples in a stripe column are candidates for significance coding in bit-plane  $p$ , and all eight neighbours of each of the four samples are currently insignificant, a single ‘‘run-length coding’’ primitive is used to efficiently code whether or not any sample in the column becomes significant in the new bit-plane. For more information on the coding primitives, the reader is referred to [1].

### 3.2. Fractional Bit-Plane Scan

To improve the embedding, JPEG2000 employs an adaptive ordering as part of the block coding process. The concept of adaptively ordering the compressed representation through context modeling was introduced independently and in somewhat different forms in [4] and [5]. It is also closely related to the coding sequence employed in SPIHT [3] and, to a lesser extent, EZW [2]. Rather than



**Fig. 2.** Organization of embedded bit-stream.

pursuing a totally adaptive approach, as in [5], JPEG2000 imposes reasonable assumptions on the data, defining context-dependent coding passes accordingly, in the spirit of [4]. For each bit-plane,  $p$ , we identify three ‘‘fractional bit-plane’’ passes:  $\mathcal{P}_1^p$ ,  $\mathcal{P}_2^p$  and  $\mathcal{P}_3^p$ . Each pass involves a scan through the code-block samples in stripes of height 4, as shown in Figure 1. Information for bit-plane  $p$  is coded for each sample in only one of the passes; that sample is skipped in the other two passes. Fractional bit-planes are treated as indivisible units, and thus determine the candidate truncation points for the codestream. The passes are as follows:

*Significance propagation,  $\mathcal{P}_1^p$* : Includes all samples which are currently insignificant, but have at least one immediate neighbour that has been found to be significant. Clearly, these samples are most likely to become significant. In fact, for a broad class of probability models these samples will yield the largest expected reduction in distortion relative to the expected increase in code length [4]. Note that membership in this coding pass is determined as the coding proceeds, following [1]. When a sample becomes significant, its insignificant neighbours become candidates for the remainder of the coding pass, which can induce a propagating wave of significance decisions along significant image features such as edges and texture regions.

*Magnitude refinement,  $\mathcal{P}_2^p$* : Includes samples which are already significant.

*Cleanup,  $\mathcal{P}_3^p$* : Includes all samples skipped in  $\mathcal{P}_1^p$  and  $\mathcal{P}_2^p$ . Note that these may or may not have significant neighbours.

Figure 2 illustrates the contribution of each coding pass to the embedded block bit-stream. The number of bit-planes,  $P$ , required to represent the code-block, is coded separately, as described in [1].

A key feature of JPEG2000 is its highly regular stripe-based scan, as shown in Figure 1. The stripe height of 4 is carefully selected to facilitate efficient hardware and software implementations. In dedicated hardware, a small local memory is required to keep track of the significance,  $\sigma[\mathbf{n}]$ , the sign,  $\chi[\mathbf{n}]$ , the current magnitude bit,  $\nu_p[\mathbf{n}]$ , and a binary state variable,  $\pi[\mathbf{n}]$ , identifying whether or not the sample has been coded in a previous pass of the same bit-plane. These quantities may be staged through a small shift register containing the information required to encode or decode a single stripe column: a  $6 \times 3$  array of  $\sigma$  bits; a  $6 \times 3$  array of  $\chi$  bits; a  $1 \times 4$  array of  $\nu_p$  bits; and a  $1 \times 4$  array of  $\pi$  bits. The refinement primitive also requires information concerning whether or not each sample has been previously refined (see [1]), but this binary state variable,  $r[\mathbf{n}]$ , can share the  $\chi$  shift register bits, which are not required in the magnitude pass. All up, local on-chip memory could be as small as  $5 \times 4096 \text{ bits}^2$ , with a 44 bit shift register providing access to the current stripe column’s coding state. Following the procedure described in Section 4, it is often possible to process all four samples in the stripe column together in a single ‘‘clock’’.

The algorithm has also been carefully designed to facilitate efficient implementation in software, as suggested by the timing figures presented in Table 2. Section 5 discusses further options for reducing complexity and increasing parallelism.

<sup>2</sup>We assume here that the magnitude bits,  $\nu_p[\mathbf{n}]$ , are loaded (flushed for decoding) as each new bit-plane is processed.

#### 4. THE MQ CODER

The binary-valued symbols produced by the bit-plane coding primitives discussed in Section 3.1 are coded using the MQ coder. The MQ coder is an adaptive, binary arithmetic coder, characterized by multiplier-free approximation, renormalization-driven probability estimation and bit-stuffing, all as in the Q-coder [6], and enhanced by a conditional exchange procedure derived from the MELCODE [7] and the state-transition table known as JPEG-FA [8]<sup>3</sup>.

At any given point in the general arithmetic coding process, the string of symbols which have been seen so far is mapped to a unique sub-interval,  $[c, c + a) \subseteq [0, 1)$ , represented by

$$c = C \cdot 2^{-16-N} \text{ and } a = A \cdot 2^{-16-N} \quad (1)$$

where  $C$  and  $A$  are integers and  $N$  is the total number of normalization shifts which have been employed to ensure that  $2^{15} \leq A < 2^{16}$ .<sup>4</sup> Upon completion, the compressed bit-stream represents  $C$  and its length can be  $N + 2$  or fewer bits, depending upon encoder and decoder termination rules. This representation is common to virtually all realizations of the arithmetic coding principle.

Suppose the next symbol occurs in context  $\kappa$ , for which the Least Probable Symbol (LPS) is currently estimated to occur with probability  $p_\kappa \in (0, 1/2)$ . Ideally, the interval length shrinks according to  $a \leftarrow ap_\kappa$  if the next symbol is the LPS and  $a \leftarrow a - ap_\kappa$  if it is the MPS. These are approximated by  $A \leftarrow \overline{p_\kappa}$  and  $A \leftarrow A - \overline{p_\kappa}$ , respectively, where  $\overline{p_\kappa}$  is an integer approximation to  $p_\kappa \alpha 2^{16}$  and  $\alpha \approx 0.71$  is an empirically observed mean of  $2^{-16}A$ . The LPS is mapped to the lower sub-interval so that  $C$  is unchanged; when an MPS occurs, we map  $C \leftarrow C + \overline{p_\kappa}$ . If this process leaves  $A < 2^{15}$ , a renormalization operation is applied to restore  $A$  to its legal range. During renormalization,  $A$  and  $C$  are repeatedly doubled and  $N$  incremented accordingly, until  $A$  is restored to the range  $2^{15} \leq A < 2^{16}$ . For LPS probabilities close to  $\frac{1}{2}$ , the procedure is modified by the conditional-exchange mechanism to improve the approximation of interval length.

##### 4.1. Carry Propagation and Bit-Stuffing

The normalization policy above ensures that  $A$  may be represented with 16 bits. At any point in the coding process, however,  $C$  has a  $16 + N$  bit representation. All arithmetic operations take place in the 16 LSB's of  $C$ , but we are prevented from immediately dispatching the more significant bits to an output buffer by the well-known carry propagation problem.

The MQ-coder incorporates a bit-stuffing mechanism, which allows code bytes to be dispatched with lower implementation cost than the QM-coder, which stacks consistent FF bytes until carries can be resolved. Assembled bytes are dispatched through a single byte buffer; whenever this buffer assumes the value FF, an extra bit is inserted into the representation of  $C$  so as to ensure that the effect of future coding steps may not propagate beyond the single byte stored in this buffer. Bit-stuffing adds approximately 0.05% to the code length, but has the desirable effect of bounding the number of operations which must be performed during renormalization.

The specific realization of bit-stuffing in the MQ-coder has the property that any consecutive pair of bytes dispatched to the compressed bit-stream is guaranteed to lie in the range 0 through

<sup>3</sup>Key differences between the MQ-coder and the QM-coder [8] are in the probability estimation state machine and the use of bit-stuffing in place of full carry resolution.

<sup>4</sup>The exposition here assumes a 16-bit representation for  $A$ .

Table 1. Symbol and renormalization rates.

bits/pel	0.125	0.25	0.5	1.0	2.0
symbols/pel	0.26	0.50	0.93	1.67	2.91
renorms/pel	0.08	0.17	0.35	0.71	1.46

FF8F. JPEG2000 exploits this property by defining unique marker codes in the range FF90 through FFFF, which may be used to enhance error resilience and facilitate parsing and reorganization of the codestream.

##### 4.2. Complexity Considerations

Arithmetic coding is widely considered to be a highly complex bit-sequential operation which does not lend itself to efficient implementation in hardware or software. This perception is not necessarily well founded, particularly when the symbols have highly skewed distributions, as is usually the case in JPEG2000. In this case the LPS probability  $p_\kappa \ll 1$  so that renormalization events occur rarely. For most symbols, then, the MQ coder performs only three simple steps: i)  $A \leftarrow A - \overline{p_\kappa}$ ; ii)  $C \leftarrow C + \overline{p_\kappa}$ ; and iii) a single bit test for the renormalization condition,  $A < 2^{15}$ . The more expensive operations occur only during renormalization; these include state transitions in the adaptive probability estimator, the assembly of completed code-bytes, and the renormalization steps themselves.

This characteristic may be exploited to achieve throughputs averaging in excess of one symbol per ‘‘clock’’ in dedicated hardware implementations. To see how this is possible, note that the propagation delay through two consecutive 16-bit adders is only marginally larger (by the factor  $1\frac{1}{16}$ ) than that of a single adder. Thus, the quantities  $A_1 = A - \overline{p_{\kappa_1}}$ ,  $A_2 = A_1 - \overline{p_{\kappa_2}}$ , ..., and  $C_1 = C + \overline{p_{\kappa_1}}$ ,  $C_2 = C_1 + \overline{p_{\kappa_2}}$ , ..., may be formed concurrently, together with the tests  $A_1 < 2^{15}$ ,  $A_2 < 2^{15}$ , .... In the extreme case, we may encode all symbols up until the next renormalization event concurrently<sup>5</sup>, so that throughput is limited by the rate at which renormalization events occur. Since renormalization extends the code length by at least one bit, it is possible to achieve average throughputs in excess of one compressed bit per ‘‘clock’’.

High throughput decoders may be implemented in an analogous manner; however, the outcome of the decoding step for each symbol may affect the context for future symbols. Thus, the run of consecutive symbols which may be concurrently decoded in a practical manner is terminated by the first decoded symbol which modifies the context for future symbols, or by renormalization, whichever comes first. Fortunately, the JPEG2000 bit-plane coding primitives have the property that contexts for future symbols are modified only when an insignificant sample becomes significant, which almost invariably corresponds to the LPS and hence coincides with a renormalization event. Consequently, both encoder and decoder offer similar potential for concurrent processing. Table 1 shows the average number of decoded MQ symbols per image pixel, taken over the three large natural images for which results are reported in Table 2. At moderate bit-rates, the last row of the table suggests that approximately 3 symbols may be encoded or decoded concurrently, with the potential for quite high average throughputs.

<sup>5</sup>If an LPS is encoded, it will be the last symbol prior to renormalization, so the modified update relationship may be incorporated as part of the renormalization logic.

## 5. MODE VARIATIONS

The JPEG2000 standard allows for a number of variations on the algorithm described so far. The mode variations are sufficiently minor that the need to support all modes imposes little burden on a compliant decoder.

### 5.1. Parallel Execution of Coding Passes

The JPEG2000 block coder lends itself to parallel implementation in several ways. Firstly, each block may be encoded or decoded in parallel, a form of “macroscopic” parallelism. Secondly, in any given coding pass it is usually possible to process several samples in parallel, since each sample is affected in only one out of every three passes, and multiple symbols can often be processed concurrently, as described above. Unfortunately, none of these techniques can guarantee a high throughput rate (e.g. one sample per “clock”) in each and every code-block, without excessive implementation cost.

To address this concern, it is necessary for all coding passes to be implemented in parallel. Some of the relevant considerations are discussed in [9]. The key steps which must be taken are: 1) terminate the MQ codeword at the end of each coding pass; and 2) initialize the MQ coder and all 18 probability models at the beginning of each coding pass. The JPEG2000 block coder provides mode switches to achieve these steps. At a cost of as little as 0.1dB (with the largest code-blocks), these modes permit parallel encoder implementations. Parallel decoder implementations are also possible; however, compliant decoders must be prepared to decode codestreams which were not generated with the relevant mode switches.

### 5.2. Lazy Coding

As one might expect, symbols coded in the less significant bit-planes tend to have substantially less skewed probability distributions. As a result, little benefit is generally derived from the use of arithmetic coding in the significance propagation,  $\mathcal{P}_1^p$ , and magnitude refinement,  $\mathcal{P}_2^p$ , coding passes for  $p < P - 4$ . The coder provides a mode for bypassing the MQ coder altogether in these coding passes, which can result in significant speedup for software implementations at high bit-rates; it can also reduce the complexity of a parallel coding pass implementation, as discussed above. For most natural images, this mode appears to have negligible effect on compression efficiency; however, compound imagery appears to suffer more significantly.

## 6. PERFORMANCE

In this section we provide some indication of the performance of the JPEG2000 coder by comparing it with the SPIHT algorithm [3], which has become a popular benchmark. Table 2 compares PSNR and decoder CPU time (excluding the inverse Wavelet transform) of the two algorithms, using the JPEG2000 Verification Model (VM7.0) and the implementation of SPIHT (arithmetic coding version) available from “www.rpi.edu”. CPU times are obtained with a Pentium II processor operating at 400MHz. The tests are performed on SNR-scalable bit-streams: a single compressed bit-stream is truncated to each of the indicated test bit-rates and decompressed. The first set of results reported in the table identifies average performance over the three most popular natural images from the JPEG2000 test set, “Bike”, “Cafe” and “Woman”, each of size  $2560 \times 2048$ . The other results correspond to the  $2347 \times 1688$  JPEG2000 test image, “Chart”, and the more widely known  $512 \times 512$  “Barbara” image. This small set of test images is

**Table 2.** PSNR (dB) and CPU time ( $\mu\text{s}/\text{pel}$ ), decoding a single codestream truncated to different rates (bpp).

bpp	Natural		Chart		Barbara	
	SPT	J2G	SPT	J2G	SPT	J2G
0.125	24.6dB 0.37 $\mu\text{s}$	24.8dB .041 $\mu\text{s}$	27.7dB .097 $\mu\text{s}$	28.8dB .053 $\mu\text{s}$	24.9 dB .057 $\mu\text{s}$	25.4dB .031 $\mu\text{s}$
0.25	27.4dB 1.50 $\mu\text{s}$	27.6dB .081 $\mu\text{s}$	31.2dB .170 $\mu\text{s}$	32.5dB .081 $\mu\text{s}$	27.6dB .114 $\mu\text{s}$	28.4dB .099 $\mu\text{s}$
0.5	31.0dB 4.11 $\mu\text{s}$	31.3dB .157 $\mu\text{s}$	36.3dB .329 $\mu\text{s}$	37.5dB .153 $\mu\text{s}$	31.4dB .305 $\mu\text{s}$	32.2dB .152 $\mu\text{s}$
1.0	35.9dB 8.84 $\mu\text{s}$	36.2dB .301 $\mu\text{s}$	42.6dB .618 $\mu\text{s}$	43.7dB .301 $\mu\text{s}$	36.4dB .687 $\mu\text{s}$	37.2dB .290 $\mu\text{s}$

selected primarily to emphasize key features of the coder. To conserve space, we do not report results for the well-known “Lena” image, with which the two coders perform almost identically.

Evidently, JPEG2000 exhibits somewhat higher compression efficiency, while offering a codestream with substantially more flexibility. We draw the reader’s attention to two important properties of the JPEG2000 coder which are revealed by the table. The compression performance obtained with many natural images is only marginally superior to that obtained with SPIHT; however, larger performance differences are observed with images whose Wavelet coefficient is a poorer match to the assumptions implicit in SPIHT. Also, working independently on small blocks, the throughput of the JPEG2000 coder is independent of image size; by contrast, SPIHT suffers substantially from non-local memory accesses as image size increases.

## 7. REFERENCES

- [1] D. Taubman, “High performance scalable image compression with EBCOT,” *IEEE Trans. Image Proc.* July, 2000.
- [2] J. M. Shapiro, “An embedded hierarchical image coder using zerotrees of wavelet coefficients,” *IEEE Data Compression Conference (Snowbird, Utah)*. pp. 214-223, 1993.
- [3] A. Said and W. Pearlman, “A new, fast and efficient image codec based on set partitioning in hierarchical trees,” *IEEE Trans. Circuits and Systems for Video Technology*. pp. 243-250, June 1996.
- [4] E. Ordentlich, M. Weinberger and G. Seroussi, “A low-complexity modeling approach for embedded coding of wavelet coefficients,” *Proc. IEEE Data Compression Conference (Snowbird)*. pp. 408-417, March 1998.
- [5] J. Li and S. Lei, “Rate-distortion optimized embedding,” *Picture Coding Symposium (Berlin)*. pp. 201-206, September 1997.
- [6] W. Pennebaker, J. Mitchell, G. Langdon and R. Arps, “An overview of the basic principles of the Q-coder adaptive binary arithmetic coder,” *IBM J. Res. Develop.* vol. 36, pp. 717-726, November 1988.
- [7] F. Ono et al., “Bi-level image coding with MELCODE- Comparison of block type code and arithmetic type codes” *Globe-com’89*. 7.6.1-6, November 1989.
- [8] W. Pennebaker and J. Mitchell, “JPEG: still image data compression standard,” *Van Nostrand Reinhold, NY* 1992.
- [9] E. Ordentlich, D. Taubman, M. Weinberger, G. Seroussi, and M. Marcellin, “Memory efficient scalable line-based image coding,” *Proc. IEEE Data Compression Conference (Snowbird)*. pp. 218-227, March 1999.