Two-Level Context Based Compression of Binary Images

Alistair Moffat

alistair@cs.mu.oz.au

Department of Computer Science,
The University of Melbourne,
Parkville 3052, Australia.

Abstract: Variable order models coupled to an arithmetic coder have proved a successful paradigm for text compression. Here we explore the usefulness of a similar scheme for binary images. An algorithm due to Langdon and Rissanen is used as test bed for an experimental investigation, and with a two-level scheme based upon a conditioning context of 18 pixels a compression gain of about 20% can be achieved. Other experiments show that, for the test data used, this compression is at most about 30% inefficient.

1 Introduction

In [LR] Langdon and Rissanen described the application of arithmetic coding to the problem of compressing black-white binary images. They showed that arithmetic coding allows a finite state machine model to be used to record a 'context' based on a number of pixels in the immediate vicinity of the pixel that is to be coded, and that conditioning the probabilities based upon this context gives good compression. The resulting 'LR' compression scheme is a benchmark for binary image compression.

The finite state machine paradigm has also been used with success in text compression. The Prediction by Partial Matching (PPM) scheme of Cleary and Witten [CW,Mo2] makes particularly good use of this idea of a conditioning context, and in recent trials [BWC] has been shown to be amongst the best of current general purpose data compression schemes.

The PPM scheme also employs a number of other heuristics that improve the compression in a practical sense. Use of a large finite state model may improve the compression asymptotically, but might also require a 'learning' period far longer than the actual data stream to be compressed, and so prove ineffective. In the PPM scheme this problem is avoided by the use of a variable order Markov model. At the beginning of the stream a low order model is used to drive the coder while statistics are accumulated for higher order models. The low order model can be expected to converge quickly, and in the early stages of the encoding the compression will be no worse than for the simple model. When sufficient statistics have been accumulated in the more complex model that it can be used with some confidence, the predictions made by the low order model are biased in favour of those made by the high order

---

This work was in part supported by Telecom Research Laboratories (Australia)
model, and after a substantial portion of the text has been processed the low order predictions can be discounted entirely.

In this way the compression will 'bootstrap' itself from a low-order model to a higher order model, perhaps combining several intermediate models as well. For example, the PPMC program used in the experiments of Bell, Witten and Cleary [BWC] uses a third order byte-based model, but also makes use of second order, first order, and zero order models, and an 8-bit unweighted code if none of the Markov models can be applied. Provided that suitable weightings are given to the predictions of each of these models at each stage of the coding the compression from the combined model will never be worse than that obtained by each of the individual models, and the 'compression curve' obtained — the instantaneous compression rate for each byte of the input stream — will be the envelope of the compression curves for the individual models.

Here we consider the application of this idea to the LR image compression scheme. We first examine models that might be used for binary images, and show that they too suffer from the problem of being asymptotically good, but slow to become useful. We then describe a two-stage model based upon the strategy employed by the PPM scheme. This two-stage image compression model offers a small improvement in compression, and, together with a larger template than used by Langdon and Rissanen, yields compression approximately 20% better than that of the original LR scheme.

Finally, we consider a number of techniques that can be used to estimate an upper bound on the compression that can be obtained. These estimation techniques suggest that the compression obtained by the modified LR scheme is at most about 30\% inefficient.

2 Coding Templates

To establish a conditioning context, Langdon and Rissanen suggested the use of a number of pixels surrounding the pixel to be coded. They elaborated upon an earlier 4-bit encoding scheme of Musmann and Preuss [MP] and described two templates, one of 7-bits and one of 10 bits:

\[
\begin{align*}
.XX&&X&&X&&X \\
.XX&&*&&??&&? \\
\end{align*}
\]

and

\[
\begin{align*}
.X&&X&&X&&X \\
X&&X&&X&&X&&X \\
X&&X&&*&&??&&? \\
\end{align*}
\]

where an 'X' marks each pixel included in the template, the values of which are already known; '*' marks the pixel about to be coded; '?' marks pixels whose values are not yet known by the decoder and so cannot be included in the template; and '.' indicates a pixel whose value is known, but is not included in the template.
The implementation of the scheme is straightforward. Suppose that $p$ is the number of pixels in the template being used. Then at each coding stage the pixels under the template are evaluated to form an $p$-bit binary value. This value is then used to index a list of occurrence counts; one count for the number of times a '0' has occurred in that context, and one count recording the number of times a '1' has occurred in that context. For small values of $p$ the indexing is done by directly accessing an array; for larger $p$, when there are likely to be only a few of all possible contexts used, some indirect indexing method, such as a hash table or a search tree should be used. In either case the frequency counts are then used to drive an arithmetic coder; for details of how this should be done see [LR,RL,WNC].

For an $p$-bit template space will be required for at most $2^{p+1}$ integer counts. These can be usefully stored in as little as 8-bits, and so for a 10-pixel template the compression and decompression can be effected using as little as 2048 bytes.

Figure 2 shows the compression attained by these two templates on a test suite of binary images. Included are a number of other compression schemes that might be considered for use on binary images. These other methods are:

- A byte based run-length coding scheme, in which multiple occurrences of the same byte-aligned 8-bit pattern within the raster image were replaced by an escape byte, a repeat count byte, and then one instance of the byte to be repeated. This method was supplied as a standard tool by the manufacturer of the hardware that was used.

- A bit based run-length encoder based upon the group three facsimile transmission standard, using two static code tables, one for white runs and one for black runs, as described by Hunter and Robinson [HR].

- A bit based run-length encoder using adaptive arithmetic coding [Mo1] to transmit the run lengths rather than fixed codes, again with one set of parameters for white runs and one set for black runs.

- The PPMC [Mo2] program, processing the image array in row-major (raster) order.

- The PPMC program, processing the image file as a two dimensional array of 8-bit bytes in column-major order, i.e., processing 8-bit wide vertical stripes of the image.

The compression factors given are the ratio of the total of the original file sizes to the total size of the compressed files, and so the larger the number, the better the compression. Values less than 1.0 indicate expansion.

The test files were 79 binary images digitised from a range of drawings and photographs. This style of image was regarded as representative of what might appear, for example, in the non-textual parts of an electronic newspaper. Image size ranged from $323 \times 252 = 81,396$ pixels to $1,152 \times 900 = 1,036,800$ pixels, and the 79 images totalled 31,582,744 pixels, or about 400,000 pixels each. One of the more complex images used is shown as figure 1.
For each method there was a wide variation in the individual compression factors attained on the test files. For example, the compression factors attained by the LR method with \( p = 7 \) ranged from 1.23 for file 'cheetah' (figure 1) to 36.0 for a test file called 'stars', an image that is almost entirely black. However the weighted sum of the individual compression was regarded as a realistic measure of overall performance, and in most cases captured the relative performance of the various methods tested.

Processing the image as a two dimensional array of bytes in column major order with the PPMC program yielded a significant compression gain compared with row major order, as it meant that each pixel except the first of each byte was effectively coded in a context of several pixels above and also pixels to the left, rather than the 'left only' context used when the compression treated the raster image as a stream of bytes. Even so, the compression obtained was only as good as that generated by the two template based methods, which also performed well. The performance of the two LR methods is even more impressive when their total storage requirement of just a few kilobytes is also taken into account. By way of contrast, the PPMC method built models of several hundred kilobytes.

The next set of experiments considered a wide range of other templates, ranging from 1 pixel to 22 pixels. A summary of these templates, and the compression values that resulted, are shown in figure 3. For each value of \( p \) the template shown is that which yielded the best overall compression for the test suite. Note that the revised 10-pixel template listed provides, for the images tested, a marked compression improvement over the previous 10-pixel template described by Langdon and Rissanen.

The tradeoff between the compression accuracy attained and the 'learning' cost of using a large model can be clearly seen. For models based on templates of fewer than about 12 pixels the learning cost is small, and so compression improves as the model gets larger. On the other hand, when more than about 14 pixels are used the model does not converge to a useful state within the number of pixels being encoded, and so compression degrades as the model gets larger.

To measure the size of the learning cost the programs were modified to record the combined self-entropy of the test data when measured by each of these models.

For any context \( c \) suppose that \( n_0(c) \) and \( n_1(c) \) are the number of occurrences of '0' or '1' respectively for that context, counted over one particular image. Then the self-entropy of that image with respect to that model is given by

\[
\sum \left( n_0(c) \cdot \log_2 \frac{n_0(c) + n_1(c) + 2\epsilon}{n_0(c) + \epsilon} + n_1(c) \cdot \log_2 \frac{n_0(c) + n_1(c) + 2\epsilon}{n_1(c) + \epsilon} \right)
\]

where the summation is over all possible contexts \( c \), and \( \epsilon \) might be taken to be either zero or one or any value between. Taking \( \epsilon = 0 \) is very optimistic, in that if a symbol is assigned a zero probability then that event can never be coded, and so these counts could not be used as a general purpose initialisation of the model. On the other hand, taking \( \epsilon = 1 \) makes provision for every symbol to be available in every context, but is pessimistic for the particular image that was used to generate the frequency counts.

The resultant self-entropy is a measure of the compression that could be obtained if, by some chance of luck, both encoder and decoder knew each conditional prob-
ability without being required to pay for any transmission or learning costs. If the self-entropy of all the images are summed, a value for the self-entropy of the test suite is obtained. These values (with $e = 1$) are shown in figure 4 for some of the templates listed in figure 3. The second value shows the difference between the measured compression and the self-entropy, expressed as a percentage of the measured compression.

The values calculated in this way are an upper bound on the compression that could be obtained by either using a variable order model or in some other way initialising the parameters controlling the model, but still retaining the ability to encode any image. As expected, there is limited potential for compression improvement in the small models, but some prospect for improvement with the larger models. Compression factors with $e = 0$ were even higher (4.74 with $p = 18$, 5.84 with $p = 22$), but these were considered to be unrealistic, since an initial model based on these parameters might be unable to compress all images.

### 3 Two Level Coding

To allow a large model to 'learn' the distribution of parameters without allowing inaccurate predictions to affect the coding efficiency, and so to come closer to the self-entropy bound, a series of two level coding schemes were implemented. The 14, 18 and 22 bit templates offer a very detailed context in which the next bit can be predicted, but each of these large contexts needs to be initialised before it can be relied on.

The programs were modified so that a pixel was coded in the full $p$ bit context only if that context had already been observed at least twice before. If, because of lack of sufficient prior occurrences, the full context was not regarded as being a reliable predictor, a subset of the full context was used to generate a smaller template. Figure 5 shows the templates that were used and the compression improvement that was achieved. The pixels used for the subordinate context are shown as 's'. Again, the templates shown are those that gave the best results, and a wide range of other templates were also tested.

Small improvements were recorded for the 14 and 18 pixel templates, but the overall compression obtained was still only a little better than that obtained by the 12 and 14 bit templates listed in figure 5. More marked improvement was recorded for the 18 and 22 pixel templates. Interestingly, all of these 'best' methods gave virtually identical compression behaviour, and the large relative improvement achieved for the cases $p = 18$ and $p = 22$ served only to make them as efficient as the model using smaller templates, for which smaller relative gains were achieved.

### 4 Initialising the Model

Another way to try and reduce the learning cost is to use some heuristic or external knowledge to initialise the model. For example, when all of the context pixels are black it seems unlikely that both white and black pixels should, in the absence of
any statistics, be predicted to be equally likely. More generally, the initial counts in each context, which must both be initialised to some non-zero value so that both possibilities can be coded, might be assigned based on the number of the pixels comprising the context that are black, and their distance from the central pixel that is to be coded in this context.

A variety of such 'initialisation' schemes were experimented with, but, surprisingly, none improved the compression. For all of the schemes tried it was more effective to initialise each context to an equi-probable state, and allow the occurrence counts to adapt from there. We can only conclude that contexts do not end up making predictions based primarily upon the dominant pixel value in the context, and that the compression effect is more subtle than had been anticipated.

5 A Limiting Template

To try and gauge whether some other technique might improve upon the LR template based method we also examined a hypothetical 'absolute best' compression scheme. To obtain an upper bound on the amount of information that any compression scheme might use to predict the value of each pixel, a number of 'clairvoyant' template based models were considered. In these models we allowed templates to include pixels from the 'future' as well as from the 'past'. Clearly, such a scheme is impossible, but it does serve to provide another way of estimating upper bounds on compressibility. Figure 6 lists the templates that were used and the compression results. For each template the value given reflects the adaptive compression that would result if that template could be used, and may be compared with the values in figure 3.

The corresponding self-entropy values were 4.97 and 5.27 for \( p = 16 \) and \( p = 20 \) respectively \((\epsilon = 1)\). It is difficult to contemplate any legitimate compression scheme that has more information available to it than knowledge of all of the neighbours of each pixel that is to be coded together with occurrence counts for each of those contexts, and so these values indicate in some sense a model independent upper bound on the compression that might be obtained for the test suite. This upper bound would also include non-template based compression schemes, such as hierarchical transmission \([Kn,WC]\) and block representation \([Ma]\). We conclude that the LR paradigm is very strong, and that, for the test suite considered, it obtains compression at most about 30\% inefficient while using as little as 14 pixels of context and about 40 kbyte of data space.

6 Summary

The basic 7 and 10 pixel templates proposed by Langdon and Rissanen can be slightly improved by allowing more pixels to influence the predictions made for each bit. For the images tested contexts of 12 and 14 pixels maximised the tradeoff between the model learning cost and the asymptotic compression gained by using a larger model. The compression could be further slightly improved with the use of a two stage process, where a full 18-bit context was only used if it had already been encountered
twice before. The combined effect of these improvements is a compression gain of about 20% compared with the original LR scheme; other experiments have suggested that this is within about 30% of the compression that might be obtained on the test files by any compression scheme.

References


Figure 1: Test image 'cheetah'

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compression Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte run length coding</td>
<td>1.70</td>
</tr>
<tr>
<td>Bit run length, fixed</td>
<td>0.72</td>
</tr>
<tr>
<td>Bit run length, adaptive</td>
<td>2.18</td>
</tr>
<tr>
<td>PPMC, horizontal</td>
<td>2.42</td>
</tr>
<tr>
<td>PPMC, vertical</td>
<td>2.97</td>
</tr>
<tr>
<td>LR, p = 7</td>
<td>3.08</td>
</tr>
<tr>
<td>LR, p = 10</td>
<td>2.99</td>
</tr>
</tbody>
</table>

Figure 2: Compression Factors for Standard Algorithms
Figure 3: Compression factors for LR compression

Figure 4: Self entropy with respect to LR models
### Figure 5: Two level Compression improvement

| LR, p = 12, 6 | 3.57 | +0% |
| LR, p = 14, 8 | 3.60 | +1% |
| LR, p = 18, 10 | 3.67 | +7% |
| LR, p = 22, 10 | 3.67 | +16% |

### Figure 6: Compression based on 'clairvoyant' templates

| LR, p = 8 | 3.24 |
| LR, p = 12 | 4.00 |
| LR, p = 16 | 4.34 |
| LR, p = 20 | 3.93 |