A Two-Stage Modelling Method for Compressing Binary Images by Arithmetic Coding

Pasi Fränti and Olli Nevalainen

Department of Computer Science, University of Turku Lemminkäisenkatu 14 A, SF-20520 Turku, Finland

A two-stage modelling schema to be used together with arithmetic coding is proposed. The main motivation of the work has been the relatively slow operation of arithmetic coding. The new modelling schema reduces the use of arithmetic coding by applying to large white regions global modelling which consumes less time. This composite method works well and with a set of test images it took only about 41% of the time required by a QM-coder. At the same time the loss in compression ratio is only marginal.

Received November 1992, revised August 1993

1. INTRODUCTION

Pictorial information is expressed by a very simple model in black-and-white images. Only two colours, black and white, are recognized and even the greyness of different picture elements is omitted so that the image consists of a configuration of pixels each representing the pure black or white colour. In spite of the binary nature of the image files they have a high demand for storage space. This brings major problems to practical applications with image data in terms of long transmitting times and large space requirements for secondary storage. A standard solution to the space problem is the application of a compression–decompression system. This gives typically an order of magnitude of reduction in the size but simultaneously frustrates the user with a long processing time.

Data compression can be divided into two disjoint processes, modelling and coding [1, 9]. The modelling phase is highly application dependent and one should use a statistical model which takes into consideration the characteristics of the data. The coding (encoding and decoding) can be done optimally by the arithmetic coding technique [4, 16] which means that the compression efficiency depends on the quality of the modelling only.

The task of modelling binary images is theoretically clearcut because of the small size of the source alphabet. It is possible to use a relatively large prediction context (i.e. a set of neighbouring points) in this case and even more than one context can be used at the same time [12]. The prediction models can be classified to static and dynamic types. In a static model the conditional probabilities of symbol values are precalculated and they are the same for all parts of the image. In a dynamic model the probabilities are determined at the time of compression (and decompression). This is advantageous, in particular when the images consist of regions with strongly differing statistical characteristics.

The arithmetic coding suffers from a relatively large running time. Much of this originates from the type of modelling schemes used; the models are local in the sense they handle an image pixel by pixel. This means about 2 million encoding steps for normal A4-sized images with 150 d.p.i. resolution.

In the present paper we propose a new practical compression algorithm which is a hybrid of arithmetic coding and block coding. The latter compression technique [7] has turned out to be fast and give good (though suboptimal) compression ratios for regions with a high density of white pixels (or black pixels). Our idea is thus to gain speed at the expense of the compression result.

The state of art in the binary image compression with arithmetic coding is JBIG (Joint Bi-level Image Group), which is a draft for international standard by ISO (International Organization for Standardization)/IEC (International Electrotechnical Committee) and CCITT (Consultative Committee for International Telephony and Telephony) [6, 15 Section 20.3]. The main component of the algorithm is QM-order, a highly sophisticated arithmetic coder, which will be the point of comparison made in this paper later on.

Comparison of different compression methods is commonly done by analysing the compression ratio and running time. The first criterion is specified by the formula:

\[ \text{compression ratio} = \frac{\text{number of input bits}}{\text{number of output bits}} \]

Execution time is consumed by the coding and decoding phases. These two times are about the same for our technique and we thus show the coding times only. (In
fact the decoding times may be critical in some applications.)

We start in Section 2 by describing the two-phase compression algorithm. Analysis of the running time is found in Section 3. Variants of the arithmetic coding algorithms are briefly discussed in Section 4. Problems of local modelling and block coding are discussed, and an algorithm for a non-hierarchical variant is given in Section 5. Analysis of the non-hierarchical method is presented in Section 6. Section 7 contains a summary of test results and, finally, conclusions on the subject are drawn in Section 8.

2. COMBINING BLOCK CODING AND ARITHMETIC CODING

Our idea is to reduce the use of time-critical arithmetic coding by replacing it with block coding [7] as much as possible. Here we will use the two-dimensional hierarchical block coding. The following facts are known for these two compression methods:

1. Block coding is fast but its compression result is suboptimal.
2. Inefficiency of the hierarchical block coding does not appear at the upper levels of coding.
3. Arithmetic coding is space optimal but it runs slowly.

We use a two-phase modelling schema in the present paper. At the first phase the hierarchical block coding is applied. The encoding starts with the block size 16 × 16. If the whole block is white (i.e. all pixels in it are white) it is coded by a single 0-bit. Otherwise the block is non-white, including at least one black pixel. Such a block is coded by bit 1 and then divided into four equal sized sub-blocks which are then recursively encoded in the same manner.

Block coding continues until the block size reaches a predefined lower limit, jumplevel. At this moment the second phase is entered by changing the modelling schema to local modelling. Jumplevel can be any of the values 16 × 16, 8 × 8, 4 × 4, 2 × 2, 1 × 1. The block is now coded pixel by pixel by arithmetic coding. The present paper presupposes a dynamic modelling technique with an 8-pixel prediction context, see Figure 1.

The two-phase modelling schema thus results in a mixture of bit strings containing data of the block coding stage and local modelling stage. All these bits are coded by an arithmetic coder. For a description of the algorithm see Figure 2.

By the selection of the value of jumplevel we can control the compression ratio and the speed of the algorithm. A large jumplevel causes much local modelling and frequent use of arithmetic coding with high compression ratio but a large running time. On the other hand, a small jumplevel increases the speed at the cost of the compression gain.

3. ANALYSIS

Encoding can be seen as a three-step sequential process:

Step 1: Input.
Step 2: Divide the input into B × B blocks.\footnote{An earlier work on the subject has shown 16 × 16 to be a good choice for a starting block size [3].}
Step 3: Process the blocks.

In the following analysis Step 1 is ignored because it is always the same regardless of the Steps 2 and 3. Step 2 is done on the addressing level and its time is insignificant in comparison with the other two steps. The Step 3 can be divided into two substeps:

Step 3a: Check whether the block is white.
Step 3b: Code the block-bit (0 if the block is white, 1 otherwise).

Further processing is necessary for non-white blocks called also active blocks:

Step 3c: Divide the active block into four sub-blocks.
Step 3d: Process the sub-blocks recursively.

Step 3c is also considered to be non-time consuming as Step 2 was. We will next use the notations:

\(B\) block size parameter
\(J\) jumplevel size parameter
\(s_p\) average execution time to process a \(b \times b\)-block, where \(b = (B, B/2, \ldots, 1)\)

\begin{verbatim}
WHILE unprocessed rows of image exists DO
    Read next 16 lines of image into buffer.
    Split the lines into 16 × 16-pixel blocks.
    FOR each block DO
        IF block size = jumplevel THEN
            Encode the pixels of the block with arithmetic coding.
        ELSE
            Check whether the block is white by scanning the pixels.
            Encode the block code (0 = white, 1 = non-white) with arithmetic coding.
            IF the block is non-white THEN
                Split it into four equal sized sub-blocks.
                Process the sub-blocks recursively in the same way.
        END-IF
    END-FOR
END-WHILE
\end{verbatim}

\textbf{FIGURE 2.} Hybrid algorithm of hierarchical block coding and arithmetic coding.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{context.png}
\caption{Alternative prediction contexts.}
\end{figure}
\( p_b \) probability for the occurrence of a non-white \( b \times b \)-block (conditional)

\( u_b \) proportion of pixels needed to test in a \( b \times b \)-block on an average

\( W \) execution time to test the colour of a single pixel

\( A \) execution time to encode a bit with the arithmetic coding and updating the model

Let \( T_n \) be the total running time for an \( n \times n \) pixel image and assume that the image is in a square, i.e., the pixel matrix consists of \( \sqrt{n} \) rows and \( \sqrt{n} \) columns. Then the total running time of the coding is

\[
T_n = \left\lceil \frac{\sqrt{n}}{B} \right\rceil \times \left\lceil \frac{\sqrt{n}}{B} \right\rceil \times s_B
\]

where \( s_B \) is the average execution time for a \( B \times B \)-block.

Execution time for a \( b \times b \)-block \((b = B, B/2, \ldots, J)\) is:

\[
s_b = b^2 \cdot A \quad \text{if } b = J \quad \text{(jumplevel)}
\]

\[
s_b = 4 \cdot p_b \cdot s_{B/2} + u_b \cdot b^2 \cdot W + A \quad \text{if } b > J
\]

At jumplevel the execution time depends linearly on the number of pixels in the block \((b^2 \times A)\). At higher levels the time consists of checking the block's colour \((u_b \times b^2 \times W)\), coding the block-bit \((A)\) and possibly processing the four sub-blocks \((4 \times b \times s_{B/2})\). We assume that the time for coding a block-bit is the same as that of coding a pixel in arithmetic coding. For the jumplevel \( J \) we have \( z = \log_2(B/J) \) levels of recursion and thus from (2):

\[
s_B = 4p_Bs_{B/2} + u_Bb^2W + A
\]

\[
= [u_Bb^2 + 4p_Bu_{B/2}(B/2)^2 + 4^2p_Bp_{B/2}u_{B/4}(B/2)^4 + \ldots + 4^z-1(p_{B/8}p_{B/16} \ldots p_{B/2z-2})u_{B/2z-1}(B/2^{z-1})^2]W
\]

\[
+ [1 + 4p_B + 4^2p_Bp_{B/2} + \ldots + 4^z-1(p_{B/8}p_{B/16} \ldots p_{B/2z-2})]A
\]

\[
= [u_B + p_Bu_{B/2} + p_Bp_{B/2}u_{B/4} + \ldots + (p_{B/2}p_{B/4} \ldots p_{B/2z-2})u_{B/2z-1}]B^2W
\]

\[
+ [1 + 4p_B + 4^2p_Bp_{B/2} + \ldots + 4^z-1(p_{B/8}p_{B/16} \ldots p_{B/2z-2})]A
\]

\[
+ [p_{B/2}p_{B/4} \ldots p_{B/2z-1}]B^2A
\]

\[
= B^2W\sum_{i=0}^{z-1} u_{B/2^i} \sum_{j=0}^{i-1} p_{B/2^j} + \sum_{i=0}^{z-1} (p_{B/2^i})B^2A
\]

(3)

where \( \Pi_{j=0}^{i-1} \) is defined to be 1.

For the arithmetic coding the running time of a \( B \times B \)-block is \( B^2A \) and therefore the ratio of execution times of the pure arithmetic coding and the hybrid compression algorithm is

\[
\frac{s_B}{B^2A} = \frac{W}{A} \sum_{i=0}^{z-1} u_{B/2^i} \sum_{j=0}^{i-1} p_{B/2^j} + \sum_{i=0}^{z-1} \frac{4^i}{B^2} \sum_{j=0}^{i-1} p_{B/2^j} + \sum_{i=0}^{z-1} p_{B/2^i} - 1 = \frac{s_B}{B^2A}
\]

(4)

The pixel processing times \( W \) and \( A \) are machine dependent constants whereas \( p_b \) and \( u_b \) \((b = B/2^i)\) depend on the image to be compressed. Table 1 shows estimates for \( p_b \) and \( u_b \) \((u_b = \text{number of colour tests divided by } b^2)\) calculated from a set of test images, see Appendix. The number of pixels to be tested was 52–74% of all pixels. At the same time the proportion of non-white blocks \((p_b)\) was 36–67%. Notice that \( p_b \) stand for the conditional probability excluding the blocks belonging to larger all-white blocks and are thus already encoded.

If we let \( k = A/W \), we can write (4) in the form

\[
f_i(k) = \frac{s_B}{B^2A} = \frac{1}{k} \sum_{i=0}^{z-1} u_{B/2^i} \sum_{j=0}^{i-1} p_{B/2^j} + \sum_{i=0}^{z-1} \frac{4^i}{B^2} \sum_{j=0}^{i-1} p_{B/2^j} + \sum_{i=0}^{z-1} p_{B/2^i}
\]

(5)

The factor \( k \) is a machine dependent constant and obviously \( k > 1 \), i.e., coding of a pixel and updating the probability model is more time-consuming than finding out the colour of a pixel. In the case of 486/33 PC-compatible \( k \) would fall between 5 and 10. Table 2 shows formulas of \( f(k) \) for different selections of jumplevel when \( p \)- and \( u \)-estimates of Table 1 have been applied in (5).

Figure 3 shows values of \( f(k) \) for different selections of jumplevel. Values \( f(k) < 1 \) indicate that the hybrid method is faster than arithmetic coding. Composite modelling schema turns out to be faster than local modelling if \( k \) is greater than a limit, say \( k = 1.3 \). Another

<table>
<thead>
<tr>
<th>Number of blocks</th>
<th>8745</th>
<th>11834</th>
<th>31943</th>
<th>80139</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-white blocks</td>
<td>2958</td>
<td>7986</td>
<td>20035</td>
<td>49521</td>
</tr>
<tr>
<td>( p_b ) (%)</td>
<td>34</td>
<td>67</td>
<td>63</td>
<td>62</td>
</tr>
<tr>
<td>Number of colour tests</td>
<td>188.90</td>
<td>32.99</td>
<td>9.05</td>
<td>2.57</td>
</tr>
<tr>
<td>( u_b ) (%)</td>
<td>74</td>
<td>52</td>
<td>57</td>
<td>64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of blocks</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-white blocks</td>
<td>2958</td>
<td>7986</td>
<td>20035</td>
<td>49521</td>
<td></td>
</tr>
<tr>
<td>( p_b ) (%)</td>
<td>34</td>
<td>67</td>
<td>63</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>Number of colour tests</td>
<td>188.90</td>
<td>32.99</td>
<td>9.05</td>
<td>2.57</td>
<td></td>
</tr>
<tr>
<td>( u_b ) (%)</td>
<td>74</td>
<td>52</td>
<td>57</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2.** \( s_B \) values for the set of test images due to factor \( k \)

<table>
<thead>
<tr>
<th>Jumplevel ( J )</th>
<th>( f(k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1.00</td>
</tr>
<tr>
<td>8</td>
<td>0.34 + 0.74/k</td>
</tr>
<tr>
<td>4</td>
<td>0.24 + 0.91/k</td>
</tr>
<tr>
<td>2</td>
<td>0.17 + 1.04/k</td>
</tr>
<tr>
<td>1</td>
<td>0.15 + 1.13/k</td>
</tr>
</tbody>
</table>

**Table 1.** Statistics from test images 1 to 4 of the Appendix

<table>
<thead>
<tr>
<th>16 x 16</th>
<th>8 x 8</th>
<th>4 x 4</th>
<th>2 x 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of blocks</td>
<td>8745</td>
<td>11834</td>
<td>31943</td>
</tr>
<tr>
<td>Non-white blocks</td>
<td>2958</td>
<td>7986</td>
<td>20035</td>
</tr>
<tr>
<td>( p_b ) (%)</td>
<td>34</td>
<td>67</td>
<td>63</td>
</tr>
<tr>
<td>Number of colour tests</td>
<td>188.90</td>
<td>32.99</td>
<td>9.05</td>
</tr>
<tr>
<td>( u_b ) (%)</td>
<td>74</td>
<td>52</td>
<td>57</td>
</tr>
</tbody>
</table>
observation is that even relatively small $k$-values give a significant speed-up in the processing time.

Figure 4 shows the observed values of $f(k)$ from the test runs with images of the Appendix. The same figure shows also theoretical values for different $k$-values from (5). It is assumed that at the jumplevel 16 the theoretical and observed running times are equal. The difference between the curves for $k=10, 20, 100$ and the observed running times is relatively small which confirms the idea that the actual value of $k$ is large enough so that significant benefit from the block coding schema will be achieved.

4. SELECTING THE ARITHMETIC CODING ALGORITHM

The hybrid modelling schema allows the use of any version of the arithmetic coding. At least seven different algorithms were available beginning with a general version by Witten et al. [20], a fixed-length version of arithmetic coding by Teuhola and Raita [19], Skew-code by Langdon and Rissanen [8], Q-coder by Pennebaker et al. [10, 11, 13, 14], its newer improved version QM-coder, [15], Quasi-arithmetic coding by Howard and Vitter [5] and few others [2, 17]. Two of these were implemented and tested.

The first one chosen was the version by Witten et al. [20] referred as A. There are several reasons for choosing this algorithm. It is optimal, relatively clear, it is easy to embed into our algorithm and the source code was available. The algorithm is general in the sense that it allows the use of a multi-alphabet source. However, our compression algorithm requires a binary alphabet only, so in the C-language implementation we modified the source code to gain advantage of this fact.

QM-coder [6, 15], abbreviated QM, has been specially tailored for binary images and one of the primary aspects in its design has obviously been the speed. For example, all multiplication operations have been replaced by fast approximations or by shift-left-operations. Therefore QM-coder clearly outperforms algorithm A for speed. Moreover, QM-coder is the arithmetic coding component in both JBIG and JPEG standards [15].

QM-coder includes its own modelling procedures, which makes the linking to block coding somewhat unconventional (Figure 5). The modelling phase determines the context to be used and the binary decision to be coded. QM-coder then picks up the corresponding probability, performs the actual coding and updates the probability distribution if necessary. The way QM-coder handles the probabilities is based on an approximation algorithm and the method adapts quickly to local variations in the image. For details see [15].

5. THE PREDICTION MODELLING

Images are usually processed in row major order from left to right. This has an influence on design of the prediction context. One can use in the prediction of a particular pixel only pixels that have already been encoded. On the other words, each pixel in context must stand on the same row to the left of the pixel to be coded or above it.

A different order of processing is applied in hierarchical block coding. Here the blocks are visited in the so-called Morton-order (Z-pattern) [18] (Figure 6). This sequence will recursively repeat itself in every sub-block. Morton-order has the property that each block (or pixel) on the north-west of a current block has been visited earlier. However, the block on the north-east may still contain some pixels which have not yet been encoded. The original 8-pixel model cited earlier contains two such pixels (marked by * in Figure 7). This means that the coding algorithm in Figure 2 is irreversible.

There are two possible ways to solve the problem.
16 row buffer is handled traditionally row by row and every pixel is encoded with arithmetic coding by the original 8-pixel context model if and only if the pixel belongs to an active block.

The test results in Section 7 will show that the best overall performance, when both running time and compression ratio are considered, is reached for prolevel 8. For these reasons we will not give the algorithm for the general case, but instead the algorithm is tailored for prolevel 8. We call this algorithm the non-hierarchical variant of block coding because no recursion is needed in it (Figure 8).

6. ANALYSIS OF THE NON-HIERARCHICAL VARIANTS

The running time for the non-hierarchical variant is got as a special-case of formula (3)

\[
s_b = p_b B^2 A + u_b B^2 W + A + BW + S
\]

(6)
The last two terms originate from the two-stage encoding process. When implementing the algorithm of Figure 8, we note that for each block one extra SET-clause is needed (the factor S). Another additional cost (BW) is caused by the IF-clause at the stage 2 which is executed once for each of the B slices in a B x B-block. The ratio of the running times of the non-hierarchical two-phase modelling and one-phase arithmetic coding is

\[
f_2(k) = \frac{s_b}{B^2 A} \approx p_b + \frac{1}{B^2} + \frac{1}{k} \left( u_b + \frac{1}{B} \right)
\]

(7)
where we have omitted the term S/B^2 A.
The worst case for the algorithm is an image where every block is non-white, but each of them contains only one black pixel which is the last one checked. Then we have \( p_b = 1 \) and \( u_b = 1 \), and for this very rare case:

\[
f_2(k) = 1 + \frac{1}{B^2} + \frac{1}{k} \left( 1 + \frac{1}{B} \right)
\]

(8)

**WHILE** unprocessed rows of image exists **DO**

Read the next 16 lines of image into buffer.

Split the lines into 16 x 16-pixel blocks, \( B_i \) (i = 1...m).

For i = 1 to m **DO** (**Stage 1**)

Check whether the block \( B_i \) is white by scanning the pixels.

IF \( B_i \) is white **THEN**

SET White[i] = TRUE.

ELSE

SET White[i] = FALSE.

Encode the block code (0 = white, 1 = non-white) with arithmetic coding.

FOR each row of the input buffer **DO** (**Stage 2**)

Split the row horizontally into 16-pixel slices.

FOR each slice **DO**

IF the slice belongs into an active block **THEN**

Encode the pixels of the slice with arithmetic coding.

END-**WHILE**

**FIGURE 8.** Hybrid algorithm of non-hierarchical block coding and arithmetic coding.
Thus for the k-values appearing in practice, the running time of the worst case is only slightly longer than the time for the arithmetic coding (Figure 9). As shown in the same figure the variant is extremely fast for all-white images. For the set of test images it used about 40% of the time of arithmetic coding.

7. SUMMARY OF TEST RESULTS

The compression methods were implemented in Turbo-C language in an MS-DOS environment with a 486/33 PC-compatible. The test runs were performed for all jumplevels (1, 2, 4, 8, 16) with each of the test images.

The hierarchical variant was tested with the arithmetic coding algorithm A. Figure 10 shows the dependency of the compression ratio and running time when jumplevel parameter was varied. An ideal selection of jumplevel turns out to be 8, since it gives a high compression ratio at a relatively high speed. (Superiority of jumplevel yet depends on how much one emphasizes compression ratio and how much speed.)

Compression efficiency and running time of the non-hierarchical variant are given in Table 4 for both versions of arithmetic coding. As to the compression efficiency the results with arithmetic coding by Witten et al. [20] (A and BA) are similar to those with QM-coder (Q and BQM). Difference in the speed is more significant. QM-coder outperforms A as expected, however the benefit of the composite method is almost the same with both versions of arithmetic coding (BA/A ≈ 0.38, BQM/QM ≈ 0.41).

Notice that the running time depends on the proportion of the white blocks in the image. An all-white image is the best case for the two-stage modelling schema and BQM algorithm took only 2 s for an A4-size all-white image (1275 × 1745 pixels). On the other hand, for an all-black image the time was 41 s, i.e. the same as for the local modelling schema.

A similar idea to ours is the Typical Prediction component in JBIG, which skips the lines that are equal to the previous one [15]. This means that successive all-white lines can be coded with less computation, just like in our method. In fact, white line skipping can be considered as a special case of our non-hierarchical variant, in which the block size is one pixel high but as long as the width of the image. Although it gives moderate benefit for the test image 1, it is poor for image 2 and has practically no effect on the images 3 and 4 (Table 5).

Different algorithms are compared in Figure 11. Here G4 stands for the CCITT Group IV standard compression algorithm, also known as READ-code [22]. The results for G4 are got by compressing the images into TlFF-format by shareware software called Image Alchemy [21].

| TABLE 4. Summary of the test results for the non-hierarchical variant |
|--------------------------|------------------|------------------|-----------------|-----------------|
|                          | Image 1          | Image 2          | Image 3          | Image 4          |
| **Compression ratios**   |                  |                  |                  |                  |
| A                        | 29.42            | 16.26            | 21.86            | 7.22             | 18.69 |
| BA                       | 28.54            | 15.93            | 21.08            | 7.30             | 18.21 |
| QM                       | 29.94            | 16.31            | 22.19            | 7.46             | 18.98 |
| BQM                      | 29.09            | 16.02            | 21.47            | 7.49             | 18.52 |
| **Running time (s)**     |                  |                  |                  |                  |
| A                        | 62.1             | 64.0             | 63.8             | 65.2             | 63.8  |
| BA                       | 15.4             | 24.6             | 22.0             | 35.1             | 24.3  |
| QM                       | 38.6             | 39.9             | 39.7             | 40.5             | 39.7  |
| BQM                      | 10.9             | 16.5             | 14.8             | 22.6             | 16.2  |

A = Local modelling + arithmetic coding [15].
BA = Two-stage modelling + arithmetic coding [15].
QM = Local modelling + QM-coder.
BQM = Two-stage modelling + QM-coder.

| TABLE 5. Proportion of white pixels skipped by different methods |
|--------------------------|------------------|------------------|-----------------|-----------------|
|                          | Image 1          | Image 2          | Image 3          | Image 4          |
| All-white lines (%)     |                  |                  |                  |                  |
| 35.0                     | 14.2             | 1.8              | 0.2              | 12.8             |
| All-white blocks (%)    |                  |                  |                  |                  |
| 79.4                     | 65.7             | 69.7             | 50.4             | 66.3             |

8. CONCLUSIONS

A new two-stage modelling schema was presented for binary image compression. Combining the block coding as global modelling and the 8-pixel Markov model as local modelling turned out to be an efficient way to reduce the use of time-consuming arithmetic coding: fewer pixels remain to be coded. Block coding can be used hierarchically as seen in Section 2, but it causes problems in local modelling. On the other hand, non-hierarchical block coding was the best choice of all.

The block coding is used only in the modelling, so arithmetic coding remains an autonomous phase. Therefore, different versions of arithmetic coding can be used and even QM-coder fits for this purpose. The proportional benefit gained by the composite method is the same with the arithmetic coding by Witten et al. [20] and with QM-coder.

The two-stage method works better the more white there is in an image. The method is as slow as the pure arithmetic coding when compressing all-black images. An all-white image is the best case for the algorithm, only a fraction of the time required by the arithmetic coding is then needed for the composite method.

The composite method can still be improved and work on this subject is in progress. Block codes were compressed by the zero-order Markov model. We can naturally extend this model to higher orders. For example, we get an improvement of about 1.3% in the compression ratio with a first-order model (using the previously coded block as a context).

Another improvement is possible by classifying the blocks into three classes: all-white, all-black and mixed blocks. This involves two binary decisions when coding the block codes and it may have an effect on the compression ratio. An advantage of this is the capability of fast coding of blocks containing lots of black pixels.

REFERENCES
