5.6

The JPEG Lossless Image Compression Standards

Nasir Memon
Polytechnic University, Brooklyn

Christine Guillemot
Campus Universitaire de Beaulieu, France

Rashid Ansari
University of Illinois at Chicago

1 Introduction

Although the Joint Photographic Expert Group (JPEG) committee of the International Standards Organization is best known for the development of the ubiquitous *lossy* compression standard, which is commonly known as JPEG compression today, it has also developed some important *lossless* image compression standards. The first lossless algorithm adopted by the JPEG committee, known as JPEG lossless, was developed and standardized along with the well-known lossy standard. However, it had little in common with the lossy standard based on the discrete cosine transform (DCT). The original goals set by the JPEG committee in 1988 stated that the lossy standard should also have a lossless mode that gives about 2 : 1 compression on images similar to the original test set. Perhaps it was also envisioned that both lossy and lossless compression be achieved by a single algorithm working with different parameters. In fact, some of the proposals submitted did have this very same capability. However, given the superior performance of DCT-based algorithms for lossy compression, and given the fact that errors caused by implementing DCT with finite precision arithmetic preclude the possibility of loss-less compression, an entirely different algorithm was adopted for lossless compression. The algorithm chosen was a very simple technique that uses differential pulse code modulation (DPCM) with either Huffman or arithmetic coding for encoding prediction errors.

Although the JPEG lossless algorithm that uses Huffman coding has seen some adoption and a few public domain implementations of it are freely available, the JPEG lossless algorithm based on arithmetic coding has seen little use as of today despite the fact that it provides about 10% to 15% better compression. Perhaps this is due to the intellectual property issues surrounding arithmetic coding and due to the perceived computational and conceptual complexity issues associated with it. To address this problem, the JPEG committee revisited the issue in 1994 and initiated the development of a new lossless image compression standard. A new work item proposal was approved in early 1994, titled *Next Generation Lossless Compression of Continuous-Tone Still Pictures*. A call was issued in March 1994 soliciting proposals specifying algorithms for lossless and near-lossless compression of continuous-tone (2–16 bits) still pictures.

After a few rounds of convergence the final baseline algorithm adopted for standardization was based largely on the revised Hewlett-Packard proposal LOCO-I_{1p} and a DIS

(Draft International Standard) was approved by the committee in 1997 [2]. The new draft standard was named JPEG-LS to distinguish it from the earlier lossy and lossless standards. JPEG-LS baseline is a modern and sophisticated lossless image compression algorithm that despite its conceptual and computational simplicity, yields a performance that is surprisingly close to that of the best known techniques like CALIC (context-based adaptive loseless image coding) [22]. JPEG-LS baseline contains the core of the algorithm and many extensions to it have been standardized.

While the JPEG-LS standard was being finalized, the JPEG committee started working on the standardization of a wavelet-based compression technique, called JPEG2000. JPEG2000 has many "modern" features like embedded quantization, region-of-interest decoding, and so forth. What is important from our point of view is that among other features, JPEG-2000 provides integrated lossy and lossless compression within a single framework.

In the rest of this chapter, the different lossless image compression standards developed by the JPEG committee are described in greater detail. In Section 2 both the Huffman and arithmetic coding versions of the original JPEG lossless standard are presented. In Section 3, JPEG-LS is described. Finally, in Section 4 we discuss the integration of lossless and lossy compression in JPEG2000, the latest standard developed by the JPEG committee.

2 The Original JPEG Lossless Standards

As mentioned in Section 1, the original JPEG lossless standards based on either Huffman or arithmetic coding both used a predictive approach. That is, the algorithm scans an input image, row by row, left to right, predicting each pixel as a linear combination of previously processed pixels and encodes the prediction error. Since the decoder also processes the image in the same order, it can make the same prediction and recover the actual pixel value based on the prediction error. The standard allows the user to choose between eight different predictors, which are listed in Table 1. The notation used for specifying neighboring pixels used in arriving at a prediction is shown in Fig. 1 in the form of a template of two-dimensional (2D) neighborhood pixels. A subset of this neighborhood is generally used for prediction and/or context determination by most lossless image compression techniques. In the rest of this chapter we shall consistently use this notation to denote specific neighbors of the pixel $P[i, j]$ in the i-th row and j-th column.

TABLE 1 JPEG predictors for lossless coding

Mode	Prediction for $P[i, j]$
0	0 (no prediction)
1	N
2	W
3	NW
4	$N+W-NW$
5	$W+(N-NW)/2$
6	$N+(W-NW/2)$
7	$(N+W)/2$

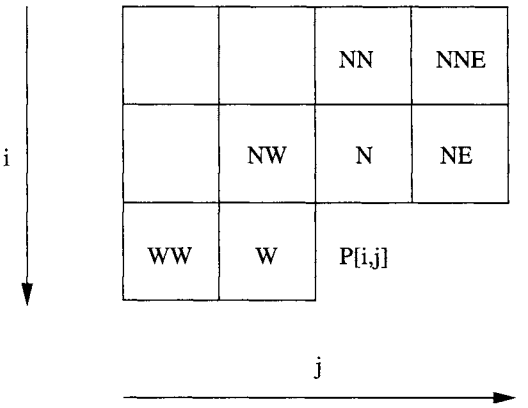

FIGURE 1 Notation used for specifying neighborhood pixels of current pixel $P[i, j]$.

Prediction essentially attempts to capture the intuitive notion that the intensity function of typical images is usually quite "smooth" in a given local region and hence the value at any given pixel is quite similar to its neighbors. In any case, if the prediction made is reasonably accurate, the prediction error has significantly lower magnitude and variance when compared with the original signal, and it can be encoded efficiently with a suitable variable-length coding technique. In JPEG lossless, prediction errors can be encoded using either Huffman or arithmetic coding, codecs for both being provided by the standard. In the rest of this section, we elaborate on the different procedures required or recommended by the standard for Huffman and arithmetic coding.

2.1 Huffman Coding Procedures

In the Huffman coding version, essentially no error model is used. Prediction errors are assumed to be independent and identically distributed (i.i.d.) and they are encoded using the Huffman table provided in the bit stream using the specified syntax. The Huffman coding procedure specified by the standard for encoding prediction errors is identical to the one used for encoding DC coefficient differences in the lossy codec.

Since the alphabet size for the prediction errors is twice the original alphabet size, a Huffman code for the entire alphabet would require an unduly large code table. An excessively large Huffman table can lead to multiple problems. First, a larger code would require more bits to represent. In JPEG, this is not a problem as a special length-limited Huffman code is used that can be specified by a small and fixed number of bits. More important, however, large Huffman tables can lead to serious difficulties in a hardware implementation of the codec.

TABLE 2 Mapping of prediction errors to magnitude category and extra bits

Category	Symbols	Extra bits
0	0	—
1	−1, 1	0, 1
2	−3, −2, 2, 3	00, 01, 10, 11
3	−7, . . . , −4, 4, . . . , 7	000, . . . , 011, 100, . . . , 111
4	−15, . . . , −8, 8, . . . , 15	0000, . . . , 0111, 1000, . . . , 1111
⋮	⋮	⋮
15	−32767, . . . , −16384, 16384, . . . , 32767	0 . . . 00, . . . , 01 . . . 1, 10 . . . 0, . . . , 11 . . . 1
16	32768	

To reduce the size of the Huffman table, each prediction error (or DC difference in the lossy codec) is classified into a "magnitude category" and the label of this category is Huffman coded. Since each category consists of a multiple number of symbols, uncoded "extra bits" are also transmitted that identify the exact symbol (prediction error in the lossless codec, and DC difference in the lossy codec) within the category. Table 2 shows the 17 different categories that are defined. As can be seen, except for the seventeenth (last) category, each category k contains 2^k members $\{\pm 2^{k-1}, \ldots, \pm 2^k - 1\}$ and hence k extra bits would be required to identify a specific symbol within the category. The extra bits for specifying the prediction error e in the category k are given by the k-bit number n by the mapping

$$n = \begin{cases} e & \text{if} \quad e \geq 0 \\ 2^k - 1 + e & \text{if} \quad e < 0 \end{cases}$$

For example, the prediction error−155 would be encoded by the Huffman code for category 8 and the 8 extra bits (01100100) (integer 100) would be transmitted to identify−155 within the 256 different elements that fall within this category. If the prediction error was 155, then (10011011) (integer 155) would be transmitted as extra bits. In practice, this mapping can also be implemented by using the k-bit unsigned representation of e if e is positive and its one's complement if negative.

The Huffman code used for encoding the category label has to meet the following conditions:

- The Huffman code is a length-limited code. The maximum code length for a symbol is 16 bits.
- The Huffman code is a canonical code. The k codewords of given length n are represented by the n-bit numbers $x+1, x+2, \ldots, x+k$ where x is obtained by left shifting the largest numeric value represented by an $(n-1)$-bit codeword.

These two conditions greatly facilitate the specification of a Huffman table and a fast implementation of the encoding and decoding procedures. In a JPEG bit-stream, a Huffman table is specified by two lists, BITS and HUFFVAL. BITS is a 16-byte array contained in the codeword stream where byte n simply gives the number of codewords of length n that are present in the Huffman table. HUFFVAL is a list of symbol values in order of increasing codeword length. If two symbols have the same code length, then the symbol corresponding to the smaller numeric value is listed first. Given these two tables, the Huffman code table can be reconstructed in a relatively simple manner. The standard provides an example procedure for doing this in its informative sections but does not mandate its usage except in the functional sense. That is, given the lists BITS and HUFFVAL, different decoders need to arrive at the same reconstruction, regardless of the procedure used. In practice, many hardware implementations of the lossy codec do not implement the reconstruction procedure and directly input the Huffman table.[1]

Furthermore, the standard only specifies the syntax used for representing a Huffman code. It does not specify how to arrive at the specific length-limited code to be used. One simple way to arrive at a length-limited code is to force probabilities of occurrence for any particular symbol not to be less than 2^{-l} and then run the regular Huffman algorithm. This will ensure that any given codeword does not contain more then l bits. It should be noted that although this procedure is simple, it does not necessary generate an optimal length-limited code. Algorithms for constructing an optimal length-limited code have been proposed in the literature. In practice, however, the above simple procedure works satisfactorily given the small alphabet size for the Huffman table used in JPEG. In addition, the standard also requires that the bit sequence of all ones not be a codeword for any symbol.

When an image consists of multiple components, like color images, separate Huffman tables can be specified for each component. The informative sections of the standard provide example Huffman tables for luminance and chrominance components. They work quite well for lossy compression over a wide range of images and are often used in practice. Most software implementations of the lossy standard permit the use of these "default" tables, allowing an image to be encoded in a single pass. However, since these tables were mainly designed for encoding DC coefficient differences for the lossy codec, they may not work well with lossless compression. For lossless compression, a custom Huffman code table can be specified in the bit stream along with the compressed image. Although this approach requires two passes through the data, it does give significantly better compression. Finally, it should be noted

[1]Actually, what is loaded into the ASIC implementing the lossy codec is not the Huffman code itself but a table that facilitates fast encoding and decoding of the Huffman code.

that the procedures for Huffman coding are common to both the lossy and the lossless standards.

2.2 Arithmetic Coding Procedures

Unlike the version based on Huffman coding, which assumes the prediction error samples to be i.i.d., the arithmetic coding version uses quantized prediction errors at neighboring pixels as contexts for conditional coding of the prediction error. This is a simplified form of error modeling that attempts to capture the remaining structure in the prediction residual. Encoding within each context is done with a binary arithmetic coder by decomposing the prediction error into a sequence of binary decisions. The first binary decision determines if the prediction error is zero. If not zero, the second step determines the sign of the error. The subsequent steps assist in classifying the magnitude of the prediction error into one of a set of ranges and the final bits that determine the exact prediction error magnitude within the range are sent uncoded.

The QM coder is used for encoding each binary decision. A detailed description of the coder and the standard can be found in [12]. Since the arithmetic-coded version of the standard is rarely used, we do not dwell on the details of the procedures used for arithmetic coding and only provide a brief summary. The interested reader can find details in [12].

The QM coder is a modification of an adaptive binary arithmetic coder called the *Q coder* [11], which in turn is an extension of another binary adaptive arithmetic coder called the *skew* coder [14]. Instead of dealing directly with the zeros and ones put out by the source, the QM coder maps them into a more probable symbol (MPS) and less probable symbol (LPS). If 1 represents black pixels, and 0 represents white pixels, then in a mostly black image, 1 will be the MPS, while in an image with mostly white regions, 0 will be the MPS. To make the implementation simple, the committee recommended several deviations from the standard arithmetic coding algorithm. The update equations in arithmetic coding that keep track of the subinterval to be used for representing the current string of symbols involve multiplications that are expensive in both hardware and software. In the QM coder, expensive multiplications are avoided and rescalings of the interval take the form of repeated doubling, which corresponds to a left shift in the binary representation. The probability q_c of the LPS for context C is updated each time a rescaling takes place and the context C is active. An ordered list of values for q_c is kept in a table. Every time a rescaling occurs, the value of q_c is changed to the next lower or next higher value in the table, depending on whether the rescaling was caused by the occurrence of an LPS or MPS. In a nonstationary situation, it may happen that the symbol assigned to LPS actually occurs more often than the symbol assigned to MPS. In this situation, the assignments are reversed (i.e., the symbol assigned the LPS label is assigned the MPS label and vice versa). The test is conducted every time a rescaling takes place. The decoder for the QM coder operates in much the same way as the encoder, by mimicking the encoder operation.

3 JPEG-LS Lossless Compression Standard

As mentioned earlier, the JPEG-LS algorithm, like its predecessor, is a predictive technique. However, there are significant differences as described in the following list:

- Instead of using a simple linear predictor, JPEG-LS uses a nonlinear predictor that attempts to detect the presence of edges passing through the current pixel and accordingly adjusts prediction. This results in a significant improvement in performance in the prediction step.
- Like JPEG lossless arithmetic, JPEG-LS uses some simple but very effective context modeling of the prediction errors prior to encoding.
- Baseline JPEG-LS uses Golomb-Rice codes for encoding prediction errors. Golomb-Rice codes are Huffman codes for certain geometric distributions which serve well in characterizing the distribution of prediction errors. Although Golomb-Rice codes have been known for a long time, JPEG-LS uses some novel and highly effective techniques for adaptively estimating the parameter for the Golomb-Rice code to be used in a given context.
- To effectively code low-entropy images or regions, JPEG-LS uses a simple alphabet extension mechanism, by switching to a run-length mode when a uniform region is encountered. The run-length coding used is again an extension of Golomb codes and provides significant improvement in performance for highly compressible images.
- For applications that require higher compression ratios, JPEG-LS provides a near-lossless mode that guarantees each reconstructed pixel to be within a distance k from its original value. Near-lossless compression is achieved by a simple uniform quantization of the prediction error.

An overview of the JPEG-LS baseline algorithm is shown in Fig. 2. In the rest of this section, we describe in more detail each of the steps involved in the algorithm and some of the extensions that are currently under the process of standardization. For a detailed description the reader is referred to the working draft [2].

3.1 The Prediction Step

JPEG-LS uses a very simple and effective predictor, the median edge detection (MED) predictor, that adapts in presence of local edges. MED detects horizontal or vertical edges by examining the North N, West W and Northwest NW neighbors of the current pixel $P[i,j]$. The North (West) pixel is used as a prediction in the case of a vertical (horizontal) edge. In case of

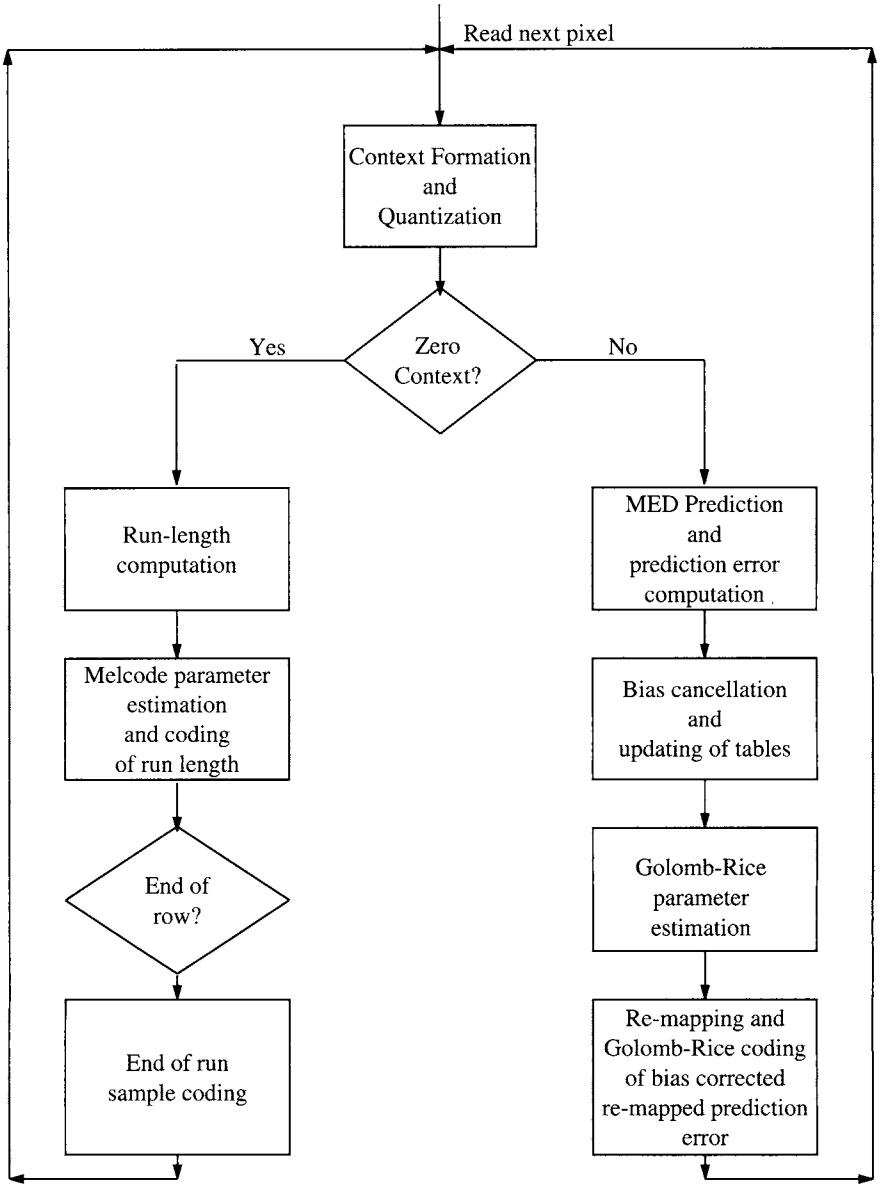

FIGURE 2 An overview of baseline JPEG-LS.

neither, planar interpolation is used to compute the prediction value. Specifically, prediction is performed according to the following equations:

$$\hat{P}[i,j] = \begin{cases} \min(N,W) & \text{if } NW < \max(N,W) \\ \max(N,W) & \text{if } NW < \min(N,W) \\ N + W - NW & \text{otherwise} \end{cases}$$

The MED predictor is essentially a special case of the median adaptive predictor (MAP), first proposed by Martucci in 1990 [7]. Martucci proposed the MAP predictor as a nonlinear adaptive predictor that selects the median of a set of three predictions in order to predict the current pixel. One way of interpreting such a predictor is that it always chooses either the best or the second best predictor among the three candidate predictors. Martucci reported the best results with the following three predictors, in which case it is easy to see that MAP turns out to be the MED predictor.

1. $\hat{P}[i,j] = N$
2. $\hat{P}[i,j] = W$
3. $\hat{P}[i,j] = N + W - NW$

In an extensive evaluation, Memon and Wu [8, 9] observed that the MED predictor gives a performance that is superior to, or almost as good as, that of several standard prediction techniques, many of which are significantly more complex.

3.2 Context Formation

Gradients alone cannot adequately characterize some of the more complex relationships between the predicted pixel $P[i,j]$ and its surrounding area. Context modeling of the prediction error $e = \hat{P}[i,j] - P[i,j]$ can exploit higher-order structures such as texture patterns and local activity in the image for further compression gains. Contexts in JPEG-LS are formed by first computing the following differences:

$$\begin{aligned} D_1 &= NE - N \\ D_2 &= N - NW \\ D_3 &= NW - W \end{aligned} \tag{1}$$

where the notation for specifying neighbors is as shown in Fig. 1. The differences D1, D2, and D3 are then quantized into 9 regions (labeled –4 to +4) symmetric about the origin with one of the quantization regions (region 0) containing only the difference value 0. Further, contexts of the type (q_1, q_2, q_3) and $(-q_1, -q_2, -q_3)$ are merged based on the assumption that

$$P(e|q_1, q_2, q_3) = P(-e| - q_1, -q_2, -q_3).$$

The total number of contexts turn out to be $\frac{9^3-1}{2} = 364$. These contexts are then mapped to the set of integers [0, 363] in a one-to-one fashion. The standard does not specify how contexts are mapped to indices and vice versa, leaving it completely to the implementation. In fact, two different implementations could use different mapping functions and even a different set of indices but nevertheless be able to decode files encoded by the other. The standard only requires that the mapping be one-to-one.

3.3 Bias Cancellation

As described earlier, in JPEG arithmetic, contexts are used as conditioning states, for encoding prediction errors. Within each state, the pdf of the associated set of events is adaptively estimated from events by keeping occurrence counts for each context. Clearly, to better capture the structure preset in the prediction residuals one would like to use a large number of contexts or conditioning states. However, the larger the number of contexts, the more the number of parameters (conditional probabilities in this case) that need to be estimated based on the same data set. This can lead to the "sparse context" or "high model cost" problem. In JPEG lossless arithmetic, this problem is addressed by keeping the number of contexts small and decomposing the prediction error into a sequence of binary decisions, each requiring estimation of a single probability value. Although this results in alleviating the sparse context problem, there are two problems caused by such an approach. First, keeping the number of conditioning states to a small number fails to capture effectively the structure present in the prediction errors and results in poor performance. Second, binarization of the prediction error necessitates the use of an arithmetic coder which adds to the complexity of the coder.

The JPEG-LS baseline algorithm uses a different solution for this problem. First, it uses a relatively large number of contexts to capture the structure present in the prediction errors. However, instead of estimating the pdf of prediction errors, $p(e|C)$, within each context C, only the conditional expectation $E\{e|C\}$ is estimated using the corresponding sample means $\bar{e}(C)$ within each context. These estimates are then used to further refine the prediction prior to entropy coding, by an error feedback mechanism that cancels prediction biases in different contexts. This process is called *bias cancellation*. Furthermore, for encoding the bias-cancelled prediction errors, instead of estimating the probabilities of each possible prediction error, baseline JPEG-LS essentially estimates a parameter that serves to characterize the specific pdf to be used from a fixed set of pdf's. This is explained in greater detail in the next subsection.

A straightforward implementation of bias cancellation would require accumulating prediction errors within each context and keeping frequency counts of the number of occurrences for each context. The accumulated prediction error within a context divided by its frequency count would then be used as an estimate of prediction bias within the context. However, this division operation can be avoided by a simple and clever operation that updates variables in a suitable manner producing average prediction residuals in the interval $[-0.5, 0.5]$. For details, the reader is referred to the proposal that presented this technique and also to the DIS [2, 21]. Also, since JPEG-LS uses Golomb-Rice codes, which assign shorter codes to negative residual values than to positive ones, the bias is adjusted such that it produces average prediction residuals in the interval $[-1, 0]$, instead of $[-0.5, 0.5]$. For a detailed justification of this procedure, and other details pertaining to bias estimation and cancellation mechanisms, see [21].

3.4 Rice-Golomb Coding

Before the development of JPEG-LS, the most popular compression standards, like JPEG, MPEG, H263, and CCITT Group 4, have essentially used static Huffman codes in the entropy coding stage. This is because adaptive Huffman coding does not provide enough compression improvement to justify the additional complexity. Adaptive arithmetic coding, on the other hand, despite being used in standards like JBIG and JPEG arithmetic, has also seen little use due to concerns about intellectual property restrictions and also perhaps due to the additional computational resources that are needed. JPEG-LS is the first international compression standard that uses an adaptive entropy coding technique that requires only

a single pass through the data and requires computational resources that are arguably lesser than what is needed by static Huffman codes.

In JPEG-LS, prediction errors are encoded using a special case of Golomb codes [5], which is also known as Rice coding [13]. Golomb codes of parameter m encode a positive integer n by encoding n modulo m in binary followed by an encoding of n div m in unary. The unary coding of n is a string of n 0 bits, followed by a terminating 1 bit. When $m = 2^k$ Golomb codes have a very simple realization and have been referred to as Rice coding in the literature. In this case n mod m is given by the k least significant bits and n div m by the remaining bits. So, for example, the Rice code of parameter 3 for the 8 bit number 42 (00101010 in binary) is given by 010000001 where the first 3 bits 010 are the three least significant bits of 42 (which is the same as 42 mod 8) and the remaining 6 bits represent the unary coding of 42 div 8 = 5, which is represented by the remaining 5 bits 00101 in the binary representation of 42. Note that, depending on the convention being used, the binary code can appear before the unary code and the unary code could have leading ones terminated by a zero instead.

Clearly, the number of bits needed to encode a number n depend on the parameter k used. For the example above, if $k = 2$ was used we would get a code length of 13 bits and the parameter 4 would result in 7 bits being used. It turns out that given an integer n the Golomb-Rice parameter that results in the minimum code length of n is $\lceil \log_2 n \rceil$.

From these facts, it is clear that the key factor behind the effective use of Rice codes is estimating the parameter k to be used for a given sample or block of samples. Rice's algorithm [13] tries codes with each parameter on a block of symbols and selects the one that results in the shortest code as suggested. This parameter is sent to the decoder as side information. However, in JPEG-LS the coding parameter k is estimated on the fly for each prediction error using techniques proposed by Weinberger et al. [20]. Specifically, the Golomb parameter is estimated by maintaining in each context, the count N of the prediction errors seen so far and the accumulated sum of magnitude of prediction errors A seen so far. The coding context k is then computed as

$$k = \min\{k' | 2^{k'} \cdot N \geq A\}.$$

The strategy used is an approximation to optimal parameter selection for this entropy coder. Despite the simplicity of the coding and estimation procedures, the compression performance achieved is surprisingly close to that obtained by arithmetic coding. For details the reader is referred to [20].

Also, since Golomb-Rice codes are defined for positive integers, prediction errors need to be accordingly mapped. In JPEG-LS, prediction errors are first reduced to the range $[-128, 128]$ by the operation $e = y - x$ mod 256 and then mapped to positive values by

$$m = \begin{cases} 2e & \text{if } e \geq 0 \\ -2e - 1 & \text{if } e < 0 \end{cases}$$

3.5 Alphabet Extension

The use of Golomb-Rice codes is very inefficient when coding low-entropy distributions because the best coding rate achievable is 1 bit per symbol. Obviously, for entropy values of less than 1 bit per symbol, such as would be found in smooth regions of an image, this can be very wasteful and lead to significant deterioration in performance. This problem can be alleviated by using *alphabet extension*, wherein blocks of symbols rather than individual symbols are coded, thus spreading the excess coding length over many symbols. The process of blocking several symbols together prior to coding produces less skewed distributions, which is desirable.

To implement alphabet extension, JPEG-LS first detects smooth areas in the image. Such areas in the image are characterized by the gradients $D1, D2$, and $D3$, as defined in equation 1, all being zero. In other words the context $(0, 0, 0)$, which we call the *zero context*. When a zero context is detected, the encoder enters a run mode where a run of the west symbol B is assumed and the total run of the length is encoded. End-of-run state is indicated by a new symbol $x \neq B$, and the new symbol is encoded using its own context and special techniques described in the standard [2]. A run may also be terminated by the end of line in which case only the total length of run is encoded.

The specific run-length coding scheme used is the MELCODE described in [10]. MELCODE is a binary coding scheme where target sequences contain a most probable symbol (MPS) and a least probable symbol (LPS). In JPEG-LS, if the current symbol is the same as the previous, an MPS is encoded otherwise a LPS is encoded. Runs of the MPS of length n are encoded using only 1 bit. If the run is of length less than n (including 0) it is encoded by a zero bit followed by the binary value of the run length encoded using $\log n$ bits. The parameter n is constrained to be of the form 2^k and is adaptively updated while encoding a run. For details of the adaptation procedure and other details pertaining to the run-mode the reader is referred to the draft standard [2]. Again the critical factor behind effective usage of the MELCODE is the estimation of the parameter value n to be used.

3.6 Near-Lossless Compression

Although lossless compression is required in many applications, compression ratios obtained with lossless techniques are significantly lower than those possible with lossy compression. Typically, depending on the image, lossless compression ratios range from about 1.5:1 to 3:1. On the other hand, state-of-the-art lossy compression techniques give

compression rations in excess of 20 : 1 with virtually no loss in visual fidelity. However, in many applications, the end-use of the image is not human perception. In such applications, the image is subjected to postprocessing to extract parameters of interest like ground temperature or vegetation indices. The uncertainty about reconstruction errors introduced by a lossy compression technique is undesirable.

This leads to the notion of a *near-lossless* compression technique that gives quantitative guarantees about the type and amount of distortion introduced. Based on these guarantees, a scientist can be assured that the extracted parameters of interest will either not be affected or be affected only within a bounded range of error. Near-lossless compression could potentially lead to significant increase in compression, thereby giving more efficient utilization of precious bandwidth while preserving the integrity of the images with respect to the postprocessing operations that are carried out.

JPEG-LS has a near-lossless mode that guarantees a $\pm k$ reconstruction error for each pixel. Extension of the lossless baseline algorithm to the case of near-lossless compression is achieved by prediction error quantization according to the specified pixel value tolerance. For the predictor at the receiver to track the predictor at the encoder, the reconstructed values of the image are used to generate the prediction at both the encoder and the receiver. This is the classic DPCM structure. Specifically, the prediction error is quantized according to the following rule:

$$Q[e] = \left\lfloor \frac{e+k}{2k+1} \right\rfloor (2k+1), \tag{2}$$

where e is the prediction error, k is the maximum reconstruction error allowed in any given pixel and $\lfloor . \rfloor$ denotes the integer part of the argument. At the encoder, a label l is generated according to

$$l = \left\lfloor \frac{e+k}{2k+1} \right\rfloor. \tag{3}$$

This label is encoded, and at the decoder the prediction error is reconstructed according to

$$\hat{e} = l \times (2k+1). \tag{4}$$

This form of quantization, where all values in the interval $[n \cdot k - \lfloor \frac{k}{2} \rfloor,\ n \cdot k + \lfloor \frac{k}{2} \rfloor]$ are mapped to $n \cdot k$, is a special case of *uniform quantization*. It is well known that uniform quantization leads to a minimum entropy of the output, provided the step size is small enough for the constant pdf assumption to hold. For small values of k, as one would expect to be used in near-lossless compression, this assumption is reasonable.

It has been experimentally observed that for bit rates exceeding 1.5 bpp, JPEG-LS near-lossless actually gives better performance than baseline lossy JPEG. However, it should be noted that the uniform quantization performed in JPEG-LS near-lossless often gives rise to annoying "contouring" artifacts. Such artifacts are most visually obvious in smooth regions of the image. In Chapter 1.1 of this volume, such "false contouring" is examined in more detail and shown to possibly occur even from simple image quantization.

In many cases, these artifacts can be reduced by some simple postprocessing operations. As explained in the next section, JPEG-LS part 2 allows variation of the quantization step size spatially in a limited manner, thereby enabling some possible reduction in artifacts.

Finally, it may appear that the quantization technique used in JPEG-LS is overly simplistic. In actuality, there is a complex dependency between the quantization error that is introduced and subsequent prediction errors. Clearly, quantization affects the prediction errors obtained. Although one can vary the quantization in an optimal manner using a trellis-based technique and the Viterbi algorithm, it has been observed that such computationally expensive and elaborate optimizing strategies offer little advantage, in practice, over the simple uniform quantization used in JPEG-LS [3].

3.7 JPEG-LS Part 2

Even as the baseline algorithm was being standardized, the JPEG committee initiated development of JPEG-LS part 2. Initially the motivation for part 2 was to standardize an arithmetic coding version of the algorithm. As it evolved, however, part 2 also includes many other features that improve compression but were considered to be too application-specific to include in the baseline algorithm. As a result, JPEG-LS part 2 is an algorithm that is substantially different from part 1 although the basic approach, in terms of prediction followed by context-based modeling and coding of prediction errors, remains the same as the baseline. In the following sections, we briefly describe some of the key features that are part of the standard.

3.7.1 Prediction

JPEG-LS baseline is not suitable for images with sparse histograms (prequantized images or images with less than 8 or 16 bits represented by 1 or 2 bytes, respectively). This is, because predicted values of pixels do not actually occur in the image, which causes code space to be wasted during the entropy coding of prediction errors. To deal with such images, part 2 defines an optional *prediction value control* mode wherein it is ensured that a predicted value is always a symbol that has actually occurred in the past. This is done by forming the same prediction as JPEG baseline using the MED predictor, but by adjusting the predictor to a value that has been seen

before. A flag array is used to keep track of pixel values that have occurred thus far.

3.7.2 Context Formation

To better model prediction errors, part 2 uses an additional gradient

$$D4 = WW - W \tag{5}$$

where W and WW are the neighboring pixels as shown in Fig. 1. D4 is quantized along with D1, D2, and D3 (defined earlier in equation 1) just as in the baseline. D4 is however quantized only to three regions. Context merging is done only on the basis of D_1, D_2, D_3, and not D_4. That is, contexts of type (q_1, q_2, q_3, q_4) and $(-q_1, -q_2, -q_3, q_4)$ are merged to arrive a total of $364 \times 3 = 1{,}092$ contexts.

3.7.3 Bias Cancellation

In the bias-cancellation step of baseline JPEG-LS, prediction errors within each context are centered around –0.5 instead of 0. As explained, this was done because the prediction error mapping technique and Rice-Golomb coding used in the baseline algorithm assign shorter code words to negative errors as opposed to a positive error of the same magnitude. However, if arithmetic coding is used then there is no such imbalance and bias cancellation is used to center the prediction error distribution in each context around zero. This is exactly the bias cancellation mechanism proposed in CALIC.

3.7.4 Alphabet Extension

If arithmetic coding is used, alphabet extension is clearly not required. Hence, in the arithmetic coding mode, the coder does not switch to run-mode on encountering the all-zeroes context. However, in addition to this change, part 2 also specifies some small changes to the run-length coding mode of the original baseline algorithm. For example, when the underlying alphabet is binary, part 2 eliminates the redundant encoding of sample values that terminated a run, as required by the baseline.

3.7.5 Arithmetic Coding

Even though the base-line algorithm has an alphabet extension mechanism for low-entropy images, performance can be significantly improved by the use of arithmetic coding. Hence, the biggest difference between JPEG-LS and JPEG-LS part 2 is in the entropy coding stage. Part 2 uses a binary arithmetic coder for encoding prediction errors that are binarized by a Golomb code. The Golomb code tree is produced on the basis of an activity class of the context computed from its current average prediction error magnitude. Twelve activity levels are defined. In the arithmetic coding procedure, numeric data are treated in radix 255 representation, with each sample expressed as 8-bit data. Probabilities of the MPS and the LPS are estimated by keeping occurrence counts. Multiplication and division are avoided by approximate values stored in a look-up table.

3.7.6 Near-Lossless Mode

The near-lossless mode is another area where part 2 differs significantly from the baseline. Essentially, part 2 provides mechanisms for a more versatile application of the near-lossless mode. The two main features enabled by Part 2 in the near-lossless mode are

- *Visual quantization.* As mentioned before, near-lossless compression can often lead to annoying artifacts at larger values of k. Furthermore, the baseline does not provide any graceful degradation mechanism between step size of k and $k + 1$. Hence JPEG-LS part 2 defines a new "visual quantization" mode. In this mode, the quantization step size is allowed to be either k or $k + 1$ depending on the context. Contexts with a larger gradients use a step size of $k + 1$ and contexts with smaller gradients use a step size of k. The user specifies a threshold based on which this decision is made. The standard does not specify how the threshold should be arrived at. It only provides a syntax for its specification.
- *Rate control.* By allowing the user to change the quantization step size while encoding an image, part 2 essentially provides a rate-control mechanism whereby the coder can keep track of the coded bytes, based on which appropriate changes to the quantization step size can be made. The encoder, for example, can compress the image to less than a bounded size with a single sequential pass over the image. Other uses of this feature are possible, including region-of-interest lossless coding, etc.

3.7.7 Fixed-Length Coding

There is a possibility that a Golomb code will cause data expansion and result in a compressed image larger than the source image. To avoid such a case, an extension to the baseline is defined where the encoder can switch to a fixed-length coding technique by inserting an appropriate marker in the bit-stream. Another marker is used to signal the end of fixed-length coding. The procedure for determining if data expansion is occurring and for selecting the size of the fixed-length representation is left entirely up to the implementation. The standard does not make any recommendation.

3.7.8 Interband Correlations

Currently there is no mechanism for exploiting interband correlations in JPEG-LS baseline and JPEG-LS part 2. The application is expected to decorrelate individual bands prior to encoding by JPEG-LS. The lack of informative or

normative measures for exploiting interband correlations, in our opinion is the most serious shortcoming of the JPEG-LS standard.

4 JPEG2000 and the Integration of Lossless and Lossy Compression

In prediction-based lossless image compression techniques, image pixels are processed in some fixed and predetermined order. The intensity of each pixel is modeled as being dependent on the intensities values in a fixed and pre-determined neighborhood set of previously visited pixels. As a result, such techniques do not adapt well to the nonstationary nature of image data. Furthermore, such techniques form predictions and model the prediction error solely on the basis of local information. Hence, they usually do not capture "global patterns" that influence the intensity value of the current pixel being processed. As a consequence, recent years have seen techniques based on a predictive approach rapidly reach a point of diminishing returns. JPEG-LS, the new lossless standard provides testimony to this fact. Despite being extremely simple, it provides compression performance that is within a few percentages of more sophisticated techniques such as CALIC [22] and universal context modeling (UCM) [19]. Experimentation suggests that an improvement of significantly more than 10% on an average is unlikely to be obtained by pushing the envelope on the current state-of-the-art predictive techniques like CALIC [9]. Furthermore, the complexity costs incurred for obtaining these improvements are enormous and usually not worth the marginal improvement in compression that is obtained.

An alternative approach to lossless image compression that has emerged recently is based on subband (or wavelet) decomposition. Subband decomposition provides a way to cope with non-stationarity of image data by separating the information into several scales and exploiting correlations within each scale as well as across scales. A subband approach also provides a better framework for capturing global patterns in the image data. Finally, the wavelet transforms used in the decomposition can be viewed as a prediction scheme (as in [4, 15]) that is not restricted to a casual template but makes a prediction of the current pixel based on "past" and "future" pixels with respect to a spatial raster scan.

In addition to the above, there are other advantages offered by a subband approach for lossless image compression. The most important of these is perhaps the natural integration of lossy and lossless compression that the subband approach makes possible. By transmitting entropy-coded subband coefficients in an appropriate manner, one can produce an embedded bit stream that permits the decoder to extract a lossy reconstruction at the desired bit rate. This enables progressive decoding of the image that can ultimately lead to lossless reconstruction [15, 16, 23]. The image can also be recovered at different spatial resolutions. These features are of great value for specific applications like teleradiology and the Web, and for applications in "network-centric" computing in general. More details on these are given in Chapter 4.2 (wavelets), Chapter 5.4 (wavelet image coding), and Chapter 6.2 (wavelet video coding) of this book.

The above facts have caused an increasing popularity of the subband approach for lossless image compression. Some of the early work done toward applying subband image coding techniques for lossless image compression includes techniques such as S+P [15], CREW [23], and EBCOT [16]. The new JPEG standard, JPEG2000, specifies a lossless component that has brought subband based techniques to the mainstream of lossless image compression.

4.1 JPEG2000 Lossless Coding: The Reversible Path

JPEG2000 specifies two coding paths: an irreversible and a reversible path. The irreversible path can be used for lossy compression only. The main components of this coding path are described in Chapter 5.5. Lossless coding requires the use of the reversible path. Note, however, that the reversible path allows the embedding of both a lossy and lossless representation in the code stream. The entire codestream gives a lossless compressed representation of the image. A lossy representation can be achieved by truncating the code stream at any given bit rate.

The main components of the reversible path are shown in Fig. 3. Most of the components of the two coding paths (irreversible and reversible) (e.g., tiling, level offset, entropy coding, bitstream structuring, ROI coding, and so forth), are common and are described in Chapter 5.5. This section describes the components that have been introduced specifically to support lossless coding (i.e., the reversible color transform, the reversible discrete wavelet transform, and the ranging operator).

4.2 Reversible Color Transform

The first steps in both paths (tiling, level offset, and color transform) are optional and can be regarded as preprocessing steps. The image may first be partitioned into rectangular and non-overlapping tiles of equal size. If the sample values are unsigned and represented with B bits, an offset of -2^{B-1} is added leading to a signed representation in the range $[-2^{B-1}, 2^{B-1}]$ (i.e., to a symmetric distribution about 0). The color component samples may be converted into luminance and colour difference components via a reversible color transform (RCT). Note that the RCT can also be used for lossy coding. It thus allows the embedding of a lossy and a lossless representation of the image in a single codestream. The RCT is a reversible integer-to-integer transform that

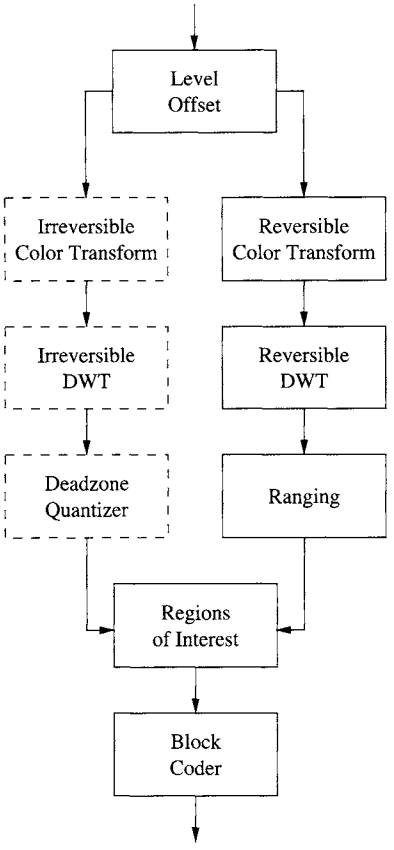

FIGURE 3 Components of the JPEG2000 reversible coding path. The boxes in dotted lines correspond to the JPEG2000 lossy coding mode (see Chapter 5.5).

actually approximates the irreversible component transform (ICT) and is defined as

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 1/4 & 2/4 & 1/4 \\ 1 & -1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

When the three image components are transformed via the RCT, their original bit depth must be identical. After application of the RCT, the luminance component retains the same bit depth, but the color difference components see their bit depth increased by 1. This has to be taken into account when choosing the range parameters (see Section 4.4) for the chrominance components.

4.3 Reversible Discrete Wavelet Transform

Each tile component is decomposed with a forward DWT into a set of $L = 2^l$ resolution levels using a dyadic decomposition. A detailed and complete presentation of filter banks and wavelets theoretic and implementation aspects is beyond the scope of this chapter. The reader is referred to Chapter 4.2 in this book and to [18] for additional insight on these issues.

TABLE 3 Reversible discrete wavelet transform analysis and synthesis filters coefficients

Index	Low-Pass Analysis Filter Coefficient	High-Pass Analysis Filter Coefficient	Low-Pass Synthesis Filter Coefficient	High-Pass Synthesis Filter Coefficient
0	6/8	1	1	6/8
+/−1	6/8	−1/2	1/2	−2/8
+/−2	−1/8			−1/8

The forward DWT is based on separable wavelet filters, which, in the reversible path, are reversible. The transform is then referred to as RDWT (reversible discrete wavelet transform). The use of the RDWT allows the embedding of both lossless and lossy compression in a single compressed codestream. The default RDWT is based on a lifting implementation of the spline 5/3 wavelet transform first introduced in [6] with five levels of decomposition. The filter coefficients are provided in Table 3. Even with integer filters such as the 5/3 wavelet kernel, the precision required at the different levels of decomposition for achieving a lossless representation may rapidly become very large.

A lifting implementation allows a convenient implementation of an integer-to-integer transform. A lifting operation consists of a prediction step followed by an update step. In the prediction step, each odd sample is predicted as a linear combination of the even samples. A prediction error is computed by subtracting the predicted value from the odd sample. The update step updates the even samples by adding to them a linear combination of the modified odd samples. To build the integer-to-integer transform, quantizers are inserted after the calculation of the prediction and the update terms. For example, in converting the 5/3 wavelet kernel into an integer-to-integer transform, the two quantizers $Q_P = -\lfloor -w \rfloor$ and $Q_U = \lfloor w + 1/2 \rfloor$ are inserted in the prediction and update steps. The forward transform hence computes

$$\begin{aligned} y(2n+1) &= x(2n+1) - \left\lfloor \frac{x(2n) + x(2n+2)}{2} \right\rfloor \\ y(2n) &= x(2n) + \left\lfloor \frac{y(2n-1) + y(2n+1) + 2}{4} \right\rfloor \end{aligned} \tag{6}$$

The inverse transform reconstructs the original samples by computing

$$\begin{aligned} x(2n) &= y(2n) - \left\lfloor \frac{y(2n-1) + y(2n+1) + 2}{4} \right\rfloor \\ x(2n+1) &= y(2n+1) - \left\lfloor \frac{x(2n) + x(2n+2)}{2} \right\rfloor \end{aligned} \tag{7}$$

The choice of the lifted integer-to-integer 5/3 wavelet kernel was driven by requirements of low implementation complexity and lossless encoding capability. Note however that, JPEG2000 part 2, allows arbitrary filter specifications in the codestream.

4.4 Ranging

In the reversible path, there is no explicit quantization. The integer transformed coefficients are supplied directly to the block entropy coder described in Chapter 5.5. A nominal range for these integer sample values may be determined from the original bit depth B and from the gains of the wavelet transform kernels. As in the irreversible path, one can also add G extra guard bits to accommodate excursions beyond the nominal range bounds. The number of bits required to represent the subband coefficients $y_i(j)$ of a subband i is hence given by $M_i = B - 1 + G + X_i$ where X_i is an extra term to account for the nominal gains of the analysis kernels for the different bands. The quantity M_i can thus be expressed similarly as in the irreversible path as $M_i = \epsilon_i + G - 1$. However, whereas in the irreversible path the parameter ϵ_i represents the quantization step exponent, in the reversible path, ϵ_i is called the ranging parameter and must be chosen so that a sufficient number of bits is available to represent the subband coefficient magnitudes. A reasonable policy for the encoder is to set $\epsilon_i = B + X_i$.

Although there is no explicit quantization, the embedded bitstreams corresponding to individual code blocks B_i can be truncated prior to decoding. This can be regarded as quantizing a sample $y_i(j)$ in the block B_i with a scalar deadzone quantizer of step size $\Delta_i(j) = 2^{p_i(j)}$. Here $p_i(j)$ represents the least significant bit of $y_i(j)$ that is truncated, and hence set to zero.

5 Discussion

The current situation leaves us with three different lossless image compression standards. This leads to the legitimate question — which one is the best? If one were just to look at compression performance for single-band images then one could rank them as follows: (a) JPEG-LS arithmetic coding, (b) JPEG arithmetic coding, (c) JPEG-LS baseline, (d) JPEG2000, and (e) JPEG Huffman coding. So if compression is the only issue, JPEG-LS arithmetic coding version is the best. However, if a hardware implementation is needed, JPEG-LS gives poor throughput performance as it is not parallelizable given its very sequential context modelling component. The arithmetic coding version of the old JPEG algorithm, however, can give almost the same performance at a much higher throughput. This same situation does not hold in a software implementation. So the best algorithm really depends on the application which is is being used for. For example, if features like embedded quantization, region-of-interest decoding, and integration of lossy and lossless compression are important, JPEG2000 would be the choice. However, the computational and memory requirements of a wavelet-based approach are typically very high. Such techniques are not suitable for printers and other applications where additional memory adds to the fixed cost of the product.

Hence, we expect all three JPEG lossless standards to be used in practice, because together they cover the range of applications that require lossless image compression. Given this fact, we also do not expect any significant improvements in the state of the art of lossless image compression over these three established standards. Our statement is supported by the fact that the last few years have not seen any algorithm that can compete with the three standards in terms of price-performance tradeoffs. There have been many extremely complex algorithms proposed but the improvements they provide over, say, JPEG-LS arithmetic have been less than 0.1 bits per pixel. Of course, there will always be niche applications where a custom designed lossless compression technique may significantly out-perform the three standards. However, such applications are not common and even for these, the improvements often turn out to be surprisingly below expectations.

6 Additional Information

A free implementation of the Huffman based original JPEG lossless algorithm, written by the PVRG group at Stanford, is available from havefun.stanford.edu:/pub/jpeg/JPEGv1.2.1.tar.Z. The PVRG code is designed for research and experimentation rather than production use, but it is easy to understand. There is also a lossless-JPEG-only implementation available from Cornell (ftp.cs.cornell.edu:/pub/multimed/ljpg.tar.Z). Neither the PVRG nor Cornell codecs are being actively maintained. They are both written in the C language and can be ported to a variety of operating systems including variants of UNIX and the different Microsoft platforms.

The JPEG committee maintains a Web site at www.jpeg.org where the JPEG-LS standard documents are available. Another site maintained by Hewlett-Packard at www.hpl.hp.com/loco/ contains an example decoder, which is a public-domain executable of their JPEG-LS implementation for Win95/NT, HP-UX, and SunOS. This site also contains literature on LOCO-I, the algorithm on which JPEG-LS baseline is largely based.

Further information on the different parts of the JPEG2000 standard can be found at www.jpeg.org/jpeg2000/index.html. This Web site provide links to sites from which various official standard and other documents can be downloaded. It also provides links to sites from which software implementations of the standard can be downloaded.

Some software implementations are available at http://jpeg 2000.epfl.ch, http://www.ee.unsw.edu.au/taubman/kakadu, and http://www.ece.ubc.ca/mdadams/jasper/.

References

[1] ISO/IEC JTC 1/SC 29/WG 1, "Call for contributions — lossless compression of continous-tone still pictures," ISO Working Document ISO/IEC JTC1/SC29/WG1 N41 (March 1994).

[2] ISO/IEC JTC 1/SC 29/WG 1, "JPEG LS image coding system," ISO Working Document ISO/IEC JTC1/SC29/WG1 N399 - WD14495 (1996).

[3] R. Ansari, N. Memon and E. Ceran, "Near-lossless image compression techniques, *J. Elect. Imaging* **7** 486-494 (1998).

[4] A. R. Calderbank, I. Daubechies, W. Sweldens, and B. L. Yeo, "Wavelet transforms that map integers to integers," *SIAM J. Appl. Math.* (1996).

[5] S. W. Golomb, "Run-length codings," *IEEE Trans. Inf. Theory*, **12** 399–401 (1996).

[6] D. Le Gall and A. Tabatabai, "Subband coding of digital images using symmetric short kernel filters and arithmetic coding techniques," in *Proc. Intl. Conf. Acoust. Speech Signal Proc. ICASSP-88* (1988) 761–764.

[7] S. A. Martucci, "Reversible compression of HDTV images using median adaptive prediction and arithmetic coding," in *IEEE Int. Symp. Circuits Systems* (IEEE, New York, 1990), pp. 1310–1313.

[8] N. D. Memon and K. Sayood, "Lossless image compression — a comparative study," in *Still Image Compression*, Proc. SPIE vol 2418 (1995).

[9] N. D. Memon and X. Wu, "Recent developments in lossless image compression," *Comput. J.* **40** 31–40 (1997).

[10] S. Ono, S. Kino, M. Yoshida, and T. Kimura, "Bi-level image coding with MELCODE — comparison of block type code and arithmetic type code," *Proc. Globecomm 89* (1989).

[11] W. B. Pennebaker and J. L. Mitchell, "An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder," *IBM J. Res. Devel.* 32 717–726 (1988).

[12] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard* (Van Nostrand Reinhold, New York, 1993).

[13] R. F. Rice, "Some practical universal noiseless coding techniques," *Technical Report 79-22* (Jet Propulsion Laboratory, California Institute of Technology, Pasadena, 1979).

[14] J. J. Rissanen and G. G. Langdon, "Universal modeling and coding," *IEEE Trans. on Inf. Theory* **27** 12–22 (1981).

[15] A. Said and W. A. Pearlman, "An image multiresolution representation for lossless and lossy compression," *IEEE Trans. Image Process.* **5** 1303–1310 (1996).

[16] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Process.* **9** 1158–1170 (2000).

[17] S. Urban, "Compression results — lossless, lossy ±1, lossy ±3," ISO Working Document ISO/IEC JTC1/SC29/WG1 N281 (1995).

[18] M. Vetterli and J. Kovacevic, *Wavelet and Subband Coding* (Prentice-Hall, Englewood Cliffs, New Jersey, 1995).

[19] M. J. Weinberger, J. Rissanen, and R. B. Arps, "Applications of universal context modeling to lossless compression of gray-scale images," *IEEE Trans. Image Process.* **5** 575–586 (1996).

[20] M. J. Weinberger, G. Seroussi, and G. Sapiro, "LOCO-I: A low complexity context-based lossless image compression algorithm," in *Proc. IEEE Data Compression Conference* (IEEE, New York, 1996) pp. 140–149.

[21] M. J. Weinberger, G. Seroussi, and G. Sapiro, "LOCO-I: A low complexity lossless image compression algorithm," ISO Working Document ISO/IEC JTC1/SC29/WG1 N203 (1995).

[22] X. Wu and N.D. Memon, "Context-based adaptive lossless image coding," *IEEE Trans. Comm.* **45** 437–444 (1997).

[23] A. Zandi, J. D. Allen, E. L. Schwartz, and M. Boliek, "CREW: compression by reversible embedded wavelets," in *Proc. Data Compression Conference* (IEEE, New York, 1995), pp. 212–221.