New Methods for Lossless Image Compression Using Arithmetic Coding

Paul G. Howard and *Jeffrey Scott Vitter*

Technical Report No. CS–91–47
August 1991

New Methods for Lossless Image Compression Using Arithmetic Coding*

Paul G. Howard† *Jeffrey Scott Vitter*‡

Department of Computer Science
Brown University
Providence, R. I. 02912–1910

Abstract

We give a new paradigm for lossless image compression, with four modular components: pixel sequence, prediction, error modeling, and coding. We present two new methods (called *MLP* and *PPPM*) for lossless compression, both involving linear prediction, modeling prediction errors by estimating the variance of a Laplace distribution, and coding using arithmetic coding applied to precomputed distributions. The MLP method is both progressive and parallelizable. We give results showing that our methods perform significantly better than other currently used methods for lossless compression of high resolution images, including the proposed JPEG standard. We express our results both in terms of the compression ratio and in terms of a useful new measure of compression efficiency, which we call *compression gain.*

Index terms: data compression, adaptive modeling, image processing, arithmetic coding, predictive coding.

---

*A shortened version of this paper appeared in the 1991 IEEE Data Compression Conference.

†Support was provided in part by NASA Graduate Student Researchers Program grant NGT–50420 and by a National Science Foundation Presidential Young Investigators Award grant with matching funds from IBM. Additional support was provided by a Universities Space Research Association/CESDIS associate membership.

‡Support was provided in part by a National Science Foundation Presidential Young Investigator Award grant with matching funds from IBM, by NSF grant CCR–9007851, by Army Research Office grant DAAL03–91–G–0035, and by the Office of Naval Research and the Defense Advanced Research Projects Agency under contracts N00014–83–K–0146 and ARPA Order No. 6320, amendment 1. Additional support was provided by a Universities Space Research Association/CESDIS associate membership.

1 Introduction

Lossless compression of high resolution images presents a unique set of challenges. Considerable research has already been done on lossless compression of text [2,3,11, 23,24]; all good methods found to date involve some form of moderately high-order exact string matching. However, we find that techniques for text compression are not effective for lossless image compression, for two reasons. First, images are two-dimensional, so the contexts are more complicated than for one-dimensional strings. Second and more important, images are typically quantized analog data, and the exact matches needed for high-order modeling are only rarely found in the data.

For lossless image compression, a better plan is to find and encode as much of the image structure of the data as possible, and then to efficiently encode the unstructured, noisy residual. We do this in four steps: we *choose a sequence in which to process the pixels*, we *predict the value* of each pixel, we *model the error* of the prediction, and we *encode the error* of the prediction. The predictions correspond to lossy compression; the error encoding makes it lossless.

We present two new compression methods that clearly separate the four steps of pixel sequence, image modeling, error modeling, and error coding. Compression is completely lossless, and we obtain excellent compression ratios, typically 2 : 1 or 3 : 1, using a modest amount of memory. The achieved compression is noticeably better than that obtained by currently-used lossless image encoders, including the proposed JPEG standard.

The first of our methods, the *multi-level progressive* method (MLP), has the additional advantages that it is *progressive* and can be parallelized. By "progressive," we mean that the image is encoded in *levels*; the first level corresponds to a very highly compressed and very lossy encoding, and each successive level provides more detail, until ultimately the encoding is completely lossless. Truncating the encoded file will result in lossy compression of the entire image, rather than exact compression of part of the image. A progressive method allows an end user to browse an image without having to decode much of the encoded data; if more detail is desired, the image can be successively refined as more of the encoded data is decoded. An excellent survey of progressive techniques appears in [20].

In both of our new methods, we obtain lossless image compression by iterating the four-step coding process. We choose a pixel p (with actual value A_p); we make a prediction P_p of the pixel's value and compute the prediction error $\Delta_p = A_p - P_p$; then we model the prediction error by estimating its variance σ_p^2; finally we encode Δ_p using the estimated variance. We can then use the actual value for predicting other pixels, since the decoder can reconstruct the actual value from the prediction and the coded difference.

Most of the output code length comes from the error encoding. We form a probabilistic model for the errors and use arithmetic coding [22] to encode the errors efficiently with respect to the model. For good compression we want the model to

assign as high a probability as possible to the prediction error that actually occurs at each pixel. To obtain this, first we need a good predictor, one whose errors have a small variance so that only a few error values are likely and the probability of each is high. In addition, we need a good model for the errors, so that the arithmetic coder is using realistic probabilities. Prediction errors for images are usually distributed according to a Laplace distribution with zero mean, so to obtain a realistic model we need a good estimate of the variance of the errors. Too high a variance allocates too much probability (and code space) to large, unlikely errors, while too low an estimate assigns too little probability to errors that actually occur. For each pixel we estimate the variance and choose one of a number of precomputed Laplace distributions. We describe the error coding process in detail in Section 2.

In both our methods the prediction is a linear combination of the values of pixels whose values are already known. Our methods differ from each other in the sequence in which pixels are coded, the specific prediction function, and the method of estimating variances. In Section 3 we describe our *multi-level progressive* method (MLP). It uses a many-point interpolation function for prediction; a number of different methods are available for variance estimation. In Section 4, we describe our *prediction by partial precision matching* method (PPPM), a context-based method with approximate matches. PPPM encodes an image in raster-scan order, predicting with a two-point average; variance estimation of each pixel is based on a context of nearby pixels, allowing for inexact matches in a novel way. PPPM gives slightly better compression than does MLP, but it is not progressive; proposed extensions described later may improve MLP's compression significantly.

Empirical results are given in Section 5 for a variety of high-resolution images, both in terms of the compression ration and in terms of a new measure of compression efficiency, which we call *compression gain*. Both MLP and PPPM achieve significantly better image compression than other methods currently used, including the proposed JPEG standard.

2 Encoding Prediction Errors

Both of our methods encode prediction errors by applying arithmetic coding to a Laplace distribution. In this section we explain our choice of the Laplace distribution and describe the application of arithmetic coding.

2.1 Choice of the Laplace distribution

It has long been known that, for most images, prediction errors can be very closely approximated by a Laplace (or symmetric exponential) distribution [6,9,12,13]. In particular, the distribution of prediction errors is sharply peaked at zero, which is characteristic of the Laplace distribution but not of the normal distribution (see Figure 1).

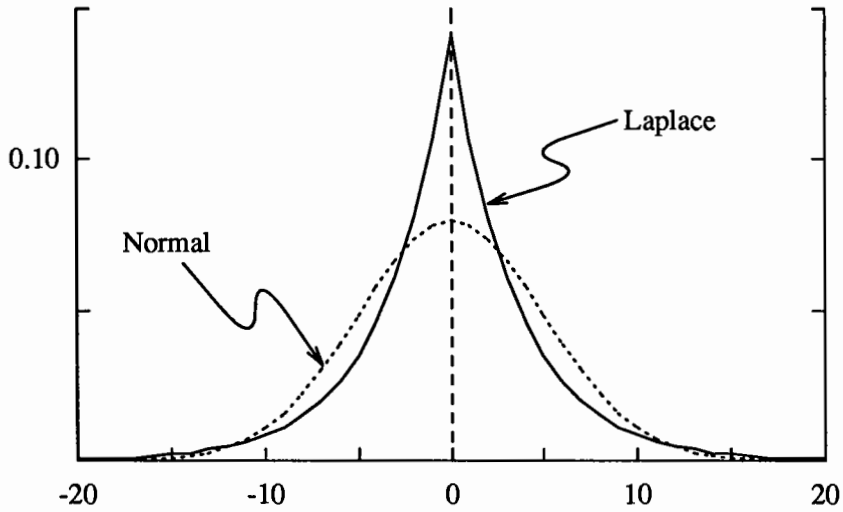

Figure 1: Comparison of Laplace and normal distributions, $\sigma = 5$.

For our prediction methods, we similarly find that prediction errors follow the Laplace distribution very closely. For example, Figure 2 shows the actual frequencies of prediction errors for 131,072 pixels of one of our test images using one of our new methods, as well as the theoretical Laplace distribution with the same variance and zero mean. The fit is almost exact.

The probability density function of the Laplace distribution with mean μ and variance σ^2 is given by

$$f_{\mu,\sigma^2}(x) = \frac{1}{\sqrt{2\sigma^2}} \exp\left(-\sqrt{\frac{2}{\sigma^2}}\,|x-\mu|\right).$$

We assume that $\mu = 0$, and we define the discrete probability mass function by

$$p_{\sigma^2}(k) = \int_{k-1/2}^{k+1/2} \frac{1}{\sqrt{2\sigma^2}} \exp\left(-\sqrt{\frac{2}{\sigma^2}}\,|x|\right) dx. \tag{1}$$

We can use the usual entropy formula to compute the average code length H (in bits per pixel) required to optimally encode a sequence of random variates taken from a Laplace distribution with mean 0 and variance σ^2:

$$H = \sum_k -p_{\sigma^2}(k)\,\log_2 p_{\sigma^2}(k). \tag{2}$$

2.2 Description of the error coding procedure

If we use probabilities computed from $p_{\sigma^2}(\cdot)$ to encode random variates from a Laplace distribution with variance $\sigma_1^2 \neq \sigma^2$, the average code length will be longer than the

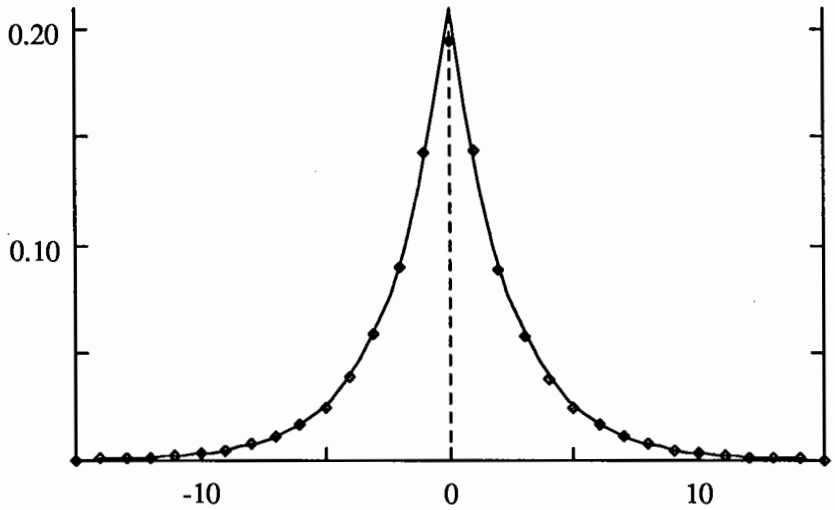

Figure 2: Comparison of continuous Laplace distribution ($\sigma = 3.370$) and the actual distribution of prediction errors (shown by small diamonds) found during the last level of application of method MLP to Band 4 of the Donaldsonville data set. The actual values are virtually indistinguishable from the theoretical values, except when the prediction error is 0; most of the difference in this case is because the curve is the continuous Laplace distribution, not the discrete distribution.

ideal code length, but if the variances are close, the extra code length will be small. The actual average code length $\overline{\ell}$ obtained by using a discrete Laplace distribution with mean 0 and variance σ^2 to encode random variates from a Laplace distribution with mean 0 and variance σ_1^2 is given by

$$\overline{\ell} = \sum_k -p_{\sigma_1^2}(k) \log_2 p_{\sigma^2}(k). \tag{3}$$

By appropriate coder design using Equations (2) and (3), we can place a limit on the average extra code length.

2.2.1 Precomputation of the distributions

We wish to obtain probabilities to pass to the arithmetic coder without excessive computation. To this end, we select a closely-spaced set of variances, with the idea of encoding a prediction error by using the closest variance from the set. We use Equation (1) to compute the Laplace distribution probabilities corresponding to each variance, and build these variances and distributions into both the encoder and the decoder. By careful choice of the set of variances, we can guarantee that the average extra code length due to using an approximate variance will always be less than any given amount. Only 37 variances and distributions are needed to ensure that the coding loss caused by the approximation is less than 0.005 bit per pixel, a barely measurable quantity. The variances and variance ranges are listed in Table 1.

Variance Range	Variance Used	Variance Range	Variance Used	Variance Range	Variance Used
0.005–0.023	0.016	2.882– 4.053	3.422	165.814– 232.441	195.569
0.023–0.043	0.033	4.053– 5.693	4.809	232.441– 326.578	273.929
0.043–0.070	0.056	5.693– 7.973	6.747	326.578– 459.143	384.722
0.070–0.108	0.088	7.973– 11.170	9.443	459.143– 645.989	540.225
0.108–0.162	0.133	11.170– 15.627	13.219	645.989– 910.442	759.147
0.162–0.239	0.198	15.627– 21.874	18.488	910.442– 1285.348	1068.752
0.239–0.348	0.290	21.874– 30.635	25.875	1285.348– 1816.634	1506.524
0.348–0.502	0.419	30.635– 42.911	36.235	1816.634– 2574.021	2125.419
0.502–0.718	0.602	42.911– 60.123	50.715	2574.021– 3663.589	3007.133
0.718–1.023	0.859	60.123– 84.237	71.021	3663.589– 5224.801	4267.734
1.023–1.450	1.221	84.237–118.157	99.506	5224.801– 7247.452	6070.918
1.450–2.046	**1.726**	118.157–165.814	139.489	7247.452–10195.990	8550.934
2.046–2.882	2.433				

Table 1: Laplace distribution variances to guarantee coding loss of less than 0.005 bit per pixel.

We use this set of 37 variances and distributions for the remainder of this paper.

Example: To encode a prediction error when we estimate that $\sigma^2 = 2$, we use the 12th line of Table 1 (highlighted in bold face), since $1.450 < 2 < 2.046$; thus we use the precomputed Laplace distribution with $\sigma^2 = 1.726$. The optimal average code length for Laplace random variates with $\sigma^2 = 2$ is 2.484 bits per sample; by using the discrete Laplace distribution with $\sigma^2 = 1.726$ we achieve an average code length of 2.488 bits per sample, within just 0.004 bits per sample of the optimal length. □

2.2.2 Coding the error

Once we have prepared the probability distributions, arithmetic coding allows us to do the coding without difficulty. Given any discrete probability distribution, we can use arithmetic coding to encode random variates from that distribution with an average code length almost exactly equal to the entropy of the distribution, $\sum_k -p_k \log_2 p_k$. The mechanism of arithmetic coding is straightforward [22], and the extra code length introduced in the coding process is provably small [7]. Unlike Huffman coding [8,21], arithmetic coding performs well even when one of the probabilities is close to 1.

To encode a prediction error Δ_p (presumed to be a random variate from a Laplace distribution with zero mean and a given estimated variance), we select the "nearest" distribution according to Table 1, that is, the one that minimizes the average extra

code length. We then simply use arithmetic coding to encode Δ_p according to the distribution selected.

3 MLP, a Multi-Level Progressive Method

In this section we describe MLP, a new multi-level progressive method for image coding. The prediction step can be done completely in parallel by the encoder, and with a high degree of parallelism by the decoder. This is a practical scheme; it uses memory proportional to the size of the image.

We encode the file in a series of *levels*, each level including twice as many pixels as the previous one. As we begin to encode each level, the pixels with known values are on the lattice points of a square grid. Using only the values of pixels on the lattice points, we predict the value of the pixel at the midpoint of each grid square, and encode the prediction error. (The decoder can make the same prediction, and hence it can use the decoded error to reconstruct the original value.) After all pixels in the level have been encoded, the set of known pixels form a checkerboard pattern; by rotating and scaling the coordinate system we obtain a new square grid of pixels with known values, with the distance between adjacent pixels only $1/\sqrt{2}$ as much as before.

Clearly the method is progressive: each level gives us the value of pixels selected uniformly from the entire image. For example, just before encoding the last level, we know the values of half the pixels of the image, in a checkerboard pattern over the whole image. If we stopped the encoding at that point, the decoder could compute the known pixels and use the prediction function to interpolate the remaining pixels. In practice, even skipping the last two levels (3/4 of the pixels in the image) leaves us with an image virtually indistinguishable from the original.

The time-consuming step in this method is prediction (not arithmetic coding), and the predictions are easy to parallelize. The set of predicting pixels for a given pixel is fixed at the beginning of coding, so for encoding we could even use one processor per pixel to predict in constant parallel time. Since the predictions made during decoding depend on values obtained at earlier levels, only the pixels at a given level can be predicted in parallel; the parallel prediction time for decoding an $n \times n$ image is still small, proportional to the number of levels, $2 \log_2 n$.

Precursors of MLP, which use much simpler predictors and less sophisticated variance estimation, are developed in [5,10,19]. A rotating coordinate system appears in [4,14].

3.1 Description of the MLP algorithm

The encoding algorithm proceeds as follows, for each successive level L:

1. For every pixel $p \in L$, we make a prediction P_p, and compute the prediction error $\Delta_p = A_p - P_p$.

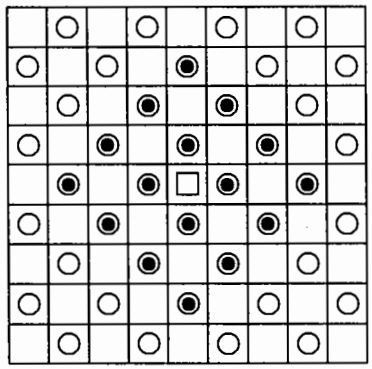

Figure 3: MLP last level prediction neighborhood. Before MLP processes its last level, the pixel values at the circled points (marked ○ and ◉) are known. The pixels indicated by ◉ are used to predict the value of the pixel at the center, marked □. All unmarked pixels will also be predicted during the last level.

0.0039	−0.0351	−0.0351	0.0039
−0.0351	0.3164	0.3164	−0.0351
−0.0351	0.3164	0.3164	−0.0351
0.0039	−0.0351	−0.0351	0.0039

Table 2: Coefficients used in MLP for 16-point midpoint polynomial interpolation.

2. For every pixel $p \in L$, we estimate the variance σ_p^2 to be used in encoding Δ_p (or we compute the variance and encode it).

3. For every pixel $p \in L$, we encode Δ_p as described in Section 2.

4. We do housekeeping to rotate and scale the grid.

Decoding is very similar. We now examine in detail the steps specific to MLP.

3.1.1 Prediction

To predict the value of a pixel, we use a linear combination of the values of a nearby group of pixels coded at earlier levels. A 4×4 array of pixels (see Figure 3) works well in practice, giving better compression and speed than 6×6 or 8×8 and better compression than 2×2. The coefficients applied to the predicting pixels are selected so that they do midpoint interpolation. We have tried polynomial and Fourier interpolation; in practice, polynomial interpolation works slightly better. Approximate values of the coefficients are given in Table 2.

3.1.2 Variance estimation

There is considerable choice in the method of variance estimation. In theory, for each pixel we can find the distribution that encodes the pixel's error as compactly as possible. For the Laplace distribution with mean 0, we find (by solving the equation $\partial p_{\sigma^2}(x)/\partial(\sigma^2) = 0$) that the optimal variance is $\sigma^2 = 2x^2$. Of course, this method is not practical because of the large overhead of encoding the optimal variance for each pixel. However, by optimally encoding the errors and *ignoring* the cost of encoding the variances, we can obtain a reasonable lower bound on the attainable code length. This lower bound applies to MLP for a given prediction function and for a particular image file, and is based on the assumption that the errors follow the Laplace distribution, but it does give a good indication of the compressibility of a particular image.

A more practical method of encoding the variances is to compute and encode the variance of all the errors at a given level; the overhead is minimal, only $\log_2 37 = 5.21$ bits per level. A refinement of this approach is to divide each level into blocks of $k \times k$ pixels and compute and encode the variance of the errors of the pixels in each block. The overhead increases with the number of blocks, but for medium-sized values of k (say $k = 16$) the net code length is often reduced because of the more realistic error model.

Since we expect that predictions made at a given pixel will be about as good as those made at nearby pixels, we can use the errors made in predicting those nearby pixels to estimate the variance of the prediction errors at a given pixel. This method has considerable promise for yielding substantially improved compression because of its potential for accurate variance estimation with no overhead in coding the variances. We are continuing investigation of it.

3.1.3 Housekeeping

After all the pixels in a level have been coded, we rotate the coordinate system by 45 degrees and scale it by a factor of $1/\sqrt{2}$. In practice we do this by alternating between diagonal and axis-parallel coordinate systems, reducing the step size at each level.

It is necessary to deal with startup and edge effects, when the points that should be used as predictors are not present in the image. Because of the small number of pixels involved, these effects have little practical significance. In our implementation we simply take the prediction coefficients of missing points to be zero, and then renormalize the remaining coefficients.

4 PPPM, Prediction by Partial Precision Matching

The success of high-order, context-based methods for text compression naturally leads us to consider similar ideas for lossless image compression. We present a method

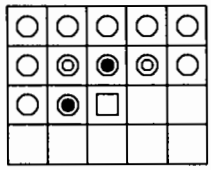

Figure 4: PPPM prediction and variance estimation contexts. When PPPM, proceeding in raster scan order, reaches the pixel marked □, the pixel values at all the circled points (marked ○, ◉, and ◎) are known. The intensity values of two pixels indicated by ◉ are used to predict the value of the □ pixel, and the prediction errors at the four pixels marked ◉ and ◎ are used to estimate the variance to be used for encoding the prediction error.

similar in spirit to the prediction by partial matching method (PPM) of Cleary and Witten [2], but with a significant difference. PPM encodes each input symbol using the longest context in which the symbol has already occurred, up to a specified maximum length. For image compression this method as presented is not completely satisfactory: the main problem is that the exactly-repeated strings that make text compression possible are not present in images. Our solution is a new way of detecting approximate two-dimensional matches, eminently suited for image compression. We call this method *prediction by partial precision matching,* or PPPM[1].

PPPM encodes an image in raster-scan order. We again separate the coding process into prediction, variance estimation, and error coding. For prediction we use a simple linear combination of a few nearby pixels. For variance estimation we attempt to make use of previous occurrences of the current context; however, instead of trying to maximize the *size* of the matching context as in PPM, we use a small, fixed-size context and focus on the *closeness* of the match. If we fail to find enough previous occurrences of the exact current context, we relax the exactness constraint, ignoring the least significant bit of each of the context pixels. If we still cannot find a match, we ignore the next significant bits until we finally find an approximate "context" that has occurred enough times; we use the statistics of that context to estimate the variance.

The prediction context and the variance estimation contexts need not be the same. In our implementation (see Figure 4) we use two pixels for prediction, the one immediately above and the one immediately to the left of the pixel being coded; the context consists of the intensity values of the predicting pixels. For variance estimation we use a four-pixel neighborhood consisting of the three pixels above and the one immediately to the right of the pixel being coded; the context consists of the errors previously made in predicting the values of the pixels.

It is possible to use four or even eight *surrounding* pixels for prediction. If this is done for each pixel, we can reconstruct the image exactly if we are given the

[1]The version of this method presented at the 1991 IEEE Data Compression Conference was called PPMI.

raw intensity value of just one pixel. However in practice this modification requires significantly more computation and achieves only marginally better compression.

4.1 Description of the algorithm

The encoding algorithm proceeds in raster-scan order. We encode each pixel p as follows:

1. We make a prediction P_p using the prediction context, and compute the prediction error $\Delta_p = A_p - P_p$.
2. We repeat the following steps as many times as necessary to estimate the variance:
   (a) We construct a key, based on the prediction errors of the pixels in the variance estimation context.
   (b) We search (using a hash table) for previous occurrences of the key.
   (c) If we find enough previous occurrences, we use the error statistics of those occurrences to obtain an estimated mean μ_p and variance σ_p^2 for the prediction error of the pixel; we leave the variance-estimation loop.
   (d) If we do not find enough previous occurrences, we drop one low-order bit from the prediction errors of the pixels in the variance estimation context.
3. We encode $\Delta_p + \mu_p$ as described in Section 2.
4. We update the statistics for one or more of the contexts just examined.

Decoding is very similar. We now examine in more detail the steps specific to PPPM.

4.1.1 Prediction

In our current implementation, we predict a pixel's intensity by the rounded arithmetic mean of the intensities of the two pixels in the prediction context. At the top and left edges we simply use fewer pixels. Our prediction in this method is admittedly very simple; a more sophisticated extrapolation function would give reduced error variances and improved compression. It does not suffice, however, simply to average more nearby pixels: using the average of the four pixels in the ⊞-shaped neighborhood gives worse compression than the two-pixel neighborhood that we use.

4.1.2 Variance estimation

As noted above, we construct "contexts" of decreasing precision based on the previously computed prediction errors of the four pixels in the variance estimation context.

(We could use intensity values, but in practice using error values works slightly better.) Since we are using errors rather than intensities, we have to deal with negative values; we use a signed-magnitude representation, the sign being associated with the most significant nonzero bit.

Example: Using 8 "bits," we represent -5 and $+10$ as follows:

-5	0	0	0	0	0	$^{-}1$	0	1
$+10$	0	0	0	0	$^{+}1$	0	1	0

□

We follow the procedure described above to find a context that has occurred enough times to allow a satisfactory estimate of the variance of prediction errors made in that context. In practice, a context's statistics are unreliable until the context has occurred 10 or 15 times; we set the default threshold at 12. For the selected context we compute the mean μ_p and the variance σ_p^2 of its previous occurrences for use in the coding step.

4.1.3 Encoding the error

We adjust Δ_p by adding μ_p to reflect the context's mean prediction error, and we encode the result as described in Section 2.

4.1.4 Updating the statistics

Each pixel occurs in a number of variance estimation "contexts" of different precisions. Instead of updating each of them, we adopt a "lazy update" rule that saves time and gives better variance estimates. After coding each pixel we update the statistics for at most two contexts, the one used for coding and, if possible, the one with one additional bit of precision.

5 Comparative Results

We now show that our methods give excellent compression, considerably better than that obtained by other lossless methods now used. For the 21 images in our test data (described below), we obtain the following compression ratios:

Method	Range	Median
MLP	1.77:1 to 9.57:1	2.34:1
PPPM	1.79:1 to 8.58:1	2.40:1

Of course, these figures are highly dependent on the images tested. To allow more meaningful comparisons, we introduce a new compression measure, more general than the simple compression ratio, called *compression gain.*

Measure	Formula	Example
Bits per input symbol (also called bit rate or average code length)	$\frac{8L}{t}$	2.92 bits per pixel
Compression ratio	$\frac{t}{L}$	2.74 : 1
Relative compressed size	$\frac{L}{t}$	0.36 or 36%
Relative compression	$\frac{t-L}{t}$	0.64 or 64%
Compression gain	$10\log_{10}\frac{H_0/8}{L/t}$	+2.15 dB
Reference gain	$10\log_{10}\frac{1}{H_0/8}$	+2.23 dB

Table 3: Measures of compression efficiency. Our new measure, given on the second-to-last line, is *compression gain*. The original file is t bytes long, the compressed file is L bytes long, and the zero-order entropy of the original file is H_0 bits per pixel. The examples in the rightmost column are based on using the PPPM method on Band 3 of the Donaldsonville data set, a typical image. The image consists of 262,144 8-bit pixels, and the compressed file is 95,587 bytes long. This image has a zero-order entropy of 4.784 bits per pixel, which corresponds to a *reference gain* of 2.23 dB.

5.1 Compression gain, a new measure of compression efficiency

There is no single accepted method for reporting and comparing compression results. Several measures are used in the literature (see Table 3). All of the existing measures have two major shortcomings from a theoretical point of view. First, they all express compression relative to the original file length. In practical applications this can be useful, but any such measure fails to discriminate between the effectiveness of the compression method and the compressibility of the files being compressed. Second, none of the measures are additive: we cannot simply add together modeling and coding effects, or the effects of multiple cascaded compressors.

We introduce a new measure, which we call *compression gain*, that removes both of these difficulties. In accordance with common engineering usage, we express the gain in decibels (dB) with respect to a theoretical standard. We choose the zero-order

entropy of the file as the reference value and define the compression gain to be

$$10 \log_{10} \frac{H_0/8}{L/t}, \tag{4}$$

where t is the length of the input file in bytes, L is the total code length of the compressed file in bytes, and H_0 is the zero-order entropy of the file in bits per input symbol. As an example, a gain of 3.01 dB means that the file is compressed to half the size of the output of a theoretical zero-order entropy compressor. For completeness we also give the *reference gain* of the zero-order entropy with respect to the original file length, namely

$$10 \log_{10} \frac{1}{H_0/8}. \tag{5}$$

The zero-order entropy of the file is a natural choice for the reference value:

- It is indicative of the inherent compressibility of the file.
- It is easily computed, without compressing the file.
- It is a property of all files, and does not directly depend on the word size of the data representation.
- Compression close to the zero-order entropy can be obtained by relatively simple methods, such as dynamic arithmetic coding [7,22].

The logarithms in Formulæ 4 and 5 gives us additivity, so we can simply add the compression gain and reference gain to obtain the total gain, $10 \log_{10}(t/L)$, if desired. Coding effects, often expressed as percentages of code length, can now be given as losses that can simply be subtracted from the compression gain of the modeling method. In this form we can clearly separate coding and modeling, and we can see how tiny the coding effects are.

The *gain/loss* terminology is natural: larger numbers mean better compression. As an additional advantage, we find that for our test images the difference in compression gain between two methods is often approximately constant from file to file.

5.2 Experimental results

We now briefly describe the other compression programs against which we compare our methods to evaluate their effectiveness in practice. The CCITT/ISO Joint Photographic Experts Group (JPEG) has recently developed a proposed standard for image compression [1] that addresses both lossless and lossy compression. The JPEG lossless mode is based on prediction by one, two, or three points, followed by Huffman coding or arithmetic coding; encoding proceeds in raster scan order. The standard gives some latitude to implementors; we report results from an implementation based

Washington, D.C. Donaldsonville, Louisiana Ridgely, Maryland

Figure 5: A 256 × 256 section of Band 4 of each data set.

on two-point prediction with arithmetic coding. For all except three highly compressible images, two-point prediction provides 0.2 to 0.4 dB more compression gain than three-point prediction.

There are three other reasonable choices for our comparisons: the UNIX *compress* program [17], a high-order text compression program, and a DPCM image compression program. The widely available *compress* program is the *de facto* standard for text compression, and in fact gives some compression on images, mostly attributable to its ability to capture the context-independent entropy of any file. High-order text compression programs outperform *compress* on text and on images; PPMC [11] is one of the best available programs in this class, so we include it in our comparisons. PPMC does best on most images with a first or second order model; we use a first order model since it is faster. We also include a simple, fast, DPCM program [16], based on one-point prediction and output of either 4 or 12 bits depending on the magnitude of the prediction error; it is representative of the class of lossless image compression programs designed for speed.

Our test data consists of three seven-band Landsat Thematic Mapper data sets consisting of 8-bit grayscale images, over Washington, D.C., Donaldsonville, Louisiana (90 kilometers west of New Orleans), and Ridgely, Maryland (70 kilometers southeast of Baltimore). The Washington and Donaldsonville images are 512 × 512 pixels; the Ridgely images are 368 × 468 pixels. Portions of three of the images, contrast enhanced, are shown in Figure 5. Each data set contains one highly compressible image (compressible to better than 8 : 1) and six "normal" images.

In Tables 4 through 7 and Figures 6 and 7, we denote the various compression methods by the following abbreviations:

Abbreviation	Compression Method
H_0	Reference gain (zero-order entropy compared to original)
comp	UNIX *compress*
PPMC	PPMC, using contexts of order 1
DPCM	Spinellis's "Delta Modulation" program
JPEG	JPEG lossless mode
PPPM	Our PPPM method
MLP	Our MLP method, using a 4×4 prediction context and a single variance for each level
MLP_B	Our MLP method, using a 4×4 prediction context and a variance encoded for each 16×16 block
MLP_opt	The bound based on using optimal variance estimation, as described in Section 3.1.2

Each figure in Table 4 is a compression ratio, defined as the original file size divided by the compressed file size. The same data are presented in Table 5 as compression gains, computed according to Formula (4) and expressed in dB. (The reference gain is computed with respect to the original file length, according to Formula (5).) For each file, the best compression is indicated in boldface. The MLP_opt figures are italicized to indicate that they are bounds instead of attainable compression. The compression ratios are depicted graphically in Figure 6, and the compression gains in Figure 7. For the Ridgely data, we can make additional comparisons (see Tables 6 and 7) with three other progressive methods: a simple block averaging method, a method based on quadtrees [15], and a method based on identifying regions containing similar points [18]. The values for these three methods are based on results given in [18], where the methods are described in detail.

5.3 Discussion of the results

For all of the 21 images, the maximum compression is obtained by one of our methods. For the 18 "normal" images, PPPM and MLP_B give the best compression, PPPM being slightly better for all of the images. MLP (unblocked) and the JPEG implementation follow in the ranking, MLP outperforming JPEG on all but two of the "normal" images. For the three highly compressible images, MLP compresses slightly better than MLP_B; both are noticeably better than the JPEG implementation, which in turn is better than PPPM. PPMC is consistently the best of the remaining methods.

The difference of about 1 dB between our methods and the MLP_opt bounds indicates that attempting to improve methods of variance estimation should be a fruitful area of further research. We expect that the idea discussed at the end of Section 3.1.2, estimating the variance of a pixel by using the variances of neighboring pixels, will

Washington, D.C.

Band	*comp*	PPMC	DPCM	JPEG	PPPM	MLP	MLP_B	MLP_{opt}
1	1.70	1.97	1.74	2.07	**2.18**	2.16	2.17	*2.67*
2	2.21	2.53	1.93	2.67	**2.79**	2.74	2.76	*3.63*
3	1.92	2.20	1.83	2.28	**2.42**	2.37	2.39	*3.02*
4	1.46	1.71	1.46	1.81	**1.93**	1.90	1.90	*2.27*
5	1.34	1.58	1.30	1.68	**1.79**	1.76	1.77	*2.09*
6	5.36	6.72	2.00	7.92	7.31	**8.66**	8.56	*16.15*
7	1.70	1.99	1.75	2.10	**2.21**	2.20	2.20	*2.70*
All	1.86	2.17	1.68	2.30	**2.42**	2.40	2.41	*3.00*

Donaldsonville, Louisiana

Band	*comp*	PPMC	DPCM	JPEG	PPPM	MLP	MLP_B	MLP_{opt}
1	1.79	2.12	1.79	2.26	**2.40**	2.27	2.34	*2.97*
2	2.36	2.80	1.91	3.07	**3.19**	2.92	3.10	*4.31*
3	1.99	2.36	1.77	2.58	**2.74**	2.45	2.59	*3.47*
4	1.34	1.66	1.41	1.85	**2.01**	1.89	1.92	*2.36*
5	1.34	1.63	1.37	1.82	**1.97**	1.83	1.87	*2.31*
6	6.14	7.77	2.00	9.25	8.58	**9.61**	9.55	*18.76*
7	1.65	1.99	1.67	2.17	**2.31**	2.21	2.24	*2.81*
All	1.87	2.26	1.67	2.49	**2.64**	2.48	2.55	*3.30*

Ridgely, Maryland

Band	*comp*	PPMC	DPCM	JPEG	PPPM	MLP	MLP_B	MLP_{opt}
1	1.79	2.14	1.85	2.28	**2.41**	2.40	2.39	*2.98*
2	2.26	2.71	1.96	2.94	**3.06**	3.04	3.04	*4.09*
3	1.86	2.25	1.85	2.45	**2.60**	2.55	2.56	*3.29*
4	1.76	2.08	1.80	2.24	**2.39**	2.32	2.34	*2.98*
5	1.34	1.64	1.45	1.78	**1.92**	1.85	1.86	*2.27*
6	1.58	1.92	1.68	2.08	**2.21**	2.17	2.17	*2.68*
7	5.50	6.99	2.00	7.43	7.02	**8.06**	7.96	*14.64*
All	1.91	2.31	1.78	2.49	**2.63**	2.60	2.60	*3.33*

Table 4: Compression ratios (original file size divided by compressed file size) for three data sets. Each data set consists of seven spectral bands (images); each band is encoded independently of the other bands.

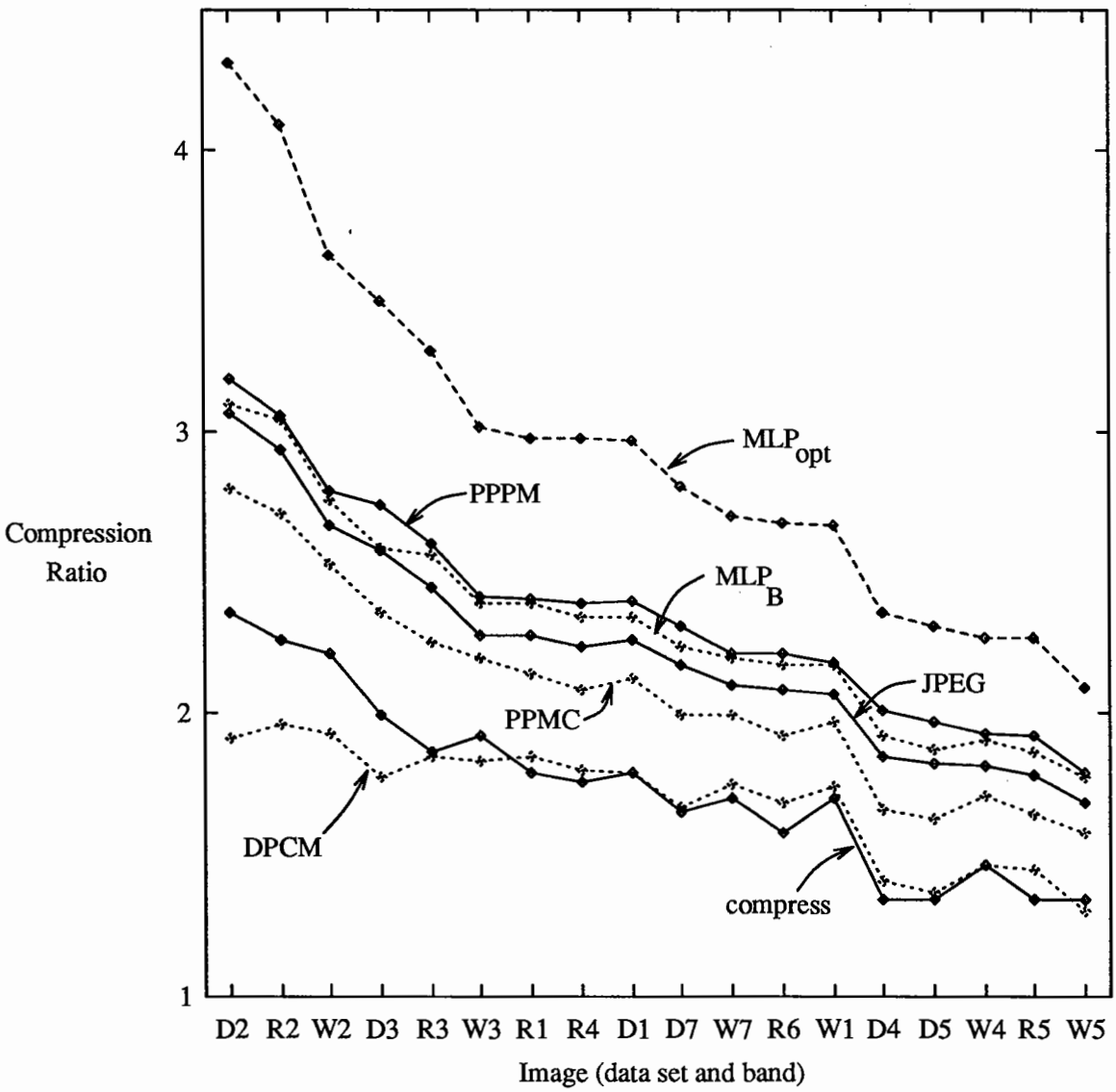

Figure 6: Compression ratios from Table 4. Images are identified by data set (Washington (**W**), Donaldsonville (**D**), or Ridgely (**R**)) and band number. They are arranged in order of decreasing compressibility, measured by the compression ratio of MLP_{opt}. Images **D6**, **W6**, and **R7** are not included: they are all highly compressible, and would appear off the scale. For these three images the performance of PPPM falls below that of MLP_B and JPEG.

Washington, D.C.

Band	H_0	*comp*	PPMC	DPCM	JPEG	PPPM	MLP	MLP_B	MLP_{opt}
1	2.32	−0.03	0.63	0.07	0.84	**1.07**	1.01	1.03	*1.95*
2	3.18	0.27	0.85	−0.34	1.09	**1.28**	1.19	1.23	*2.42*
3	2.69	0.15	0.73	−0.05	0.89	**1.14**	1.05	1.09	*2.12*
4	1.68	−0.04	0.66	−0.04	0.89	**1.17**	1.10	1.11	*1.89*
5	1.49	−0.21	0.50	−0.33	0.77	**1.04**	0.98	0.99	*1.71*
6	3.66	3.63	4.61	−0.65	5.32	4.97	**5.71**	5.66	*8.42*
7	2.26	0.04	0.72	0.18	0.96	**1.18**	1.16	1.17	*2.05*
All	2.41	0.28	0.95	−0.15	1.20	**1.42**	1.39	1.41	*2.37*

Donaldsonville, Louisiana

Band	H_0	*comp*	PPMC	DPCM	JPEG	PPPM	MLP	MLP_B	MLP_{opt}
1	2.19	0.33	1.06	0.35	1.35	**1.60**	1.36	1.50	*2.53*
2	2.83	0.89	1.65	−0.01	2.05	**2.20**	1.83	2.09	*3.51*
3	2.23	0.75	1.50	0.26	1.88	**2.15**	1.65	1.91	*3.17*
4	1.02	0.24	1.18	0.47	1.64	**2.01**	1.74	1.81	*2.70*
5	1.13	0.14	1.00	0.25	1.46	**1.81**	1.48	1.58	*2.50*
6	3.80	4.08	5.11	−0.79	5.86	5.54	**6.03**	6.00	*8.93*
7	1.79	0.39	1.21	0.42	1.57	**1.85**	1.65	1.71	*2.70*
All	2.05	0.66	1.49	0.18	1.90	**2.16**	1.89	2.02	*3.13*

Ridgely, Maryland

Band	H_0	*comp*	PPMC	DPCM	JPEG	PPPM	MLP	MLP_B	MLP_{opt}
1	2.18	0.34	1.13	0.50	1.40	**1.64**	1.62	1.61	*2.57*
2	2.69	0.85	1.64	0.23	1.98	**2.17**	2.14	2.14	*3.42*
3	2.10	0.60	1.42	0.58	1.80	**2.05**	1.97	1.98	*3.08*
4	2.23	0.23	0.95	0.31	1.26	**1.54**	1.42	1.46	*2.51*
5	1.14	0.15	1.01	0.47	1.38	**1.70**	1.54	1.56	*2.43*
6	1.66	0.33	1.17	0.59	1.52	**1.79**	1.71	1.70	*2.63*
7	4.33	3.07	4.12	−1.32	4.38	4.13	**4.73**	4.68	*7.33*
All	2.24	0.57	1.39	0.26	1.72	**1.96**	1.91	1.92	*2.99*

Table 5: Compression gains (in dB) for three data sets. Each data set consists of seven spectral bands (images); each band is encoded independently of the other bands.

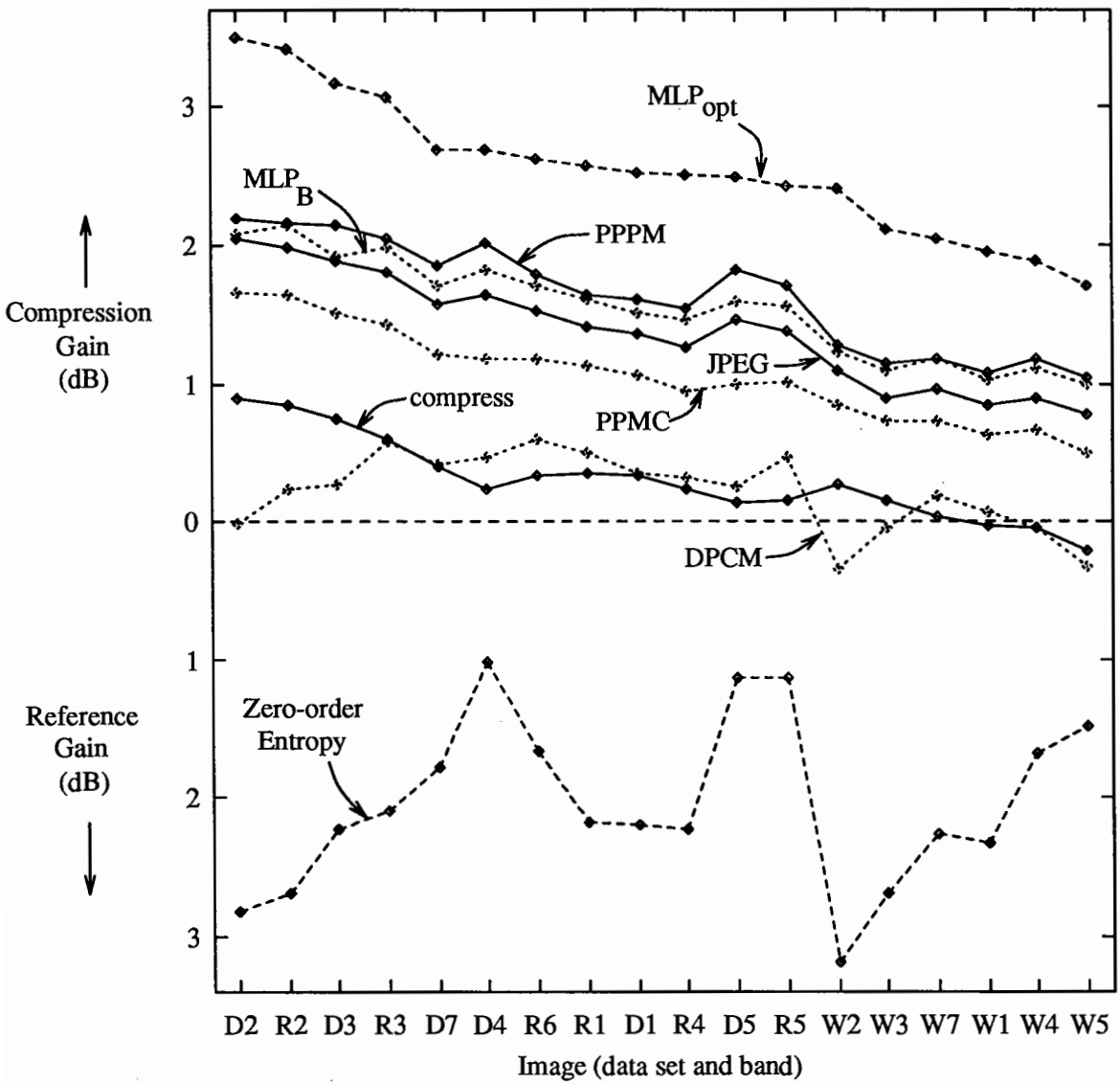

Figure 7: Compression gains from Table 5. Images are identified by data set (Washington (**W**), Donaldsonville (**D**), or Ridgely (**R**)) and band number. They are arranged in order of decreasing compressibility, measured by the gain of MLP$_{opt}$. (This is not the same order as in Figure 6.) Images **D6**, **W6**, and **R7** are not included: they are all highly compressible, and would appear off the scale. The reference gains appear inverted in the lower half of the graph.

Method	Band 5	All Bands
Block Average	1.2*	1.2
Quadtree	1.2	1.6
Region Growing	0.9*	1.6
compress	1.34	1.91
PPMC	1.64	2.31
DPCM	1.45	1.78
JPEG	1.78	2.49
PPPM	**1.92**	**2.63**
MLP	1.85	2.60
MLP_B	1.86	2.60
MLP_{opt}	*2.27*	*3.33*

*estimated

Table 6: Compression ratios for the Ridgely data set, Band 5 separately and all seven bands combined.

Method	Band 5	All Bands
H_0 (Reference gain)	1.14	2.24
Block Average	−0.3*	−1.4
Quadtree	−0.3	−0.2
Region Growing	−1.6*	−0.2
compress	0.15	0.57
PPMC	1.01	1.39
DPCM	0.47	0.26
JPEG	1.38	1.72
PPPM	**1.70**	**1.96**
MLP	1.54	1.91
MLP_B	1.56	1.92
MLP_{opt}	*2.43*	*2.99*

*estimated

Table 7: Compression gains (in dB) for the Ridgely data set, Band 5 separately and all seven bands combined.

give good variance estimation with no overhead; this should considerably improve the performance of MLP.

6 Conclusions

In this paper we identify the four components of a good predictive lossless image compression method: pixel sequence, image modeling and prediction, error modeling, and error coding. We give a practical method for error coding based on arithmetic coding applied to a set of Laplace distributions. We propose two new compression methods, the multi-level progressive method (MLP) and prediction by partial precision matching (PPPM) that address the other three components. Both MLP and PPPM give significantly better compression than do other currently used methods, including the proposed JPEG standard. In addition, MLP is both progressive and parallelizable.

The hierarchical decomposition of MLP, along with a judicious choice of data structures for arithmetic coding, makes MLP amenable to parallel processing. Work is continuing on a massively parallel implementation of MLP. We are also continuing work on mathematical modeling of the prediction errors. Preliminary work indicates that the variance estimates can be improved significantly, further improving the performance of MLP. Additional work is underway to extend our methods to three-dimensional images, multi-spectral data sets, and animations, exploiting the coherence in the extra dimension to get improved compression.

Acknowledgements. We wish to thank James C. Tilton of the NASA Goddard Space Flight Center for supplying the images used for our experiments and for the experimental results that appear in Tables 6 and 7. We also wish to thank Allan R. Moser of E. I. duPont de Nemours and Company (Inc.) for assisting us with experimental evaluation of lossless mode JPEG compression.

References

[1] "Digital Compression and Coding of Continuous-Tone Still Images, Part 1, Requirements and Guidelines," ISO/IEC JTC1 Committee Draft 10918–1, Feb. 1991.

[2] J. G. Cleary and I. H. Witten, "Data Compression Using Adaptive Coding and Partial String Matching," *IEEE Trans. Comm.* COM–32 (Apr. 1984), 396–402.

[3] G. V. Cormack and R. N. Horspool, "Data Compression Using Dynamic Markov Modelling," *Computer Journal* 30 (Dec. 1987), 541–550.

[4] T. Endoh and Y. Yamakazi, "Progressive Coding Scheme for Multilevel Images," *Picture Coding Symp.* (1986), 21–22.

[5] N. Garcia, C. Munoz, and A. Sanz, "Image Compression Based on Hierarchical Coding," *SPIE Image Coding* 594 (1985), 150–157.

[6] A. Habibi, "Comparison of nth-Order DPCM Encoder With Linear Transformations and Block Quantization Techniques," *IEEE Trans. Comm. Tech.* COM–19 (Dec. 1971), 948–956.

[7] P. G. Howard and J. S. Vitter, "Analysis of Arithmetic Coding for Data Compression," in *Proc. Data Compression Conference*, J. A. Storer and J. H. Reif, eds., Snowbird, Utah, Apr. 8–11, 1991, 3–12, invited paper.

[8] D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proceedings of the Institute of Radio Engineers* 40 (1952), 1098–1101.

[9] A. K. Jain, "Image Data Compression: A Review," *Proc. of the IEEE* 69 (Mar. 1981), 349–389.

[10] K. Knowlton, "Progressive Transmission of Gray-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes," *Proc. of the IEEE* 68 (July 1980), 885–896.

[11] A. Moffat, "A Note on the PPM Data Compression Algorithm," Dept. of Computer Science, Univ. of Melbourne, Research Report 88/7, Melbourne, Victoria, Australia, 1988.

[12] A. N. Netravali and J. O. Limb, "Picture Coding: A Review," *Proc. of the IEEE* 68 (Mar. 1980), 366–406.

[13] J. B. O'Neal, "Predictive Quantizing Differential Pulse Code Modulation for the Transmission of Television Signals," *Bell Syst. Tech. J.* 45 (May–June 1966), 689–721.

[14] P. Roos, M. A. Viergever, M. C. A. van Dijke, and J. H. Peters, "Reversible Intraframe Compression of Medical Images," *IEEE Trans. Medical Imaging* 7 (Dec. 1988), 328–336.

[15] H. Samet, "The Quadtree and Related Hierarchical Data Structures," *Comput. Surveys* 16 (1984), 187–260.

[16] D. Spinellis, "*deltac*," computer program, 1989.

[17] S. W. Thomas, J. McKie, S. Davies, K. Turkowski, J. A. Woods, and J. Orost, "*compress*," computer program, 1985.

[18] J. C. Tilton and M. Manohar, "Hierarchical Data Compression: Integrated Browse, Moderate Loss, and Lossless Levels of Data Compression," in *Proc. of International Geoscience and Remote Sensing Symposium*, College Park, MD, May 20–24, 1990.

[19] H. H. Torbey and H. E. Meadows, "System for Lossless Digital Image Compression," *SPIE, Visual Communication and Image Processing* (Nov. 1989), 989–1002.

[20] K. H. Tzou, "Progressive Image Transmission: a Review and Comparsion of Techniques," *Optical Engineering* 26 (July 1987), 581–589.

[21] J. S. Vitter, "Design and Analysis of Dynamic Huffman Codes," *Journal of the ACM* 34 (Oct. 1987), 825–845.

[22] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression," *Comm. ACM* 30 (June 1987), 520–540.

[23] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Inform. Theory* IT–23 (May 1977), 337–343.

[24] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable Rate Coding," *IEEE Trans. Inform. Theory* IT–24 (Sept. 1978), 530–536.