

Determining Optimal Compression Effort For JPEG-LS Images

Mark J. Boyd

Abstract

JPEG-LS encoding uses a low complexity compression concept, taken from LOCO-I, to compress images without loss of resolution. The standard uses prediction based on pixels to the left, above, to the upper-left, and to the upper-right of the current pixel being decoded. The standard then uses Golomb-Rice coding or run-length coding to encode the prediction error.

The JPEG-LS standard has a much lower compression ratio than its lossy counterpart. This makes JPEG-LS undesirable for images immediately presented to the human eye, but very useful for images, such as those collected by satellite imagery, that are likely to be processed after transmission for extraction of otherwise imperceptible detail.

Because the amount of compression achieved is a function of the number of attempts at compressing the image, if an image will be downloaded many times, more effort should be initially expended compressing the image. This paper presents an equation describing the optimal relationship between compression effort and expected required bandwidth for JPEG-LS.

Introduction

M. J. Weinberger, G. Seroussi, and G. Sapiro initially presented the idea of Low Complexity Lossless Compression for Images (LOCO-I) [3]. The JPEG-LS standard¹ uses the LOCO-I idea, with some minor modifications, to encode compressed images. This is outlined in Figure 1, and described in the following section.

Implementation of JPEG-LS Decoding

Causal template.

This JPEG-LS standard [2] differs from the LOCO-I standard in its use of only four nearby previously decoded pixels. Pixel e is excluded, and pixels a and b are transposed. The LOCO-I template, Table 1,

is therefore marginally different from the JPEG-LS causal template, shown in Table 2.

¹a public draft is found at <http://www.jpeg.org/public/jpeglinks.htm>

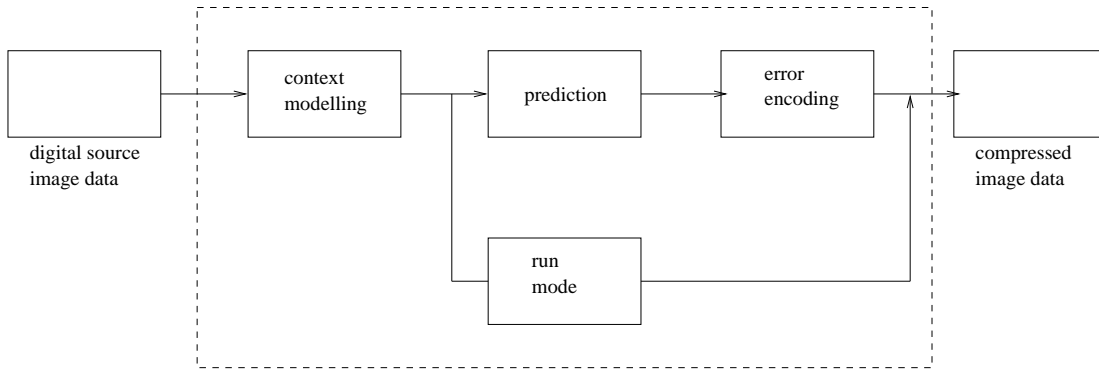


Figure 1: Lossless encoder simplified diagram

	c	a	d	
e	b	x		

Table 1: LOCO-I Causal template

The template determines the actions in the following steps, and was chosen as a compromise between using too many local pixels (and thereby having a great deal of irrelevant information) or not including enough pixels (and thereby foregoing pixels which may have unique predictive relevance).

The previous lossless standard used 8 prediction selection vectors (PSVs) which allowed the encoder to determine which combination of previously predicted pixels produced the most relevant prediction.

Unlike that approach, JPEG-LS uses all of the pixels of the template but allows some limited customization of contexts based on user-customized thresholds and their resulting context mappings.

In a following section I describe this context model, and in the final section on JPEG-LS pitfalls, I describe the limitations imposed by the limiting of user customization of thresholds.

Run length encoding.

The first step in encoding the image is to determine if the current pixel is part of a run. If the reconstructed values of a, b, c, and d are identical, R_x is set to R_a and run mode is started. The case for using a run mode is simple.

	c	b	d	
	a	x		

Table 2: JPEG-LS causal template used for context modelling and prediction

Without bias correction, a geometric probability distribution with $p[0] = .5$ would generate the probability x of a run of zeros of length n

$$x = \frac{1}{2^n}$$

In practice, the smoothness of images and large background scenes make long runs of zeros even more likely since $p[0]$ is often larger than $.5$, and the probability of a 0 is much higher given the previous prediction is 0.

Since even an ideal Huffman encoding requires one bit per 0, and run-length encoding improves this [5], this encoding provides additional compression.

The JPEG-LS standard keeps track of the lengths of runs previously encountered, and uses a Golomb-Rice encoding to encode the value of the most popular length as the shortest sequence. The details of the encoding are described in a later section, the pertinent difference being the mapping to run lengths instead of prediction errors.

Edge detection.

The second step in the encoding procedure is a test to detect vertical or horizontal edges. These occur very commonly in an image and indicate large differences between the nearby pixels and the current pixel. Since an average of the current pixels will always generate an incorrect prediction, while edge detection often produces an exact match (at least for horizontal and vertical edges), edge detection is a useful heuristic.

The implementation is simple. If a horizontal edge is detected, the value at a , (R_a) is predicted. If a vertical edge is detected, the value at b , (R_b) is predicted.

$$R_x = \begin{cases} \min(R_a, R_b) & \text{if } R_c \geq \max(R_a, R_b) \\ \max(R_a, R_b) & \text{if } R_c \leq \min(R_a, R_b) \\ R_a + R_b - R_c & \text{otherwise} \end{cases}$$

If no edge is detected, a third value, $R_a + R_b - R_c$, is generated and used as the current pixel prediction. This value is an extrapolation of the predicted intercept of the magnitude of pixel x with a plane described by the magnitudes of the pixels at a , b and c .

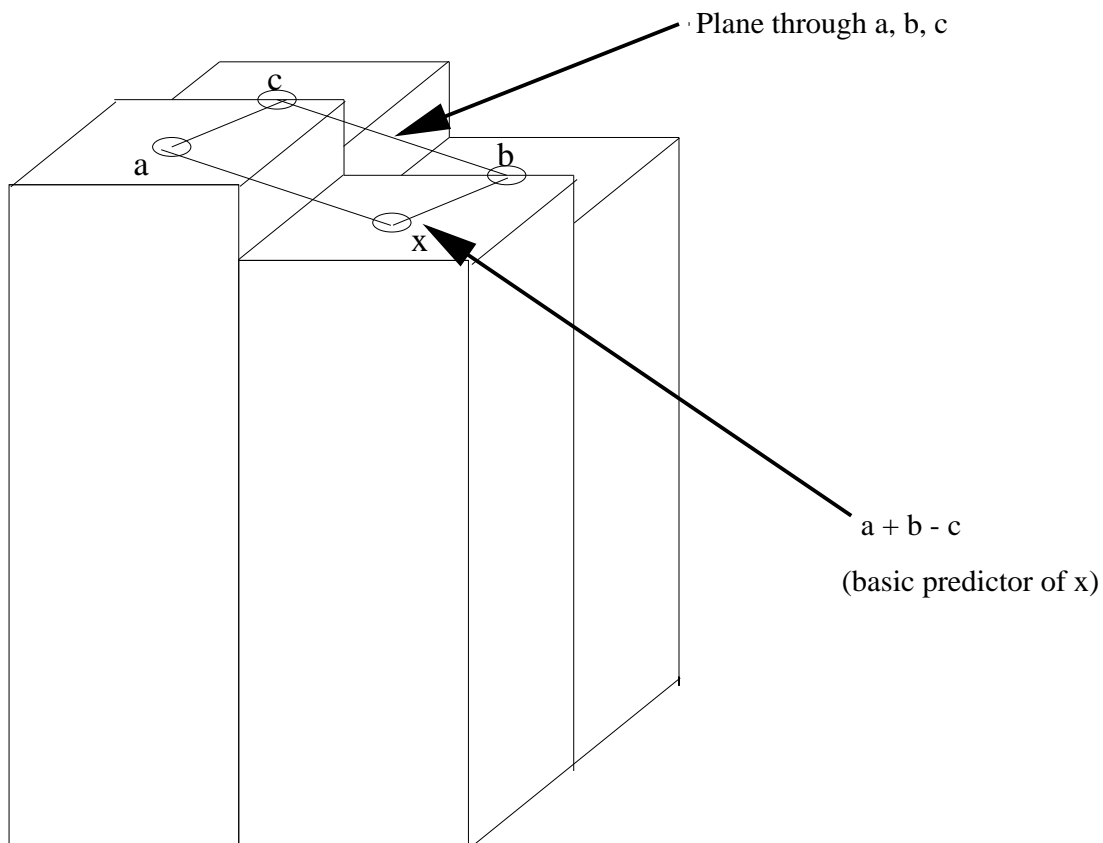


Figure 2: Predicting x by extrapolating a plane a, b, c

Context determination.

If the pixel was not part of a run, we used either edge detection or the plane extrapolation to determine the basic prediction. We now must determine the prediction error. This requires us to first determine the context. Each context is a vector which allows us to access the relevant error statistics in a lookup table.

Determining the current context is a bit complex. It depends on the difference values:

$$\begin{aligned} D1 &= R_d - R_b \\ D2 &= R_b - R_c \\ D3 &= R_c - R_a \end{aligned}$$

as well as the threshold quantization values $T1, T2,$ and $T3,$ which can be set by the encoder or have default values:

T1 = 3
T2 = 7
T3 = 21

With T1, T2, and T3 in hand, we determine the vector (Q1,Q2,Q3) by quantizing each difference, D1, D2, D3:

if (Di ≤ -T3) Qi = -4;
else if (Di ≤ -T2) Qi = -3;
else if (Di ≤ -T1) Qi = -2;
else if (Di < - NEAR) Qi = -1;
else if (Di ≤ NEAR) Qi = 0;
else if (Di < T1) Qi = 1;
else if (Di < T2) Qi = 2;
else if (Di < T3) Qi = 3;
else Qi = 4;²

This gives us 9 x 9 x 9 = 729 possible vectors.³

The vector references a lookup table with the correction value (C) used to offset the bias of the distribution for that context, as well as a decay-rate value (m), both of which are discussed in the following section.

Golomb-Rice code.

A Golomb-Rice coding [1, 5] is now used to decode the prediction error. Golomb-Rice binary encoding is optimal for exponentially decaying (geometric) probability distributions [4] where $m = 2^k$ for some $k \in \text{integers}$. However, most images do not follow this distribution. If we used this distribution centered at 0, we would achieve poor compression.

This is why the decoder instead uses the decay rate *and* bias correction determined from the context-dependent vector to encode and decode the prediction error. As long as the errors have a consistent bias and follow some sort of geometric distribution, the encoding for errors should be shorter after a few occurrences of the context establish meaningful statistics.

Clearly, for the value $m = 1$, each prediction error is uniquely encoded by its bucket code. For any $m > 1$, however, we need k bits to encode the remainder.

²The default for NEAR is 0. Changing the default implements the near-lossless part of the JPEG-LS standard, which we do not address in this paper.

³If the first non-zero element of the vector (Q1,Q2,Q3) is negative, all of the vector signs are reversed to obtain (-Q1,-Q2,-Q3). This maps likely symmetrical vectors to the same context. A more robust explanation of this and the special case of the (0,0,0) vector are found elsewhere.[2]

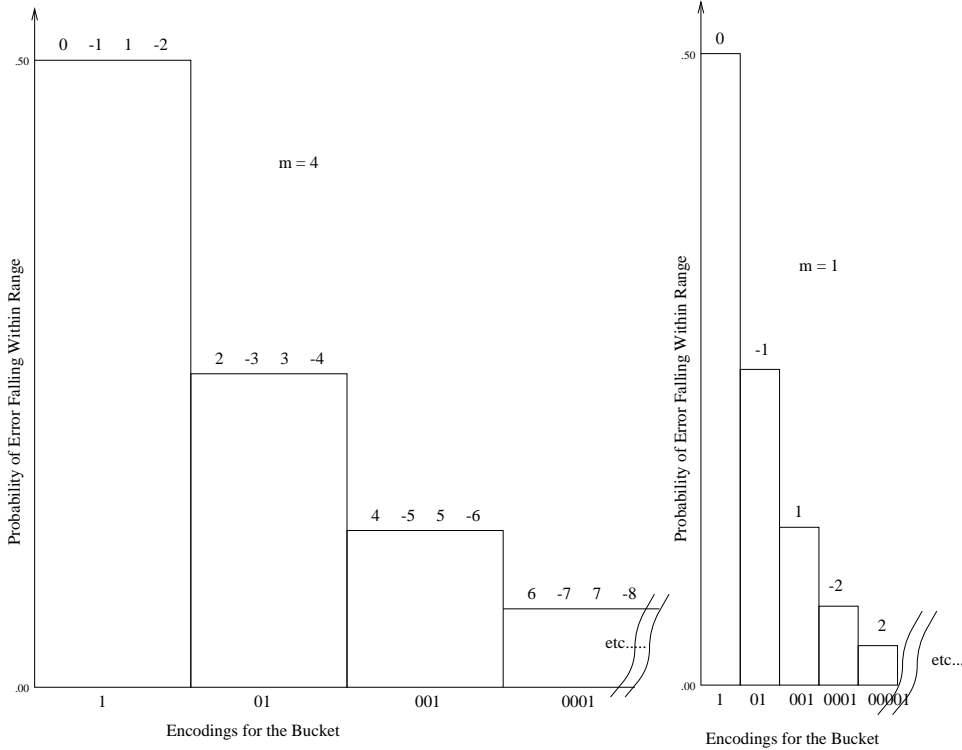


Figure 3: Golomb code with $m=4$ compared to $m=1$

(1) Bias Correction.

Each context has an associated value C , the correction value to offset bias. This error feedback mechanism is aimed at “centering” the distribution. The Golomb-Rice coder takes the value P_x corrected by this bias, and determines the prediction error from this new value.

(2) Decay Rate.

Each context also has an associated value N , the count of the number of occurrences of the context so far, and A , the accumulation of the absolute value of the prediction errors so far. The resulting k from the operation

$$\text{for } (k=0; (N \ll k) < A ; k++);$$

determines the decay rate m of the distribution that will provide the best encoding.

$$m = 2^k$$

From these equations it should be clear that since k has a least value of 0, m has a least value of 1, ensuring that any prediction error value can always be coded (see Figure 3 for an example of $m = 1$.)

Prediction error	Mapped value	Code (bucket,remainder)
0	0	1 00
-1	1	1 01
1	2	1 10
-2	3	1 11
2	4	01 00
-3	5	01 01
3	6	01 10
-4	7	01 11
4	8	001 00
-5	9	001 01
5	10	001 10
-6	11	001 11
6	12	0001 00
-7	13	0001 01
7	14	0001 10
-8	15	0001 11

Table 3: Example coding of prediction error for m=4

Pitfalls of the JPEG-LS lossless standard

Non-Geometric distributions

If the distributions are not geometric, the encoder can actually expand the size of the encoded image. An example is an image where most of the pixels have a prediction error of 4 or -4. This bimodal distribution can cause us to use five and four bits respectively to encode values that should require only one or two bits.

The same problem holds true for the ranking system used to encode run-lengths. If the run-length histogram is skewed greatly from a geometric distribution, encoding of run lengths may be less than optimal.

Sparsely populated contexts

For small images, many contexts may be unpopulated when using the default threshold values. Furthermore, for images with many vertical and horizontal components, contexts may be lightly populated as the meaningful buckets are divided by contexts introduced by the uncorrelated values of c or d. This means that if c or d are random predictors of the values for x, they divide the meaningful context by

$$9 \times 9 = 81$$

more contexts. Given 30 pixels per inch, a 2 inch by 2 inch image with 3600 pixels, which decodes half of its pixels by run-length encoding, may have 5 or fewer samples for each context *even at the completion of its encoding*.

The standard does not allow the encoder/decoder to ignore selected pixels (such as c or d) selectively. The only way to reduce or redistribute the contexts is to alter the T threshold values. Since this affects the context determinations for D1, D2, and D3 simultaneously, this is not a good solution.

Threshold values have global impact on contexts

T1, T2, and T3 are used to quantize each difference, D1, D2, D3. This doesn't give much flexibility to the encoder of the ways to perform the quantization. A change in T1 affects the thresholds for D1, D2, and D3 simultaneously. This means that to populate the contexts where $Q1 = 3$, we increase the value of T3. Unfortunately, this also increases the population for vectors where $Q2 = 3$ and $Q3 = 3$.

A better implementation would be to have an individual threshold for each Di (9 thresholds total). These could be $T1_1, T1_2, T1_3, T2_1, T2_2, T2_3, T3_1, T3_2$ and $T3_3$. This gives us a slightly modified context generation algorithm.

```
if ( Di ≤ -T3i) Qi = -4;
else if ( Di ≤ -T2i) Qi = -3;
else if ( Di ≤ -T1i) Qi = -2;
else if ( Di < - NEAR) Qi = -1;
else if ( Di ≤ NEAR) Qi = 0;
else if ( Di < T1i) Qi = 1;
else if ( Di < T2i) Qi = 2;
else if ( Di < T3i) Qi = 3;
else Qi = 4;
```

Individual thresholds would allow the encoder to build a fairly detailed context model that can be tuned to populate contexts quickly and evenly. If the search for correct threshold values to achieve greater compression is too CPU intensive, the encoder can always decide to use the default values or set the thresholds according to the old technique.

Results

Spend more time finding optimal encoding

Most images are written once and read many times. Because of this, the encoding time is not as critical as the amount of compression and the decoding time. Customizing thresholds fine tunes the context model for a particular image. This can very likely yield greater compression. Although this tuning may require an increased number of CPU cycles on a processor, this one-time additional expense may be less than the savings gained by many transmissions of a slightly smaller image. The exact amount of additional time we should spend further compressing the image is determined by the following analysis.

Variables

$M_{timetodecodeabit}$
 $E_{numoftransmissions}$
 $C_{CPUtimetotuneabit}$
 $T_{totaltransmissiontime}$
 $L_{imagelengthbits}$
 $U_{timetotransmitabit}$

Simplifications

To simplify the math, we scale M to U, which means that M is the fractional amount of time it takes to decode a message compared to the time to transmit the message. Keep in mind that for the base encoder, the encoding and decoding time are the same. This allows us to take U as 1 and simplify the math.

Using another simplification, let's assume that increasing the number of CPU cycles to find better threshold values does give better compression, but with decreasing returns.

This means that:

$$T = U \times L \times \frac{1}{C}$$

Base analysis, slow encoder/fast transmission, $M > U$

To minimize the amount of total time, the encoding and (decoding + transmission time) should be equivalent. For the original approach, we use 1 CPU cycle and run one encoding for one image:

$$MLC = T + LM$$

Since JPEG-LS allows us to decode in parallel with the transmission, we instead have:

$$MLC = \max(T, LM)$$

expanded:

$$MLC = \max\left(U \times L \times \frac{1}{C}, LM\right)$$

and for $M > U$:

$$MLC = LM$$

In this case expending more CPU cycles only increases the total time. Does this change if we write the image once, and perform E transmissions (reads) of the compressed image?

$$MLC = \max(T, LM) \times E$$

If $T \leq LM$, the answer is still no. We cannot improve the time by increasing C , since that compresses the file, but doesn't speed up the decoding (which is the controlling value of max).

**Fast encoder/slow transmission/many transmissions, $M < U$,
 $E > 1$**

Instead let us assume that T is *greater* than LM . This is closer to reality, where CPU speeds are much faster than transmission speeds, and the gap between them continues to grow.

Taking U as 1 and expanding T :

$$MLC = \max(L \times \frac{1}{C}, LM) \times E$$

Simplifying for $T \leq LM$:

$$MLC = L \times \frac{1}{C} \times E$$

Simplifying by removing L from both sides yeilds:

$$M \times C = \frac{1}{C} \times E$$

It doesn't take Einstein to figure out that multiplying both sides by C and reversing the order gives us:

$$E = MC^2$$

Keeping in mind that M may be a very *small* fraction, we should spend more time encoding according to the relationship:

$$C = \sqrt{E/M}$$

If we can complete an encoding ten times faster than a transmission, and the image will be downloaded 10,000 times, we should spend 1000 times as many cycles compressing as decoding. Even if we change the compression function to:

$$T = U \times L \times \frac{1}{C^2}$$

so that:

$$C = \sqrt[3]{E/M}$$

the previous figures still indicate that we should spend $\sqrt[3]{100,000} \approx 45$ times as many CPU cycles compressing as decoding.

The most popular uses for JPEG-LS involve write-once, read-many and transmission times that are longer than decoding times. Given these conditions, we should spend more time finding an optimal compression than transmitting or decoding.

This is why JPEG-LS should provide the encoder with the option of customizing contexts and encoding schemes to optimize compression, even if this customization is CPU intensive.

Future Work

Implementing the threshold finder would require slight encoder modifications, changing references to T1, T2 and T3 to T1_i, T2_i, T3_i as outlined previously.

Following this, the threshold finder simply encodes the image for all possible values of the nine thresholds and picks the thresholds that provided the most compression.

This should improve compression over the current standard.

Conclusion

The JPEG-LS lossless standard uses edge prediction, run-length coding, context determination based on a causal template, initial prediction based on a plane through causal template values, and Golomb-Rice encoding with decay-rate adjustment and bias correction for the prediction error.

The standard is optimized for an image with many horizontal and vertical edges, smooth grading transitions in other areas, and repeating contexts throughout the image. The JPEG-LS standard completes compression or decompression in one pass of the bit stream, without using any instructions more computationally complex than shifting, table lookup, or addition/subtraction.

To further improve the standard and reduce transmission time for images that are accessed many times, the standard should allow the encoder to determine custom thresholds to quantize each of the three difference values separately. Future work should demonstrate that this approach improves compression.

References

- [1] Golomb, S. W. Run-length encodings. In *IEEE Transactions on Information Theory*, pages 399–401, July 1966. [golo66]
- [2] (JPEG/JBIG), ISO/IEC JTC1/SC29 WG1. *FCD 14495, Lossless and near-lossless coding of continuous tone still images (JPEG-LS)*. ISO/IEC JTC1/SC29, July 1997. [jpeg97]
- [3] M. J. Weinberger, G. Sapiro, G. Seroussi. Loco-i: A low complexity, context-based, lossless image compression algorithm. In *Proceedings of the 1996 Data Compression Conference*, pages 140–149, April 1996. [wein96]
- [4] R. Gallager, D. V. Voorhis. Optimal source codes for geometrically distributed integer alphabets. In *IEEE Transactions on Information Theory*, volume IT-21, pages 228–230, March 1975. [gall75]
- [5] Rice, R. F. Some practical universal noiseless coding techniques. Technical Report JPL-79-22, Jet Propulsion Laboratory, Pasadena, CA, March 1979. [rice79]