Signal Processing: *Image Communication* 4 (1992) 103–111
Elsevier

Technical features of the JBIG standard for progressive bi-level image compression

Horst Hampel[1], Ronald B. Arps[2], Christos Chamzas[3], David Dellert[4], Donald L. Duttweiler[5], Toshiaki Endoh[6], William Equitz[2], Fumitaka Ono[7], Richard Pasco[2], Istvan Sebestyen[8], Cornelius J. Starkey[9], Stephen J. Urban[10], Yasuhiro Yamazaki[6] and Tadashi Yoshida[11]

[1] *Alcatel, SEL Research Center, Pforzheim, Germany*
[2] *IBM, Almaden Research Center, San Jose, USA*
[3] *Democritus University of Thrace, Xanthi, Greece*
[4] *Eastman Kodak Company, Rochester, USA*
[5] *AT&T Bell Laboratories, Holmdel, USA*
[6] *KDD, R&D Laboratories, Kamifukuoka-shi, Japan*
[7] *Mitsubishi, Communication Systems Laboratory, Kamakura-shi, Japan*
[8] *Siemens AG, Private Communication Systems, Munich, Germany*
[9] *DataBeam, Lexington, USA*
[10] *Delta Information Systems, Horsham, USA*
[11] *CANON, Research Center, Kawasaki-shi, Japan*

Abstract. The JBIG coding standard like the G3 and G4 facsimile standards defines a method for the lossless (bit-preserving) compression of bi-level (two-tone or black/white) images. One advantage it has over G3/G4 is superior compression, especially on bi-level images rendering greyscale via halftoning. On such images compression improvements as large as a factor of ten are common. A second advantage of the JBIG standard is that it can be parameterized for progressive coding. Progressive coding has application in image databases that must serve displays of differing resolution, image databases delivering images to CRT displays over medium rate (say, 9.6 to 64 kbit/s) channels, and image transmission services using packet networks having packet priority classes. It is also possible to parameterize for sequential coding in applications not benefiting from progressive buildup. It is possible to effectively use the JBIG coding standard for coding greyscale and color images as well as bi-level images. The simple strategy of treating bit-planes as independent bi-level images for JBIG coding yields compressions at least comparable to and sometimes better than the JPEG standard in its lossless mode. The excellent compression and great flexibility of JBIG coding make it attractive in a wide variety of environments.

Keywords. Facsimile, arithmetic coding.

1. Introduction

The Joint Bi-level Image Experts Group (JBIG) is under the auspices of ISO-IEC/JTC1/SC29/WG9 and CCITT/SGVIII/Q16. It was formed in 1988 to establish a standard for the progressive coding of bi-level (two-tone or black/white) images. As of this writing its work is nearing completion and an ISO CD ballot has begun [10, 11, 22].

Like the one-dimensional modified Huffman (MH) and the two-dimensional modified READ (MR) and modified–modified READ (MMR) coding schemes of CCITT recommendations T.4 [3, 8] (G3) and T.6 [4] (G4 and G3), JBIG coding is lossless (bit-preserving). Hence there is no image quality issue. Decoded images are digitally identical to the images at the encoder input.

One advantage of JBIG coding over G3/G4 coding is superior compression. On images

containing text and/or line art JBIG compression is generally 1.1 to 1.5 times that of MMR coding, the most efficient of the G3/G4 techniques. On bi-level images rendering greyscale via halftoning JBIG's compression ratio advantage typically increases to a factor of 2 to 30.

A second advantage of the JBIG standard is that it can be parameterized for progressive coding. Progressive coding is defined and applications for it are cited in the next section.

It is also possible to effectively use the JBIG standard for the lossless coding of greyscale and color images. The simple strategy of coding bit-planes as independent bi-level images can yield good compression. There are various ways in which such bit-planes might be created. One good choice for greyscale images is to let the bits of a folded-binary (Gray) representation [7] of intensity define bit-planes. If this is done, compression ratios at least comparable to those of lossless JPEG [9] coding are obtained. If the intensity resolution is coarse, say, less than 8 bits, JBIG compression can be significantly better.

The JBIG approach to lossless greyscale and color image coding offers coding unification. One underlying algorithm efficiently serves for coding bi-level images, greyscale photographic images, color photographic images and computer-generated images with bit-plane overlays.

In the remainder of this paper we will restrict attention to bi-level images not because the extensions for greyscale and color are seen as unimportant, but rather because they are so straightforward.

A JBIG encoder working progressively needs to generate half-resolution versions of images. The JBIG standard describes an algorithm called PRES (progressive reduction standard) that can be used for this purpose. The half-resolution images it generates are of excellent subjective quality and far superior to those generated by simply subsampling every other row and column. This is true for both images containing text and/or line art and images containing greyscale rendered by halftoning or dithering. The PRES algorithm will be useful in its own right whenever half-resolution images are needed.

2. Coding modes

2.1. Progressive coding

When decoding an image that has been progressively encoded, a low-resolution rendition of the original is made available first with subsequent doublings of resolution as more data is decoded.

One application for progressive coding is in image databases serving output devices with widely differing resolution capability. Using a non-progressive coding technique like MMR requires that images be stored as compressions of the image at the resolution of the highest-resolution device to be served, say, a laser printer. When an image is to be displayed on a low-resolution device like a CRT, the compressed image must be decoded to high resolution and then mapped to low resolution, often by simply discarding $N-1$ of every N rows and columns. There are obvious transmission and processing inefficiencies. Additionally, if the mapping to low resolution is via the above subsampling technique, the subjective quality of the displayed image is worse than it need be for the resolution available.

Storing the database images in progressive form solves both problems. Only that information in the compressed image required for reconstruction to the resolution of the display is transmitted and decoded and the displayed image has subjective quality nearly as good as is possible for the display resolution available.

If after viewing the image at low resolution additional resolution enhancement is desired for, say, a paper copy, only the needed updating information has to be additionally sent and processed.

Another application for progressive coding is in image browsing over medium rate communication links. A low resolution rendition can be rapidly transmitted and displayed, and then followed by as much resolution enhancement as is desired. Each stage of resolution enhancement builds on the previous stage. Progressive coding makes it easy

for a user to quickly recognize the image being displayed, which in turn makes it possible for that user to quickly interrupt the transmission of an unwanted image.

This advantage for progressive coding only occurs on medium rate links, roughly those with speeds between 9.6 and 64 kbit/s when bi-level images are being retrieved. If the communication link is slower, no viewer will have the patience needed for image browsing no matter what the form of presentation. On high speed links the image comes so fast relative to human reaction times that how it develops is immaterial.

A third application for progressive coding is in packet networks where packets can or must be priority classified as droppable or non-droppable. Priority classification is being considered for broadband ISDN. The packets carrying the information for the final resolution doubling would be sent at low priority and if they had to be dropped the only penalty would be a slightly less sharp image. No entire regions would be lost or destroyed.

The resolution R_D of the top layer in a progressive hierarchy is not restricted. Choices such as 400 or 200 dots per inch (dpi) provide a hierarchy of resolutions commensurate with current facsimile standards. Choosing R_D as 600 or 300 dpi gives a progressive hierarchy more compatible with laser printer resolution available as of this writing.

When progressive coding is desired, it is anticipated that D, the number of resolution doublings, will usually be chosen so that the bottom layer resolution is roughly 10 to 25 dpi. Typical bi-level images when reduced to such a resolution are not legible, but nonetheless such low-resolution renditions are still quite useful in that they function as automatically generated icons. Page layout is usually apparent and recognition of particular pages that have been seen before at higher resolution is often possible.

2.2. Compatible progressive/sequential coding

Compatible progressive/sequential coding is a feature of the JBIG standard that can be quite useful, but unfortunately it is often somewhat difficult to understand at first. It is of primary interest for the application of a database serving displays of different resolution. Progressive coding is desirable to efficiently serve the various displays, but hardcopy displays have no need for the lower resolution renditions developed along the way in decoding to their final resolution. JBIG exploits this fact to eliminate the usual progressive-coding need for a frame buffer at the next-to-the-highest resolution (R_{D-1}).

Table 1
Possible data orderings

HITOLO	SEQ	Example order
0	0	0, 1, 2 3, 4, 5 6, 7, 8
0	1	0, 3, 6 1, 4, 7 2, 5, 8
1	0	6, 7, 8 3, 4, 5 0, 1, 2
1	1	6, 3, 0 7, 4, 1 8, 5, 2

Compatible progressive/sequential coding is achieved by breaking an image into smaller parts before compression. These parts are created by dividing the image in each of its resolution layers into horizontal bands called stripes. Typically the stripes are quite small and 30 to 40 make up the full image. For simplicity though, Fig. 1 shows stripe decomposition when there are only 3 stripes. It makes the further assumption that there are three resolution layers, which is again a somewhat smaller than typical number.

Each stripe s at each resolution d is coded into a subfile $C_{s,d}$. The JBIG file to describe the total image is a concatenation of header information and the $C_{s,d}$ subfiles. Four ways of concatenating the stripe codings are defined in Table 1.

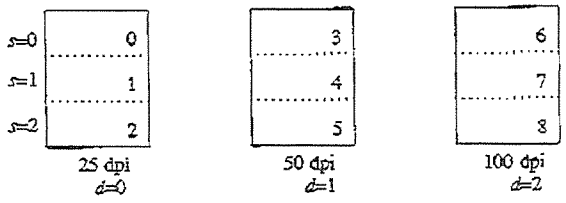

Fig. 1. Stripe decomposition with 3 layers and 3 stripes.

An encoder likes to work from high resolution down, so in normal progressive operation it processes stripes in the second-last order shown in the table. However, alternatively it could choose to proceed 'stripewise' as in the last row. An advantage in doing so is that there is no need for a 50 dpi frame buffer. A 'stripe buffer' now serves the same purpose. Here that stripe buffer is 1/3 the size of a frame buffer, but more typically it is just 1/30 to 1/40 of a frame buffer and the saving is substantial. Decoders work naturally from low resolution up and so prefer the first two orderings of the table.

The key point is that the total information in the JBIG compressed file is the same for all data orderings. Decoders working in normal progressive fashion as well as decoders working in compatible sequential fashion are both supported from the same database. Moreover, the decoders neither know nor care about the order in which the subfiles were concatenated by the encoder for transmission to the database.

2.3. *Single-layer coding*

The JBIG specification does not restrict the number D of resolution doublings. It can be set to 0 if progressive coding is of no utility, as is the case, for example, in hardcopy facsimile. Doing so retains JBIG's compression advantage over G3/G4 (and, in fact, usually increases it somewhat), while eliminating the need for any buffering and simplifying the algorithm somewhat. Single-layer JBIG coding has potential applications identical to those of G3/G4 coding. Images compressed by a single-resolution-layer encoder will be readable by decoders capable of progressive decoding, although of course only the lowest resolution version of a progressively encoded image will be decodable by a single-resolution-layer decoder.

3. Functional blocks

We will describe some of the main functional blocks of an encoder. Since decoders are similar and in fact somewhat simpler because resolution reduction is not needed, they will not be described. Also, to make the presentation as simple as possible, decomposition into stripes will be ignored by assuming there is only one stripe per image. Full detail is of course in the specification.

Conceptually a JBIG encoder can be decomposed (see Fig. 2) into a chain of D differential layer encoders followed by a bottom-layer encoder. In Fig. 2, I_d denotes the image at layer d and C_d denotes its encoding. A hardware implementation would in all likelihood time-share one physical differential layer encoder, but for heuristic purposes the decomposition of Fig. 2 is helpful.

Each differential layer encoder can be decomposed into the functional blocks shown in Fig. 3.

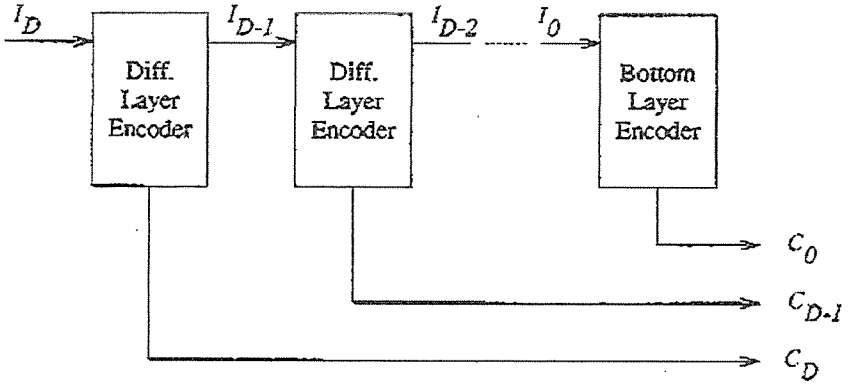

Fig. 2. Decomposition of an encoder.

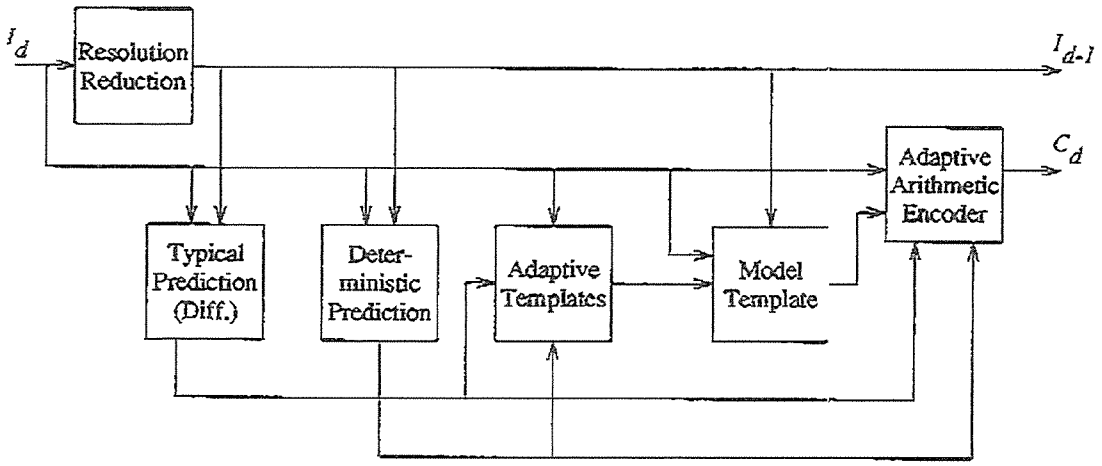

Fig. 3. Differential layer encoder.

The bottom-layer encoder has the somewhat simpler decomposition of Fig. 4.

3.1. Resolution reduction

The resolution-reduction block accepts a high-resolution image and creates a low-resolution image with, as nearly as possible, half as many rows and half as many columns as the original. (There need not in general be an even number of rows and columns in the input image.) The particular resolution-reduction method suggested by the JBIG standard has been carefully designed and extensively tested [23]. As noted earlier, it creates excellent quality low-resolution renditions for text, line art, dithered greyscale, halftoned greyscale and error-diffused greyscale.

It is a table based algorithm. The low resolution image is created pixel by pixel in the usual raster scan order, that is, from top to bottom and left to right. The color of any given low resolution pixel is uniquely determined by the colors of nine particular high-resolution neighbors that are in fixed spatial relationship to it and three particular low-resolution neighbors that are in causal and fixed spatial relationship to it.

3.2. Differential-layer typical prediction

The differential-layer typical prediction (TP) block [5] provides some coding gain, but its primary purpose is to speed implementations. Differential-layer TP looks for regions of solid color and when it finds that a given current high-

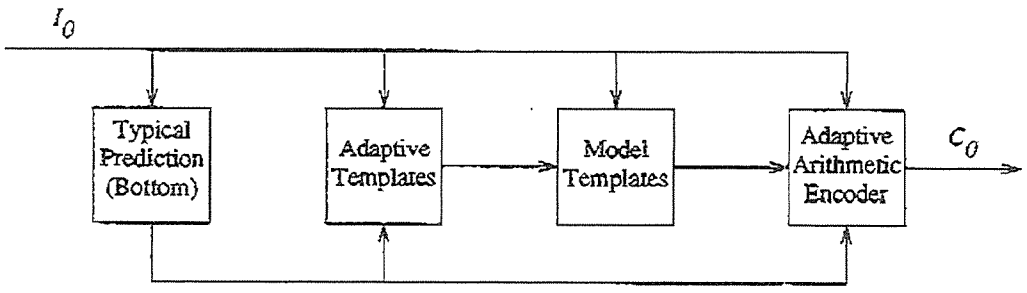

Fig. 4. Bottom-layer encoder.

resolution pixel for coding is in such a region, none of the processing normally done in the deterministic prediction, adaptive templates, model templates and arithmetic coding blocks is needed. On text or line-art images, differential-layer TP usually makes it possible to avoid coding over 95% of the pixels. On bi-level images rendering greyscale, processing savings like this are not possible.

The key idea behind differential-layer TP is that if all the pixels in an eight-neighborhood of a low-resolution pixel are the same color, then it is extremely likely that all four high-resolution pixels to be associated with it are that same color. Unfortunately, it is not certain that this is so, but exceptions occur infrequently enough that it is efficient and reasonable to flag them. In particular, an encoder notes at the beginning of each high-resolution line pair whether or not a decoder would ever go wrong on that line pair if it always were to 'typically expand' any low resolution pixels it found to be within a common-color eight-neighborhood into four high-resolution pixels of that color. The failure or success of this strategy for the line pair is coded and sent to the decoder. Note that 'failure' here is not that some low-resolution pixel in the line pair is not within a common-color eight-neighborhood, but rather that some low-resolution pixel in the line pair and in a common-color eight-neighborhood is not to be associated with four high-resolution pixels of that color. Failure in this sense is extremely rare and on many images never occurs. When success is coded, both the encoder and decoder skip over high-resolution pixels associated with any low-resolution pixels found to be within a common-color eight-neighborhood. If failure must be coded, the only penalty is that no skipping can be done for the line pair and everything must be coded.

3.3. Bottom-layer typical prediction

Typical prediction in the bottom layer is quite different from that in the differential layers. The algorithm used is a line-skipping algorithm. A given line is said to be 'typical' and all its pixels are declared 'typical' if it is identical to the line above it. Which lines are typical is again transmitted to the decoder. Both the encoder and decoder skip the coding of all pixels in typical lines and generate them instead by line duplication.

Bottom-layer TP like differential-layer TP is primarily intended to speed processing. It is not, however, able to achieve as high a percentage of skipped pixels. On images with text and line art, bottom-layer TP allows skipping about 40% of the pixels.

3.4. Deterministic prediction

The purpose of the deterministic-prediction (DP) block [20] is to provide coding gain, typically about 7%. When images are reduced in resolution by a particular resolution-reduction algorithm, it sometimes happens that the value of a particular current high-resolution pixel to be coded is inferable from the pixels already known to both the encoder and decoder, that is, all the pixels in the low-resolution image and those in the high-resolution image that are causally related (in a raster sense) to the current pixel. When this occurs, the current pixel is said to be deterministically predictable. The DP block flags any such pixels and inhibits their coding by the arithmetic coder.

DP is a table driven algorithm. The values of particular surrounding pixels in the low-resolution image and causal high-resolution image are used to index into a table to check for determinicity and, when it is present, obtain the deterministic prediction. DP tables are highly dependent on the particular resolution reduction method used. Provision is made for an encoder to download DP tables to a decoder if it is using a private resolution reduction algorithm. Decoders are required to always know the DP tables useful for the suggested resolution reduction scheme. Hence, if the suggested resolution reduction algorithm is used, no DP table need be sent.

3.5. Model templates

For each high-resolution pixel to be coded, the model-templates block provides the arithmetic

coder with an integer called the context. For differential-layer coding this integer is determined by the colors of six particular pixels in the causal high-resolution image, by the colors of four particular pixels in the already available low-resolution image, and by the spatial phase of the pixel being coded. The term spatial phase denotes which of the four possible orientations the high-resolution pixel has with respect to its corresponding low-resolution pixel. The six particular high-resolution pixels and four particular low-resolution pixels whose colors (along with spatial phase) define the context are known as the coding template or model template.

The arithmetic coder maintains for each context an estimate of the conditional probability of the symbol given that context. The greatest coding gain is achieved when this probability estimate is both accurate and close to 0 or 1. Thus good templates have good predictive value, so that when the values of the pixels in the template are known, the value of the pixel to be coded is highly predictable.

For bottom-layer coding, the coding template only includes high-resolution pixels. There are no low-resolution pixels to incorporate nor is there any analog of the spatial-phase concept in differential-layer coding. A JBIG parameter allows choosing between two different bottom-layer coding templates both of which have ten pixels. In the first, three pixels are from the line two above the pixel being coded, five are from the line immediately above it, and the remaining two are the two immediately preceding it on the current line. This template is referred to as the three-line bottom-layer template. The alternative is a two-line template with six pixels from the line immediately above and four from the current line. Some software implementations may execute somewhat faster with the two-line template. The price paid for the faster execution is about a 5% loss in coding efficiency.

3.6. Adaptive templates

The adaptive-templates (AT) block provides substantial coding gain, sometimes as much as a factor of two, on images rendering greyscale with halftoning. AT looks for a horizontal periodicity in the image and on finding it changes the template so that the pixel preceding the current pixel by exactly this periodicity is incorporated into it. Such a pixel has excellent predictive value.

Any such changes should be made infrequently, and when one occurs, a control sequence is multiplexed into the output data stream. Hence, decoders need not do any processing to search for the correct setting for AT.

How an encoder determines when a template change is desired and where the AT pixel should be placed is a private matter for it since the decoder tracks via the control information. However, one particular algorithm for determining the when and where for AT pixel movements is suggested in the JBIG standard. This particular algorithm uses a bank of counters. Each counter has a number called a 'lag' associated with it and counts the number of color coincidences between pairs of pixels horizontally separated by that particular lag. Hence an approximation to the autocorrelation function is created. When the count at some lag is unusually high and close to the maximum possible, it is likely that the image is halftoned and an AT change to incorporate a pixel at that lag would be beneficial.

The JBIG specification defines controls structures allowing vertical as well as horizontal movement of the AT pixel. However, algorithms to control vertical movement have not yet been investigated.

3.7. Arithmetic coder

Arithmetic coders are entropy coders, that is, they are coders that obtain a coding advantage by exploiting the fact that the symbols to be coded are not equally likely. What distinguishes arithmetic coders from other forms of entropy coders is that, conceptually at least, they are mapping a string of input symbols into a real number x on the unit interval. What is transmitted or stored instead of the original sequence of symbols is the binary expansion of x.

The particular number x to which the input sequence maps is determined by recursive probability interval subdivision as in the Elias coder [1]. Figure 5 shows an example of such interval division through an initial sequence 0, 1, 0, 0 to be coded.

The portion of the unit interval on which x is known to lie after coding an initial sequence of symbols is known as the current coding interval. For each binary input the current coding interval is divided into two sub-intervals with sizes proportional to estimates of the relative probabilities of symbol value occurrences. The new current coding interval becomes the portion of the old coding interval that is associated with the symbol value that actually occurred. The fact that most frequently the symbol to be coded will be the more probable one and that this is coded onto the larger of the two sub-intervals means that most of the time the coding interval is not reduced in size by as much as 1/2 and an input symbol is coded without generating as much as one bit in the binary expansion of x. It is true that whenever the less-probable symbol does happen to occur more than one output bit is generated, but the central result of information theory is that on the average there is still less than one bit generated for each input symbol.

The Elias coder described above was conceived soon after Shannon's seminal paper giving birth to Information Theory in 1948 [19]. It was of little practical use, however, until ways were found in the mid 1970s to perform all the necessary arithmetic with finite precision arithmetic and in a pipeline fashion so that a decoder could start its processing without having to wait for an encoder to finish [14, 17, 18, 21].

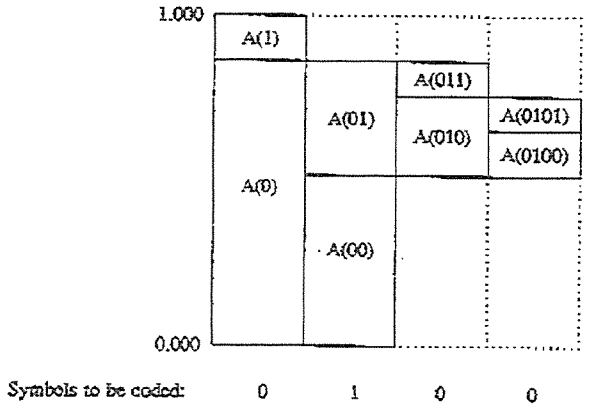

Fig. 5. Interval subdivision.

In some arithmetic coding applications the conditional probabilities with which the input symbols take on their two values are known a priori and fixed. More usually though they are not known a priori and may even be time varying and hence must be estimated on the fly. This is true in the JBIG application. Adaptively estimating probabilities for each of the contexts is a non-trivial statistical problem [6, 15]. A balance must be struck between obtaining extremely accurate estimates and the conflicting need of adapting quickly to changing underlying statistics.

The arithmetic problems associated with interval subdivision and the problem of adaptively estimating probabilities are conceptually disjoint. By merging them, however, it is possible to gain implementation efficiency by exploiting some of the variables created to perform finite precision arithmetic to also help with adaptive probability estimation [2, 12, 13, 16].

The arithmetic encoder of Fig. 3 notes the outputs of the TP and DP blocks to determine if it is even necessary to code a given pixel. Assuming it is, it then notes the context and uses its internal probability estimator to estimate the conditional probability that the current pixel will be a given color. Often the pixel is highly predictable from the context so that the conditional probability is very close to 0 or 1 and a large entropy coding gain is realized. The arithmetic encoder of Fig. 4 operates similarly, but has no bottom-layer analog to the output of the DP block.

The JBIG arithmetic coder is identical to the arithmetic coder used by the JPEG standard in some of its modes of compression.

4. Implementation

Hardware implementations of the full JBIG algorithm are underway. Implementing JBIG takes

more circuitry than implementing the MH, MR and MMR algorithms of G3 and G4, but a single-chip realization still appears reasonable. Coding speeds comparable to those of available devices for G3/G4 coding (50 to 100 mega-pixels per second) look possible.

The JBIG algorithm has now been implemented in software by many different programmers. Because of its adaptive nature, it appears that JBIG software will never execute quite as fast as G3/G4 software. Experiments to date show G3/G4 to have a speed advantage of from 2.5 to 10. Even so, JBIG coding in software can be fast and for many applications it will be acceptably fast. RISC type microprocessors executing compiled code have decoded typical 200 dpi typewritten pages in less than 2 seconds.

5. Acknowledgments

The JBIG algorithm is a result of committee work carried out by JBIG with the support and review of ISO-IEC/JTC1/SC2-29/WG8-9 and CCITT/SG/VIII. Numerous members of all these groups have spent countless hours and tremendous effort to make this collaborative international standardization effort successful. The authors wish to acknowledge the significant contribution of all members of all these groups in the development of this standard and regret that space does not permit individual recognition.

References

[1] N. Abramson, *Information Theory and Coding*, McGraw-Hill, New York, 1963, pp. 61–62.

[2] R.B. Arps, T.K. Truong, D.J. Lu, R.C. Pasco and T.D. Friedman, "A multi-purpose VLSI chip for adaptive data compression of bilevel images", *IBM J. Res. Development*, Vol. 32, No. 6, November 1988, pp. 775–795.

[3] CCITT Recommendation T.4, Standardization of Group 3 facsimile apparatus for document transmission.

[4] CCITT Recommendation T.6, Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus.

[5] C. Chamzas and D.L. Duttweiler, "Encoding facsimile images for packet-switched networks", *IEEE J. Sel. Areas Comm.*, Vol. 7, No. 5, June 1989, pp. 857–864.

[6] C. Chamzas and D. Duttweiler, "Probability estimation in arithmetic coders", in preparation.

[7] R.W. Hamming, *Coding and Information Theory*, Prentice Hall, Englewood Cliffs, NJ, 1980, pp. 96–98.

[8] R. Hunter and A.H. Robinson, "International digital facsimile coding standards", *Proc. IEEE*, Vol. 68, No. 7, July 1980, pp. 854–867.

[9] ISO Committee Draft 10918-1, Digitial compression and coding of continuous-tone still images—Part 1: Requirements and guidelines.

[10] ISO Committee Draft 11544, Coded representation of picture and audio information—Progressive bi-level image compression.

[11] S. Okubo, T. Omachi and F. Ono, "International standardization on picture coding", *IEICE Trans.*, Vol. E.74, No. 3, March 1991, pp. 533–539.

[12] F. Ono, S. Kino, M. Yoshida and T. Kimura, "Bi-level image coding with MELCODE—Comparison of block type code and arithmetic type code", *IEEE Global Telecomm. Conf.*, November 1989, pp. 255–260.

[13] F. Ono, M. Yoshida, T. Kimura and S. Kino, "Subtraction-type arithmetic coding with MPS/LPS conditional exchange", *Ann. Spring Conf. IEICE*, D-288, 1990 (in Japanese).

[14] R. Pasco, Source coding algorithms for fast data compression, Ph.D. thesis, Department of Electrical Engineering, Stanford University, 1976.

[15] W.B. Pennebaker and J.L. Mitchell, "Probability estimation for the *q*-coder", *IBM J. Res. Development*, Vol. 22, No. 6, November 1988.

[16] W.B. Pennebaker, J.L. Mitchell, G.G. Langdon, Jr. and R.B. Arps, "An overview of the basic principles of the *q*-coder adaptive binary arithmetic coder", *IBM J. Res. Development*, Vol. 32, No. 6, November 1988, pp. 717–726.

[17] J.J. Rissanen, "Generalized Kraft inequality and arithmetic coding", *IBM J. Res. Development*, Vol. 20, 1976.

[18] F. Rubin, "Arithmetic stream coding using fixed precision registers", *IEEE Trans. Inform. Theory*, Vol. IT-25, No. 6, November 1979, pp. 672–675.

[19] C.E. Shannon, "A mathematical theory of communication", *Bell Syst. Techn. J.*, Vol. 27, 1948, pp. 379–423 and 623–656.

[20] D. Sheinvald and R. Pasco, "Deterministic prediction in progressive coding", *IEEE Trans. Inform. Theory*, to appear.

[21] I.H. Witten, R.M. Neal and J.G. Cleary, "Arithmetic coding for data compression", *Commun. ACM*, Vol. 30, No. 6, June 1987, pp. 520–540.

[22] Y. Yamazaki, F. Ono, T. Yoshida and T. Endoh, "Progressive build-up coding scheme for bi-level images—JBIG algorithm", *J. IIEE of Japan*, Vol. 20, No. 1, February 1991, pp. 41–49 (in Japanese).

[23] T. Yoshida, T. Endoh, Y. Hirabayashi and H. Kato, "Progressive reduction scheme for bi-level images (PRES)", *Proc. 5th Picture Coding Symposium of Japan (PCSJ 90)*, October 1990, pp. 37–40.