Lossless Image Compression Using Pixel Reordering

Michael Ciavarella

Alistair Moffat

Department of Computer Science and Software Engineering The University of Melbourne Victoria 3010, Australia

http://www.cs.mu.oz.au/

Abstract

Lossless image compression techniques typically consider images to be a sequence of pixels in row major order. The processing of each pixel consists of two separate operations. The first step forms a prediction as to the numeric value of the next pixel. Typical predictors involve a linear combination of neighboring pixel values, possibly in conjunction with an edge detection heuristic. In the second step, the difference between that prediction and the actual value of the next pixel is coded. In high-performance mechanisms such as JPEG-LS, the error differential is coded in a conditioning context established by a possibly-different set of neighboring pixels. A per-context arithmetic, minimum-redundancy, or Rice coder completes the processing of each pixel.

In this paper we explore pixel reordering as a way of reducing the start-up (or learning) cost associated with each context. By permuting the prediction errors into an order that reflects the assessed volatility of the conditioning context, we are able to use a single coding context, with the changing probability estimates captured by local adaptation in the coder. In this sense, our proposal has elements in common with the Burrows-Wheeler text compression mechanism. The result is a lossless compression implementation that achieves excellent compression rates.

Keywords: Image compression, pixel reordering, Burrows-Wheeler transformation.

1 Introduction

Lossless image compression techniques typically consider images to be a sequence of pixels in row major order. As each row of pixels is handled, all pixels in the preceding rows are available to the decoder and hence to the encoder too, and these already-known values provide useful information to suitably bias the prediction as to the next pixel value. Pixels in the current row prior to the one in question are also available as part of the context.

The processing of each pixel consists of two separate operations. The first step forms a prediction as to the numeric value of the next pixel. Typical predictors involve a linear combination of neighboring pixel values, possibly in conjunction with an edge detection heuristic that attempts to allow for discontinuities in intensities.

Copyright © 2004, Australian Computer Society, Inc. This paper appeared in the Proceedings of the Twenty-Seventh Australasian Computer Science Conference, Dunedin, New Zealand, published as *Conferences in Research and Practice in Information Technology*, volume 26, edited by Vladimir Estivill-Castro. Reproduction for academic, not-for profit purposes is permitted provided this text is included.

In the second step, the difference between the predicted pixel value and the actual intensity of the next pixel is coded using an entropy coder and a chosen probability distribution. In high-performance mechanisms such as JPEG-LS [Weinberger et al., 1996] (see also http://www.hpl.hp.com/loco/), the error differential is coded in a conditioning context established by a possibly-different set of neighboring pixels, and multiple different probability distributions are maintained in parallel as the image is coded, one for each possible value arising in the context evaluation heuristic. Memon and Wu [1997] provide a useful overview of this type of image compression technique.

A per-context arithmetic or minimum-redundancy coder completes the processing of each pixel, using the probability distribution established by that context and built up in the transmission of the previous residuals encountered in that context. Alternatively, if maintaining low computational cost is more important than obtaining high compression effectiveness, a Rice or Golomb coder might be used. For example, the LOCO-I mechanism of Weinberger et al. [1996] uses a Rice coder at this stage of the process. Use of a simpler coder also reduces the parameter space associated with this step, and means that the start-up time, or "learning cost" associated with all adaptive coders, can be reduced.

In this paper we explore pixel reordering as an alternative way of reducing the start-up costs associated with multiple contexts. In a conventional system, the number of different conditioning contexts needs to be controlled, so as to avoid the problem of context dilution, which in essence is a blowout in costs caused by there being too many contexts for the probability estimates to converge in a satisfactory way.

By permuting the set of pixels (or the set of residuals) into an order that reflects the assessed volatility of the conditioning context, and allowing the set of pixels handled by one context to flow seamlessly into the set of pixels associated with the next context, we are able to use a single coding context, with the changing probability estimates captured by local adaptation in the coder. In this sense, our proposal has elements in common with the Burrows-Wheeler text compression mechanism.

The result is a lossless compression implementation that achieves excellent compression rates.

2 Context-based text compression

The Burrows-Wheeler transformation has proven to be a useful technique for text compression [Burrows and Wheeler, 1994]. It takes a stream of integers (normally taken to be a sequence of byte values), and reorders them into an equivalent sequence in which each byte is located

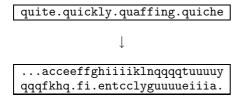


Figure 1: The Burrows-Wheeler transform applied to the string "quite quickly quaffing quiche". The last line is the output of the transformation, and the line above it shows the preceding character that is the dominant part of the sort key. Only the bottom line of characters is transmitted to the receiver.

according to the immediately preceding characters in the original string. For details of this transformation, the reader is referred to Moffat and Turpin [2002].

In text, the immediately preceding characters form a good basis for estimating a probability distribution across possible successors. So the BWT output can be thought of as the concatenation of subsequences, with each subsequence arising from a particular combination of preceding characters. For example, all of the successors of "q" form a single subsequence in the BWT output, and that subsequence can be further subdivided into the "aq", "bq", "cq", and so on, components. Figure 1 shows a short input string, and the corresponding BWT output. Comparing the second and the third lines, note the way in which all of the successors of "q" appear together in the output in one section, and how the letters that follow "qu" similarly appear together in a different section of the output.

The BWT output is then further transformed, in order to capitalize on the locally distinctive nature of each subsequence. This is usually achieved with a move-to-front (MTF) or other ranking transformation, which replaces each integer in the sequence by a count of the number of distinct values that have been encountered since the last appearance of that value. For example, the "qu"-prefixed subsequence derived from some input text might include the string "a,a,a,e,a,a,o,i,a,e,e,o,o". This would be transformed into the equivalent sequence " t_a ,0,0, t_e ,1,0, t_o , t_i ,2,3,0,3,0", where t_x is the value that would be generated on the first use of symbol x.

The beauty of the MTF transformation is that it allows exploitation of localized repetitions regardless of which symbols it is that are being repeated. So while the output of the BWT contains exactly the same distribution of character values as were input, the output of the MTF can be dramatically different. On input sequences that represent English text, the output of a BWT/MTF combination contains more than 50% of "0" symbols, and is amenable to entropy coding using either a minimum-redundancy (Huffman) or an arithmetic coder (see Moffat and Turpin [2002] for details of these coding mechanisms).

Similar context-based compression can be achieved through direct context analysis. In the PPM compression mechanism (again, see Moffat and Turpin [2002] for details) statistics are maintained in parallel for each n-gram of characters that has been observed in the preceding text, for some fixed value of n typically in the range three to seven. Those statistics are used to estimate probabilities when the same n-gram context reoccurs. Also required is a mechanism for escaping to lower-order contexts when a character has never been seen before in the nth order one, so that novel events can be accommodated.

The advantage of this on-line mode of operation is that the statistics are accumulated adaptively; on the other hand, many thousands of coding contexts are simultaneously active, and the entropy coder must be capable both of dealing with this plurality, and also of operating effective in an adaptive environment. These two constraints mean that almost inevitably an arithmetic coder is required. When it is, very good compression effectiveness can be attained.

The BWT/MTF system – which also processes characters in terms of the context established by the immediately preceding characters – is an off-line compression system, and must operate on a block-by-block basis to achieve good results. That is, both encoder and decoder must maintain block buffers, typically containing message blocks of one or more megabytes of source text.

On the plus side, in the BWT/MTF system there is not the same need for adaptive entropy coding, and no need for the bits generated from multiple contexts to be interleaved. In the BWT/MTF system just one coding context is used. Every MTF value is handled in the same way as all of the other ones, with reference to the same fast-adapting probability distribution. This makes such systems easier to handle at the coding stage than PPM.

3 Context-based image compression

The standard approach to image compression involves PPM-like conditioning, but with one extra step: prior to the selection of a context to be used as a basis for coding, the actual pixel value is estimated as a (usually) linear combination of a set of available neighboring pixels. It is the difference between the actual pixel and the estimated intensity that is coded in that context, rather than the pixel value itself [Sayood, 2000]. In addition, the conditioning context can be a two-dimensional one of nearby pixels, rather than the strictly one dimensional context of preceding characters that is the basis of the PPM mechanism.

Taking the difference between the predicted and actual pixel results in a reduction in the complexity of the coding domain in a manner not unlike the way that the MTF reduces the variability of the BWT output. The stream of differentials being coded should have a mean close to zero, and a smooth symmetric probability distribution that monotonically decreases as the magnitude of the residual increases. In the BWT/MTF case, the distribution of values coded is only one sided, but it is still monotonically decreasing.

In a context-based image compression system, one of the design issues is how best to select the context in which to code the error residual. Choice of too precise a combination of pixels and pixel values means that the number of contexts becomes large, and there is insufficient evidence available in the image to provide accurate support for any set of probabilities. On the other hand, when the set of contexts is too small, they end up representing aggregate statistics that, if separated, might provide better compression. That is, there is a fine balancing act between the need for each context to be frequent enough for its pixel predictions to be plausible, and the desire to use a large number of distinct contexts, so that each context is highly discriminating.

Our aim in this project is to investigate whether reordering pixels into a context ordering, and then using a single locally adaptive entropy coder, can approach the compression rates attained when the contexts are disjoint and the pixels are processed in strictly row-major order. The next section illustrates the framework we are considering, and then goes on to consider variants.

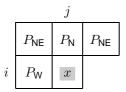


Figure 2: The set of immediate neighbors of an unknown pixel at offset (i, j) in the image.

4 Context quantizations

Consider the arrangement shown in Figure 2. The nearest scan-preceding neighbors of pixel x are the northern adjacent pixel, $P_{\rm N}$ and the western adjacent pixel, $P_{\rm W}$. Two other pixels, $P_{\rm NE}$ and $P_{\rm NW}$, are also likely to be strongly correlated with the value x. All four of these neighboring pixels are encountered prior to x when the image is processed in row-major (raster scan) order, and may be used by encoder and decoder to establish a conditioning context.

Suppose that for some reason it is decided to use $P_{\rm N}$ and $P_{\rm W}$ to establish a context. If each is an eight-bit value, then there are $2^8 \times 2^8 = 65,536$ possible contexts, and on a (say) $512 \times 512 = 262,144$ pixel image, each context is used an average of four times. Of course, some contexts might be used very heavily, while others are not used at all; nevertheless, there is a considerable risk that contexts will be used without an adequate volume of statistics being accumulated to render them accurate. Tischer et al. [1993] consider the issues associated with this form of context-based image compression.

One way of controlling the number of distinct contexts is to quantize them onto a smaller range. For example, we might choose to take some number of significant bits from each of $P_{\rm N}$ and $P_{\rm W}$, and combine them. Taking 6 bits of each, for example, creates $2^{12}=4{,}096$ different combinations, a considerable reduction.

We also need a way of predicting the intensity of x. Typically a linear combination of $P_{\rm NE}$, $P_{\rm N}$, $P_{\rm NW}$, and $P_{\rm W}$ is used, with an error e then calculated:

$$e = f(P_{NE}, P_{N}, P_{NW}, P_{W}) - x$$
.

Pixels even more remote from x might also be incorporated into the estimator. Possible estimators are considered below.

If coding is taking place on-line, and contexts are discrete, then there is nothing more to be done – the error residual e is adaptively coded in the context that has been calculated, and the next pixel is considered in the same way.

The alternative we consider in this paper is to associate a numeric context identifier C with each pixel, and use it as a basis for ordering either the set of error residuals e, or the set of pixel values x. The coding step can then deal with residuals or pixels in the permuted order. To ensure that the process is reversible, the i and j locations are used as secondary keys to force the stability of the set of pixels within each context. That is, a list of tuples

$$(C(P_{\mathsf{NE}}, P_{\mathsf{N}}, P_{\mathsf{NW}}, P_{\mathsf{W}}), i, j, e)$$

is formed covering the entire image, and is sorted into lexicographic order. The "e" column in this list of tuples is then sent to the decoder, in the same way that the "next character" column after the BWT is sent to the decoder.

A key component of this strategy is an additional requirement that the C function for assigning integers to pixel patterns should in some sense be "smooth", so that any given context selector C can be assumed to make predictions that are not unlike the predictions made by contexts C-1 and C+1, and to a lesser extent, not unlike those made by contexts C-2 and C+2, and so on.

As an example, and to draw out the notion of "smoothness" described in the previous paragraph, consider the two functions

$$f_1(P_{\mathsf{NE}}, P_{\mathsf{N}}, P_{\mathsf{NW}}, P_{\mathsf{W}}) = \left\lfloor \frac{P_{\mathsf{N}} + P_{\mathsf{W}}}{2} \right\rfloor$$

and

$$C_1(P_{NE}, P_{N}, P_{NW}, P_{W}) = |P_{N} - P_{W}|.$$

In this case, the next pixel value is being estimated as the truncated integer mean of the two closest neighbors, and the context – an assessment of a value that can be used to try and categorize how far the prediction might vary from the actual intensity – is taken to be the magnitude of the difference between the same two pixels. When the image is stable, C will be small, and the residuals should be tightly clustered about their mean. When the image is more dynamic, C is likely to be larger, and the residuals less tightly clustered.

Now consider the sorted list of tuples after the reordering. All of the C=0 values are followed by all of the C=1 values, all of the C=2 values, and so on, with the list rounded out (for eight-bit images) by tuples for the C=255 contexts, if any. Within each C value, the pixels are in row-major order.

In this scheme all error residuals for the cases when $P_{\rm N}=P_{\rm W}$ form a contiguous block in the output. Similarly, all of the error residuals for $|P_{\rm N}-P_{\rm W}|=k$ form a contiguous block, for each value of k up the maximum value determined by the nature of the C function being used. Experimental results demonstrating the power of this approach are given in Section 5.

Another way that we can use the same pixel reordering framework is via the functions

$$f_2(P_{\mathsf{NE}}, P_{\mathsf{N}}, P_{\mathsf{NW}}, P_{\mathsf{W}}) = 0,$$

$$C_2(P_{\mathsf{NE}}, P_{\mathsf{N}}, P_{\mathsf{NW}}, P_{\mathsf{W}}) = P_{\mathsf{N}} + P_{\mathsf{W}}.$$

Now the pixel value x is being directly coded, and the contexts are ordered according to a prediction as to what that value will be. Again, we expect clustering to appear once the reordering step has taken place. Both of these pairs of functions are explored in the experimental results presented below.

To allow the reordering to be reversed, the decoder is also told the number n_k of times context k appears in the list of context identifiers, for each possible value of k generated by the function C. This list of integers must be sent as a second component of the compressed image, accompanying the entropy-coded set of reordered pixel values e, using some suitable representation. In the case of eight-bit images and the function C_1 , the list consists of 256 integer values, and is a small overhead, even if sent completely uncompressed. For example, 256×32 bits represents a cost of 0.03 bits per pixel on a 512×512 pixel image.

When the number of different context values is larger relative to the image size, a more careful representation for the list of bucket sizes is necessary.

Decoding consists of building a table containing the n_k values, and recreating the list of e values in the permuted

Image	Size	Pixel Entropy
barb	512 × 512	7.47
boat	512×512	7.12
france	672×496	6.28
frog	621×498	4.97
goldhill	512×512	7.48
lena	512×512	7.45
library	464×352	5.85
mandrill	512×512	7.36
mountain	640×480	6.22
peppers	512×512	7.57
washsat	512×512	2.87
zelda	512×512	7.27
Average		6.49
·		·

Table 1: Test images used in experiments. The last column shows the zero-order self-information of the pixel sequence in the image, with no conditioning or prediction, measured in bits per pixel. It represents a simple assessment of compressibility.

order in which they were sent by the encoder. The original image is then regenerated in row-major order: for each pixel, the context value c is determined; the next residual e is extracted from the cth subsection of the list of residuals; that value is added to the calculated $f(P_{\rm NE}, P_{\rm N}, P_{\rm NW}, P_{\rm W})$ value for this location; and the resulting intensity is written to the output image.

5 Locally adaptive entropy coding

Provided that the conditioning function C has the requisite blend of predictive discrimination and smoothness, the list of reordered residuals should be locally homogeneous in nature, but with a shift in symbol probability distribution between the beginning and the end. It thus makes sense to employ a recency transformation when processing the permuted sequence, and/or a locally adaptive probability estimator for the eventual entropy coder.

When the BWT is used in sequential compression, an MTF transformation is coupled with a structured arithmetic coder [Fenwick, 1996] (or some other locally adaptive coder) to rapidly adapt to the varying rank statistics. In the situation described in the previous section, where the values might be pixel prediction residuals, it may be that an MTF is again an appropriate transformation, or it may be that a different combination of techniques needs to be devised.

Working with residuals

The first experiments we carried out were designed to determine whether or not the MTF transformation was of any benefit when pixel values were predicted via a plausible f function, and error residuals are coded.

As a test suite, we used the University of Waterloo "Waterloo Repertoire GreySet2" collection (available from http://links.uwaterloo.ca/greyset2.base.html). These twelve images are all eight bits deep, and range in size from 464×352 to 672×498 pixels. Table 1 summarizes the twelve images in the suite. The final column of Table 1 lists the zero-order self-information of the set of pixels in each file, and represents a lower bound on the compression that could be attained if the pixel values in each file were independent of each other.

Image	Residual	Residual,
C	only	reordered
	·	then MTF
barb	5.52	5.92
boat	4.90	5.23
france	2.62	0.42
frog	5.99	4.61
goldhill	5.04	5.35
lena	4.65	5.06
library	5.85	5.57
mandrill	6.27	6.62
mountain	6.73	6.55
peppers	4.79	5.15
washsat	3.08	2.41
zelda	4.28	4.59
Average	4.98	4.79

Table 2: Applying pixel reordering and then a ranking transformation to the stream of residuals. In this experiment, the estimation function is $f_1()$, and the context ordering function is $C_1()$. All values represent the zero-order self-information of the resultant stream of integers, in bits per integer. Reordering alone, without the ranking transformation, does not alter the self-information.

Table 2 shows the dramatic effect on compressibility that coding residuals has, compared to coding raw pixel values; and then the extent to which further compression gains might be possible through the use of pixel reordering and an MTF transformation. The values in Table 2 again represent the self-information of the stream of values, rather than an actual compression effectiveness. Compared to the third column of Table 1, the middle column in Table 2 shows that residuals are far more compressible than are raw pixel values. Even the simple prediction function f_1 greatly reduces the variability of values in the stream to be compressed.

To represent the residuals as positive integer values, the residual value is folded into the same range as the pixel values, by reflecting it about the predicted value, and interleaving the positive and negative values as far as is necessary. For example, if the predicted value is 5, then the space of possible residuals is

$$[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, \dots, 250],$$

and that set of 256 values gets mapped to the target space

$$[10, 8, 6, 4, 2, 0, 1, 3, 5, 7, 9, 11, 12, 13, \dots, 255].$$

To obtain the third column in Table 2, the same sequence of residual values was pixel-reordered using the $C_1()$ function as a primary sort key, as described in Section 4, and then further modified via the MTF transformation. Applying the pixel reordering process in isolation does not affect the zero-order self-information of the residual sequence, as the same values are present, merely in a different order. In this sense, the reordering achieves nothing.

But when the pixel reordering is followed by a ranking transformation, any locally significant variation in residual distribution should be exploited, with a consequent decrease in self-information. The third column of Table 2 shows that for some of the images – notably france, frog, and washsat – the potential gain brought by pixel reordering and the MTF is large. On the other hand, for the majority of the images, pixel reordering does not assist, and compression is worsened. It would appear that

in these files the MTF is too crude a tool to capture the localized patterns caused by the reordering. Use of the MTF does reduce the average cost, but the bulk of the gain arises from just a single file, and the improvement is not compelling.

The other way of exploiting the locality effect created by pixel reordering is to use a locally adaptive entropy coder. The compression rates in Table 2 are optimistic in one sense, in that they make no allowance for the set of bucket sizes, and assume a perfect entropy coder. In practice, adding in the list of bucket sizes, and using a (for example) Huffman coder adds slightly to the compressed size, around 0.05 bits per pixel.

On the other hand, the compression rates listed in Table 2 are pessimistic, in that they are the result of a self-information computation. As such, they are predicated on the assumption of a stationary source, whereas the express purpose of pixel reordering is to create a non-stationary source. Even after the subsequent ranking transformation, it might still be that the stream of values has local variations – it certainly does in the standard character-based sequential BWT process [Fenwick, 1996].

Hence, the next step in our investigation was to explore the use of actual entropy coders. Two implementations were used – a carefully engineered implementation of minimum-redundancy (Huffman) coding created as part of the investigation undertaken by Turpin and Moffat [2000]; and a structured arithmetic coder derived from the work of Peter Fenwick [1996].

The first of these two coders – shuff – processes a file of integers block by block in a semi-static manner. Before any of the symbols in the block are transmitted to the decoder, the block is analyzed, a canonical minimum-redundancy code constructed, and a description of the code transmitted (including a list of the distinct symbols that appear in that block). The input file is simply a list of integers, and arbitrary-sized alphabets can be handled. In particular, there is no requirement that the input symbols be in the range 0...255. This software is available from http://www.cs.mu.oz.au/~alistair/mr_coder/.

The other coder used is sint, also developed at the University of Melbourne. In a structured arithmetic coder, symbols are assumed to be drawn from a continuous probability distribution rather than a discrete one. A two-stage probability estimation process first assigns a probability to the magnitude of the current symbol, and then, within that range or bucket, assigns a probability to the symbol being coded. In each bucket, probability estimates are refined relatively slowly. But at the first level, the probability estimates are given a small half-life, and are adapted quickly so that gross shifts in the probability distribution are quickly adapted to. In the coder sint used in these experiments, the buckets are determined by a geometric sequence with radix 1.5. Rounded to integers, this puts symbols 1 and 2 in buckets of their own; [3, 4] in the third bucket; [5, 6, 7] in the fourth; [8, 9, 10, 11] in the fifth, and so on. An occurrence of any of the symbols in that fifth bucket results in the probabilities of all fours symbols in that bucket being boosted.

The two middle columns of Table 3 shows the result of applying the entropy coders to the stream of reordered residuals, after the MTF ranking transformation. Comparing the third column of Table 2 and the second column of Table 3 shows that when the entire image is processed as one shuff block, the previous self-information calculation is a fair assessment of compressibility. The systematic discrepancy between the entropy limit and the actual values shown is a result of the fact that the Huffman coder

Image	Residual,	Residual,	Residual,
C	reordered,	reordered,	reordered,
	then MTF,	then MTF,	then sint
	then shuff	then sint	
barb	5.97	5.60	5.16
boat	5.27	4.89	4.52
france	1.12	0.32	0.59
frog	4.66	4.53	5.32
goldhill	5.40	5.13	4.80
lena	5.10	4.89	4.47
library	5.63	5.08	4.89
mandrill	6.68	6.48	6.10
mountain	6.60	6.29	6.18
peppers	5.20	5.03	4.65
washsat	2.47	2.35	2.09
zelda	4.63	4.49	4.17
Average	4.89	4.59	4.41

Table 3: Applying pixel reordering and then either a ranking transformation followed by a structured arithmetic coder, or a structured arithmetic coder directly. The estimation function is $f_1()$, and the context ordering function is $C_1()$. Values are actual bit rates, and represent reversible compression systems, including the list of context bucket sizes required to invert the pixel reordering.

admits a small amount of compression inefficiency, and the need for 256 bucket sizes n_k to be prepended to the file.

When sint is applied to the same stream (column 3 of Table 3) there is a significant gain in compression effectiveness, a clear indication that there are localized correlations remaining, even after the taking of residuals, and even after the application of the MTF. Some of the gain also accrues from the use of an arithmetic coder – note in particular that image france is now being represented in less than one bit per pixel, a feat not possible with a minimum-redundancy coder.

However, even using sint, for more than half of the test suite the self-information of the non-MTF residuals (column two in Table 2) is less than the actual compression rate attained when the MTF is applied (column three of Table 2). This means that the role of the MTF is still uncertain.

To determine whether or not the MTF is contributing in any way, the final column in Table 3 shows the result of applying sint directly to the reordered sequence of residuals, without the intervening MTF transformation. Rapidly-adapting coding mechanisms have been used directly with the BWT for text compression [Wirth and Moffat, 2001]. Without the MTF, the adaptive coder itself is responsible for capturing and exploiting all local variations in the probability estimates. As the last column of Table 3 demonstrates, with the exceptions of files france and frog, compression of images is enhanced when the MTF transformation is removed; and for all of the files, compression effectiveness using sint exceeds the selfinformation bound. The pixel reordering is clearly having a beneficial effect, and a core component of our rationale in carrying out this investigation has been validated.

Figure 3 draws out the source of the compression gain that has been achieved. It shows, for a single image, the residual probability distribution in three different C_1 contexts, when the prediction function is f_1 . The evolution of the residual probability is relatively smooth as the context shifts, from a steeply-dropping function in context C=0,

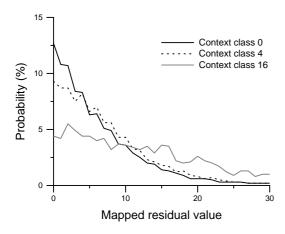


Figure 3: Probability of various prediction errors, for different values of the context function. These figures are for image peppers, with context function $C_1()$, prediction function $f_1()$, and no MTF. The residual has been folded, so that all values are positive.

through to a less skewed function when C=16. This migration in probabilities is neatly exploited by the structured arithmetic coder.

One of the drawbacks of sint is, of course, that it is an arithmetic coder, and thus somewhat slower in execution than a well-tuned minimum-redundancy or Rice coder. However local adaptation is also possible in the framework of a semi-static minimum-redundancy coder such as shuff, by simply breaking the overall sequence into smaller blocks, and processing each of the blocks independently. The experiments in Table 3 treated each image as a single block, and so the self-information is a lower bound on compression effectiveness. Use of a smaller block size means a greater overhead in terms of the combined costs of the preludes, as more codes have to be described. But if there are localized variations in rank probabilities, the extra cost might be recouped, plus more.

Table 4 shows the results of using three different block sizes with shuff, in an attempt to approach the compression effectiveness achieved by sint. Transmitting a fresh code for each 1,000 residuals is a significant cost. Even so, for one of the images (france), use of a very small block size generates the best overall compression effectiveness. A block size of 10,000 symbols is the best compromise across this particular test suite.

The bzip2 software of Seward and Gailly [1999], which is a carefully engineered implementation of the sequential BWT/MTF process, also uses localized minimum-redundancy codes in order to capture any high-frequency variations in the stream of MTF values.

Working with pixel values

We also explored the best way to employ the second pair of functions sketched above, C_2 and f_2 . Taking $f_2=0$ trivially means that x is coded as a raw pixel value, and taking C_2 to be the sum of the north and west pixels means that the context number used as the basis of the permutation should be closely correlated with the value x being coded. Note that there are now 511 different contexts in use, and that the cost of transmitting them to the decoder is correspondingly greater.

Table 5 summarizes the result of these experiments. When there is no use of a prediction function, the MTF is indispensable, and all of the results shown include an MTF

Image	Residual, reordered, then shuff			
	-b1000	-b10000	-b100000	
barb	5.59	5.27	5.29	
boat	4.87	4.63	4.70	
france	1.30	1.44	1.64	
frog	4.53	4.62	5.46	
goldhill	5.17	4.93	4.95	
lena	4.83	4.57	4.59	
library	5.23	5.11	5.58	
mandrill	6.63	6.25	6.21	
mountain	6.36	6.24	6.50	
peppers	5.02	4.74	4.76	
washsat	2.30	2.22	2.44	
zelda	4.48	4.25	4.26	
Average	4.69	4.52	4.70	

Table 4: Applying pixel reordering and then using a semistatic minimum-redundancy coder, with a range of different block sizes (indicated by the \neg b argument). The estimation function is $f_1()$, and the context ordering function is $C_1()$. Values are actual bit rates, and represent reversible compression systems, including the list of context bucket sizes required to invert the pixel reordering.

step. The same trend as before is apparent – namely, that excellent compression is attained compared to both the pixel self-information, and relative to the self-information of the residual stream; and that using shuff with a block size of 10,000 captures the majority of the localized effects that are exploited by the structured arithmetic coder.

There is, however, an interesting point to note, and that is that the two techniques are, in a sense, complementary: in files for which the " C_1 , f_1 , no MTF" combination is better than the " C_2 , f_2 , MTF" one, it tends to be better by a significant margin; and when the " C_2 , f_2 , MTF" method of Table 5 is better than the " C_1 , f_1 , no MTF" mechanism of Tables 3 and 4, the margin is again significant.

Table 6 shows the effectiveness attained by a hybrid system that applies both alternatives, and then retains the more compact file, adding a one bit flag to indicate which choice was made. The highlighted entries in Table 6 are discussed shortly.

Coding the list of bucket sizes

The first part of each compressed image consists of a list of bucket sizes, to allow the pixel reordering to be reversed. With the C_1 context function there are at most 256 buckets, and with the C_2 function, not more than 511. The compression results for shuff shown in the tables in this paper include the cost of handling these integers using the same minimum-redundancy code, while the sint compression rates are based upon passing them directly to the decoder as a list of 32-bit integers, without any compression taking place. A small additional saving of approximately 0.05 bits per pixel results if an appropriate compression technique is used for the list of 511 bucket sizes. One suitable technique is the Interpolative coder of Moffat and Stuiver [2000].

6 Topological reordering

Figure 4 shows a third way in which pixel reordering can be applied. If the standard raster-scan pixel ordering is handled sequentially, each pixel is set in a strictly onedimensional context of pixels to its left. This is why

Image	Pixel values, reordered, then MTF, then			
·	sint	shuff	shuff	shuff
		-b1000	-b10000	-b100000
barb	6.03	6.52	6.11	6.13
boat	5.16	5.54	5.25	5.34
france	0.39	1.26	1.15	1.14
frog	3.51	3.78	3.59	3.58
goldhill	5.37	5.77	5.44	5.47
lena	5.09	5.47	5.14	5.15
library	4.54	5.06	4.79	4.90
mandrill	6.64	7.19	6.73	6.67
mountain	5.00	5.41	5.10	5.21
peppers	5.18	5.56	5.24	5.27
washsat	2.32	2.50	2.33	2.34
zelda	4.69	5.01	4.72	4.71
Average	4.49	4.92	4.63	4.66

Table 5: Applying reordering to pixel values and then applying an MTF transformation, followed by entropy coding using either sint or shuff, with a range of different block sizes. The estimation function is $f_2()$, and the context ordering function is $C_2()$. Values are actual bit rates, and represent reversible compression systems, including the list of context bucket sizes required to invert the pixel reordering.

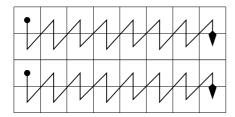


Figure 4: Permuting pixels to aid sequential compression programs. The reordering creates a pseudo two-dimensional context by placing neighboring pixels into the sequential context used by a one-dimensional compression regime.

tailored image compression tools outperform generalpurpose compression systems. The latter are completely unable to exploit the two-dimensional correlations present in most images.

If the pixel ordering is permuted to bring in some of that two-dimensional structure, better sequential compression might be observed. Table 7 shows the result of applying the pixel reordering shown in Figure 4, in conjunction with the general compression tool bzip2. The rearrangement is based purely upon pixel position, rather than a function of pixel value, hence the designation topological reordering. A range of such reorderings are considered by Memon et al. [2000].

Bzip2 is a high-quality implementation of the Burrows-Wheeler transform. When used sequentially, each pixel is coded in the context of its raster-scan predecessor, $P_{\rm W}$, and then in the secondary context of the pixel to the left of that one. When used on the shuffled pixel sequence, some of the west predecessors move further back in the sequence, and are replaced by nearby pixels that are either north or south of the one in question.

As can be seen in Table 7, use of the permuted pixel ordering usually (but not always) resulted in a small compression gain. The bold-face values in the table are discussed in the next section.

Image	Best	t so far
barb	5.16	(t3c4)
boat	4.52	(t3c4)
france	0.39	(t5c2)
frog	3.51	(t5c2)
goldhill	4.80	(t3c4)
lena	4.47	(t3c4)
library	4.54	(t5c2)
mandrill	6.10	(t3c4)
mountain	5.00	(t5c2)
peppers	4.65	(t3c4)
washsat	2.09	(t3c4)
zelda	4.17	(t3c4)
Average	4.12	

Table 6: A hypothetical composite compression system, using sint as a coder, and for each file choosing the better of the " C_1 , f_1 , no MTF" combination and the " C_2 , f_2 , MTF" alternative. The notation in the third column reflects the table and column from which that compression rate was derived.

Image	bzip2	Shuffled,	
		then bzip2	
barb	6.12	5.95	
boat	5.33	5.26	
france	0.39	0.30	
frog	3.46	3.42	
goldhill	5.60	5.56	
lena	5.30	5.20	
library	4.48	4.50	
mandrill	6.60	6.69	
mountain	5.13	5.08	
peppers	5.34	5.26	
washsat	2.52	2.42	
zelda	5.12	4.91	
Average	4.62	4.55	

Table 7: Shuffling the pixel order based on location in the image, and then applying a standard sequential compression process.

7 Summary

Our intention in this project was to develop a pixel reordering regime that captured some of the benefits of multicontext lossless image compression, but with a single coding context.

We have explored three different ways of achieving that goal.

In the first mechanism, a conventional prediction function was combined with a second "smooth" function intended to serve as a primary sort key. Good compression effectiveness was attained, particularly when the residuals were processed by a locally adaptive arithmetic coder.

The second mechanism is closer to the sequential BWT/MTF process, and involves a context estimation function, then direct processing of pixel values via an MTF transformation. Again, good compression can be attained using a locally adaptive entropy coder.

The surprising feature of these first two approaches is that they are complementary, and each of the test images tended to be handled well by only one of the two approaches. This aspect of their behavior suggests that better compression is possible (without the kludge of "applying both and then retaining the smaller output file") if

Image	FELICS	LOCO-I	CALIC
barb	5.34	4.73	4.45
boat	4.67	4.25	4.15
france	2.36	1.41	0.82
frog	6.13	6.05	5.85
goldhill	4.97	4.71	4.63
lena	4.60	4.24	4.11
library	5.47	5.10	5.01
mandrill	6.30	6.04	5.87
mountain	6.57	6.42	6.27
peppers	4.79	4.49	4.38
washsat	4.15	4.13	3.67
zelda	4.29	4.01	3.86
Average	4.97	4.63	4.42

Table 8: Compression results on the test suite using a range of compression programs.

a mechanism can be found that brings together these two distinct modalities.

The third mechanism – interleaving adjacent rows of pixels and then using a sequential compression regime – is so simple as to be almost trivial, yet it results in improved compression.

Table 8 gives a final perspective of the results achieved in this investigation, by listing the performance of a range of image compression tools. These values are all actual compression rates, in bits per pixel, measured as the output of validated programs. The columns correspond respectively to:

- The FELICS compression mechanism of Howard and Vitter [1993], using an implementation included in the mg software package at http://www.cs.mu. oz.au/mg/.
- An implementation of the LOCO-I technique of Weinberger et al. [1996], accessed from http://www.hpl.hp.com/loco/.
- An implementation of the CALIC approach developed by Wu et al. [1996], accessed from ftp://ftp.csd.uwo.ca.pub/from_wu/, using the variant based on arithmetic coding.

The highlighted values in Table 8, and also in Tables 6 and 7, correspond to the best compression attained by any system on that test file. As expected, the majority of the "best"s go to CALIC – it is a highly specialized image compression system, and sensitive to image characteristics in a number of different ways. Nevertheless, our hybrid system based on pixel reordering obtains superior compression on two of the test files (the highlighted values in Table 6). Finally, note that bzip2 – which is a completely general-purpose sequential compression system – performs remarkably well, and obtains one of the highlighted "best on file" awards in its sequential form, and scores two more when it was applied to a reordered input sequence (Table 7).

In terms of average behavior over the test suite, the best results are attained by the hybrid mechanism captured in Table 6. While this approach is in some ways a "rort" in that it combines two different sets of results, it might be that all that is required to convert these nominal result into a single real system is for an appropriate choice of mechanism to be made on a pixel-by-pixel basis. We plan to continue exploring this approach.

We also have ahead of us the possibility that better context estimation functions, and/or more elegant prediction functions, will provide further compression gains compared to those that we have achieved to date in this investigation.

Acknowledgement This work was supported by the Australian Research Council.

References

- M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, May 1994.
- P. Fenwick. The Burrows-Wheeler transform for block sorting text compression: Principles and improvements. *The Computer Journal*, 39 (9):731–740, Sept. 1996.
- P. G. Howard and J. S. Vitter. Fast and efficient lossless image compression. In J. A. Storer and M. Cohn, editors, *Proc. 1993 IEEE Data Compression Conference*, pages 351–360. IEEE Computer Society Press, Los Alamitos, California, Mar. 1993.
- N. Memon, D. Neuhoff, and S. Shende. Scanning techniques for context-based lossless image compression. *IEEE Transactions on Image Processing*, 9(11):1837–1848, Nov. 2000.
- N. Memon and X. Wu. Recent developments in context-based predictive techniques for lossless image compression. *The Computer Journal*, 40(2/3):127–136, 1997.
- A. Moffat and L. Stuiver. Binary interpolative coding for effective index compression. *Information Retrieval*, 3(1):25–47, July 2000.
- A. Moffat and A. Turpin. Compression and Coding Algorithms. Kluwer Academic Publishers, Boston, MA, 2002.
- K. Sayood. Introduction to Data Compression. Morgan Kaufmann, second edition, Mar. 2000. ISBN 1558605584.
- J. Seward and J. L. Gailly. Bzip2 program and documentation, 1999. sourceware.cygnus.com/bzip2/.
- P. Tischer, R. Worley, A. J. Maeder, and M. Goodwin. Context-based lossless image compression. *The Computer Journal*, 36(1):68–77, 1002
- A. Turpin and A. Moffat. Housekeeping for prefix coding. *IEEE Transactions on Communications*, 48(4):622–628, Apr. 2000. Source code available from www.cs.mu.oz.au/~alistair/mr_coder/.
- M. J. Weinberger, G. Seroussi, and G. Sapiro. LOCO-I: A low complexity, context-based, lossless image compression algorithm. In J. A. Storer and M. Cohn, editors, *Proc. 1996 IEEE Data Compression Conference*, pages 140–149. IEEE Computer Society Press, Los Alamitos, California, Apr. 1996.
- A. I. Wirth and A. Moffat. Can we do without ranks in Burrows Wheeler transform compression? In J. A. Storer and M. Cohn, editors, *Proc.* 2001 IEEE Data Compression Conference, pages 419–428. IEEE Computer Society Press, Los Alamitos, California, Mar. 2001.
- X. Wu, N. D. Memon, and K. Sayood. A context-based, adaptive, lossless/near-lossless coding scheme for continuous-tone images, June 1996. ISO Working Document ISO/IEC/SC29/WG1/N256.