Signal Processing: *Image Communication* 6 (1994) 69–76

SIGNAL PROCESSING: IMAGE COMMUNICATION

A fast and efficient compression method for binary images

Pasi Fränti

Department of Computer Science, University of Turku, SF-20520 Turku, Finland

Received 5 April 1993

Abstract

The use of arithmetic coding for binary image compression achieves a high compression ratio while the running time remains rather slow. A composite modelling method presented in this paper reduces the size of the data to be coded by arithmetic coding. The method is to code the uniform areas with less computation and apply arithmetic coding to the areas with more variation.

Key words: Image compression; Binary images; Block coding; Arithmetic coding

1. Introduction

Binary images are a favourable source for *statistical* data compression because of the small size of the source alphabet. The best results have been achieved by *local context models* together with *arithmetic coding* [4]. However, the use of arithmetic coding is restricted by the slowness of the method compared to the conventional CCITT Group 3 and 4 compression algorithms [1,3].

The most important criteria for the quality of an image compression algorithm are the *compression ratio* and the *running time.*

$$\textit{Compression ratio} = \frac{\textit{size of the original data}}{\textit{size of the compressed data}}.$$

In this paper a composite modelling method will be given that aspires to meet both of these demands. An A4-sized image with 150 dpi resolution consists of over 2 million individual pixels and by the local context model method each of the pixels is to be separately coded. The idea of the new method is simple: A global modelling method will be applied for reducing the number of data to be coded by arithmetic coding. The uniform, non-informative areas of the image are separated from the active areas. The primary goal here is to decrease the compression and decompression time while losing as little as possible in the compression ratio.

2. The hybrid compression algorithm

The image is processed by 16 rows at a time. The pixels of the first 16 rows are stored into *input buffer*, which is then operated by a two stage compression algorithm including global and local stages. When these have been processed, the next 16 lines of the image are brought to input buffer while two bottom rows of the previous bunch of lines are kept in memory for reference. The last pixel rows of the image consist of less than 16 rows in case the image size is not a multiple of 16. At that time the deficient rows of the buffer are filled by white pixels.

The global modelling stage is the *Block Coding* method given by [5]. Here the 16-row buffer is decomposed into 16×16-pixel blocks. Again, if the last block overlaps the border of the image, the overlapping area is filled with white pixels so that all the blocks are of size 16×16. Each block is classified either as an *all-white* or *non-white* block. A block is *all-white* if and only if all of its pixels are white. Otherwise the block is *non-white*. The type of each block is coded and therefore only non-white blocks need further processing.

Local modelling is applied to the non-white blocks. In the statistical context model each pixel is coded by arithmetic coding according to its probability. The probabilities are given by an 8-pixel adaptive context model, shown in Fig. 1. Higher degree contexts are possible; however, they require more memory and are slower to operate with. For details of the local modelling scheme, see [2,7,9].

The two-stage modelling leads into a mixture of global and local data, but that does not cause any problems because the encoder and the decoder can be synchronised. The same arithmetic coding method is used for both local and global data to retain the continuity of the arithmetic code. The global modelling data consists of a one-bit decision of the block type. A 0-bit represents a white block and a 1-bit a non-white block and they are coded according to the probability distribution adapted from the image to be compressed.

We use a QM-coder for arithmetic coding because it is fast, efficient and tailored for a binary source [8]. One of its features is that it updates the context model itself. As the other arithmetic coders require a probability as input, the QM-coder requires an index to the context (together with the symbol to be coded of course), so that no bookkeeping of the counts is needed. The 8-pixel template requires 256 different contexts. One extra context is reserved for the global data so that 257 contexts are used overall.

The global and local modelling stages are separated in the sense that all blocks in the buffer are analysed (and coded) before the local stage takes place, see Fig. 2. The local modelling processes the buffer in row major order keeping track of the pixels belonging to a non-white block and thus needs to be coded.

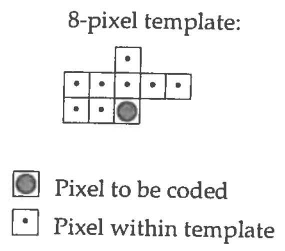


Fig. 1. The template used in the local modelling.

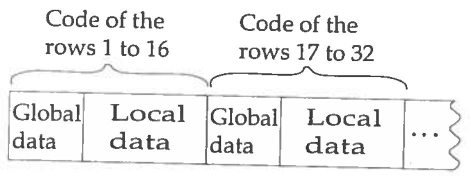


Fig. 2. Output data from the hybrid compression algorithm.

The hybrid compression algorithm works well with images consisting of many white pixels. With the four test images (see Appendix A), the average running time was decreased to 40% of that with pure local modelling, while the loss in the compression ratio was only marginal. A detailed algorithm, its analysis and results of test runs, are found in [2] where a hierarchical variant is considered.

3. Extended block coding

While the compression algorithm of Section 2 works well in most cases, nevertheless some unfavourable images exist for the method. For example an image with an unusually high proportion of black pixels cannot be speeded-up by the basic algorithm. To overcome this problem the blocks are classified into three categories; *all-white*, *all-black* and *mixed*. As a *binary* arithmetic coder is used, the classifying is considered as two different binary classifications, see Fig. 3. Note that in the

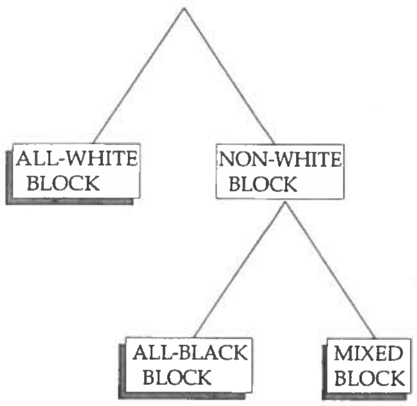


Fig. 3. Decision tree of block classification.

basic method of Section 2, it was sufficient to encode the first choice of the decision tree only.

For typical facsimile images, like the test images 1 to 3 (Figs. 6–8) of Appendix A, the analysis shows that there is not even a single 16×16 all-black block. Therefore no speed-up is expected. In fact, experimental results show that the running time will remain almost the same. On the other hand, the test image 4 (Fig. 9) consists of 322 all-black blocks out of 8800. Even this is too few to have any significant impact on the running time.

At first sight it appears that the extended method would make the compression ratio worse because there are two decisions to code instead of one. This is really the case, but the drawback is insignificant if the block codes are encoded according to their probabilities. In adaptive modelling, the probability for a mixed-block rapidly ends up as 0.99998 (the highest probability in a QM-coder) yielding the entropy of only 0.00002. Thus with about 2500 non-white blocks to be coded, the increase of the extra classification remains below 1 bit. Because of the nature of the dynamic modelling, the probability is lower at the beginning, but still the overall effect remains marginal.

Test image "black hand" (see Appendix A) represents a class of images for which the extended block classification improves. There are 26% all-white, 21% all-black and 53% of mixed blocks in the image. Therefore, the number of blocks to be coded by local modelling is reduced (from 74% to 53%), as compared with the original algorithm. Note that similar artifacts, like the black areas in an image "black hand", are often possible while transforming an image into digital form.

Table 1 gives the summary of the test results, where the BQ refers to the hybrid of block coding and the QM-coder, and BQE refers to the extended BQM-coder. While the BQ takes only 68% of the time taken by the QM-coder, the time of the BQE decreased down to 46%. Compare this to the results of typical white-dominating facsimile images

Table 1
Summary of test results

	Compression ratio			Running time (s)		
	QM	BQM	BQME	QM	BQM	BQME
Image 1	29.9	29.1	29.1	40.7	11.5	12.0
Image 2	16.3	16.0	16.0	42.5	17.3	17.7
Image 3	22.2	21.5	21.5	41.8	15.5	15.9
Image 4	7.5	7.5	7.5	42.7	24.0	23.0
"Black hand"	8.8	8.8	8.7	41.9	28.7	19.7
100% white	139600	139600	139600	41.3	3.1	3.5
100% black	39885	34900	139600	41.3	41.4	3.8

QM = QM-coder.
BQM = BQM-coder.
BQME = Extended BQM-coder.

(circa 40%), like the test images 1–4. All the methods can compress all-white and all-black images into a fraction of the original size, but only BQE-variant manage to do it in minimum time, while the basic BQM-coder fails with the all-black image.

Test runs were performed in a 486/33 PC-compatible machine by using Turbo C 2.0 programming language. It is noted that the compression and decompression times for the algorithms are of the same order of magnitude and therefore only compression times appear.

4. Block size

So far we have applied the global modelling to 16 × 16 blocks only. The effect of the block size on the compression ratio can be summarised as follows:

Large block size

+ High compression ratio.
– Slow compression/decompression
– High consumption of working space,

Small block size

– Low compression ratio.
+ Fast compression/decompression.
+ Low consumption of working space.

Fig. 4 illustrates the dependency of the number of different block types as a function of the block size. Further, the interaction of the compression ratio and running time is shown in Fig. 5. The choice of the block size should be parametrised in the implementation so that one can fine-tune the method according to the demands at the time.

5. Other similar ideas

A similar idea to ours is the *typical prediction* component in JBIG, which skips lines that are equal to the previous one [8]. This means that successive all-white and all-black lines can be coded with less computation. In fact, this can be considered as a special case of our 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 3–4, see Table 2.

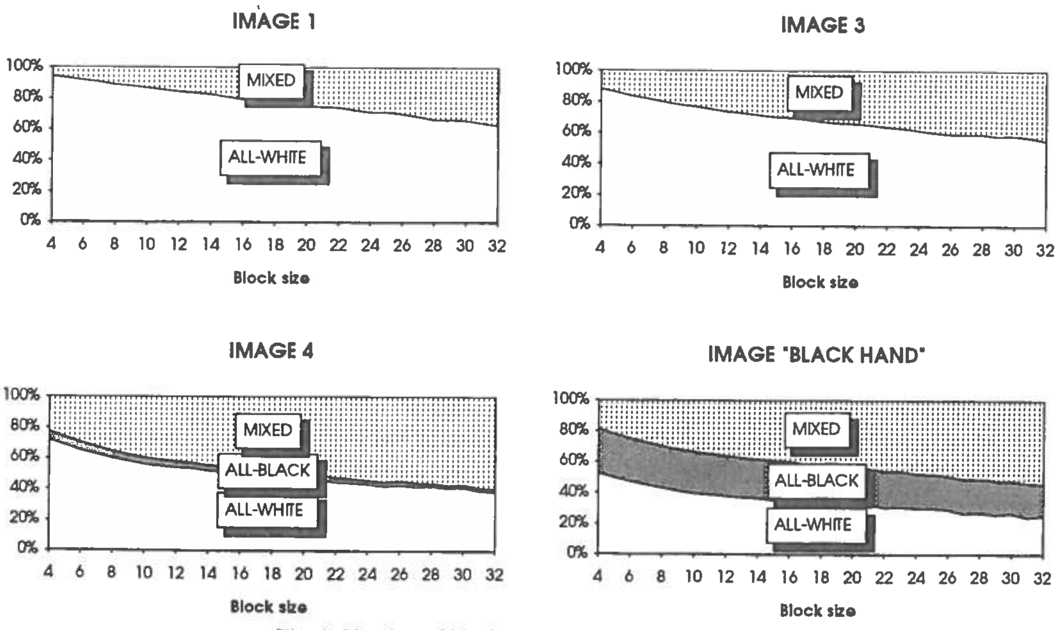


Fig. 4. Number of block types as a function of block size.

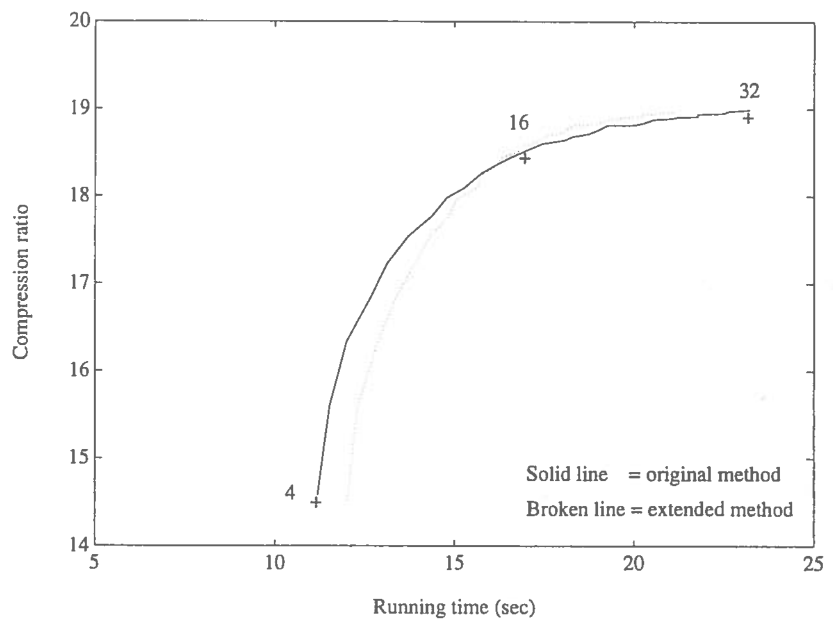


Fig. 5. Test result of different block sizes for the test images 1 to 4.

Table 2
Proportion of pixels skipped by different methods

	Image 1	Image 2	Image 3	Image 4	"Black hand"
All-white lines	35.0%	14.2%	1.8%	0.0%	0.0%
All-black lines	0.0%	0.0%	0.0%	0.0%	0.0%
All-white blocks	79.4%	65.7%	69.7%	50.4%	35.8%
All-black blocks	0.0%	0.0%	0.0%	3.7%	25.2%

The *speed-up mode* of JBIG tackles the same problem with a different way [8], and thus can be seen as an alternative to our method. The coding phase can be made quicker if the context of consecutive pixels remains identical. This is the case inside all-white and all-black areas. The speed-up mode, however, does not allow to skip the coding of a pixel entirely. The context must yet be determined and the coding phase is not totally costless either. For more information of the speed-up mode, see [6].

6. Conclusion

A hybrid compression algorithm for binary images was presented. The method gains advantage from the large uniform areas existing in typical facsimile images by coding them extremely quickly. At the same time a high compression ratio is achieved by the use of arithmetic coding for the non-monotonic areas. The QM-coder is a good choice for the coding algorithm while the hybrid modelling method does not exclude any other arithmetic coder.

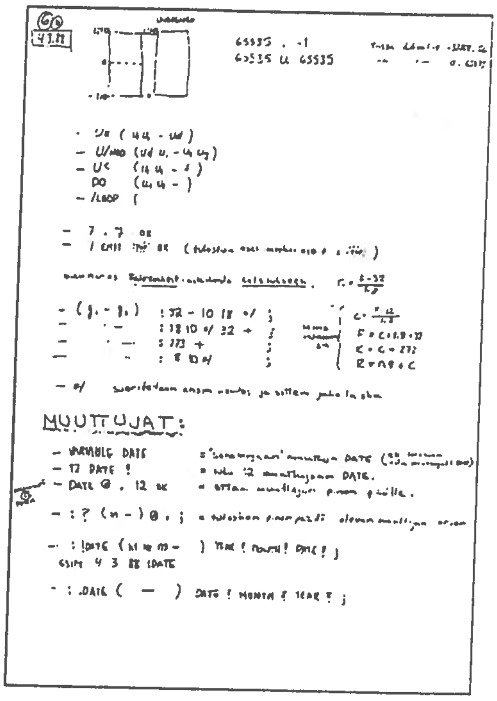


Fig. 6. Image 1.

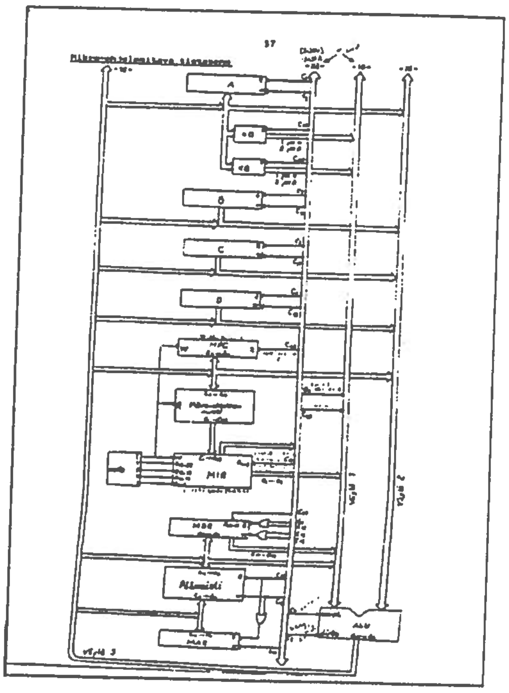


Fig. 8. Image 3.

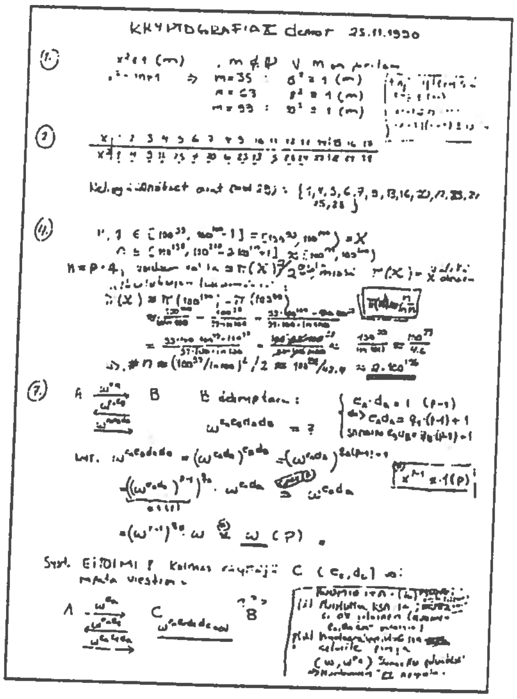


Fig. 7. Image 2.

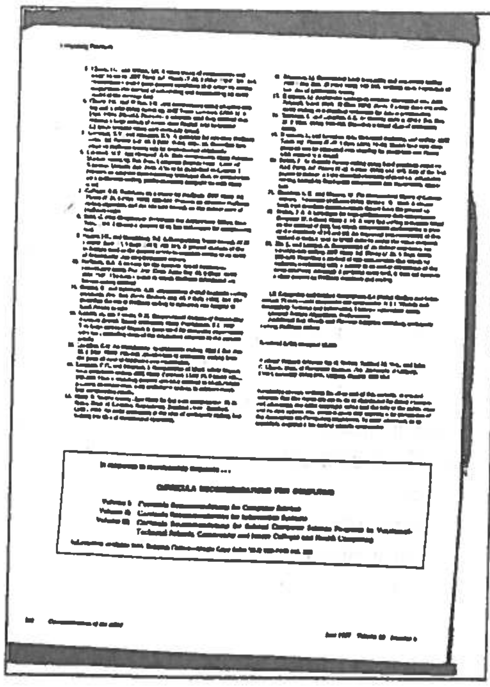
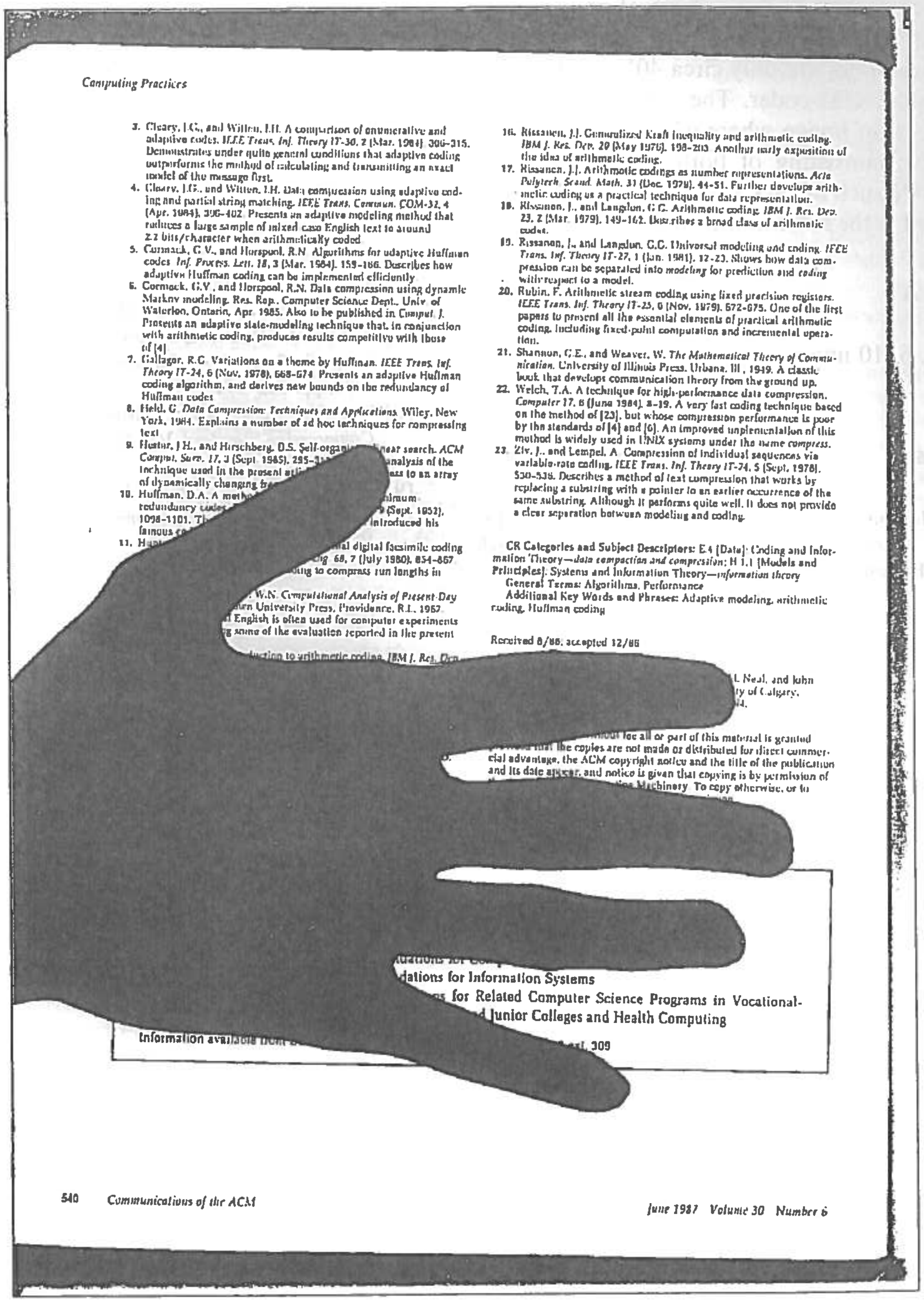


Fig. 9. Image 4.

IMAGE "BLACK HAND"



Computing Practices

3. Cleary, J.G., and Witten, I.H. A comparison of enumerative and adaptive codes. *IEEE Trans. Inf. Theory IT-30*, 2 (Mar. 1984), 306–315. Demonstrates under quite general conditions that adaptive coding outperforms the method of calculating and transmitting an exact model of the message first.
4. Cleary, J.G., and Witten, I.H. Data compression using adaptive coding and partial string matching. *IEEE Trans. Commun. COM-32*, 4 (Apr. 1984), 396–402. Presents an adaptive modeling method that reduces a large sample of mixed case English text to around 2.2 bits/character when arithmetically coded.
5. Cormack, G.V., and Horspool, R.N. Algorithms for adaptive Huffman codes. *Inf. Process. Lett. 18*, 3 (Mar. 1984), 159–166. Describes how adaptive Huffman coding can be implemented efficiently.
6. Cormack, G.V., and Horspool, R.N. Data compression using dynamic Markov modeling. Res. Rep., Computer Science Dept., Univ. of Waterloo, Ontario, Apr. 1985. Also to be published in *Comput. J.* Presents an adaptive state-modeling technique that, in conjunction with arithmetic coding, produces results competitive with those of [4].
7. Gallager, R.G. Variations on a theme by Huffman. *IEEE Trans. Inf. Theory IT-24*, 6 (Nov. 1978), 668–674. Presents an adaptive Huffman coding algorithm, and derives new bounds on the redundancy of Huffman codes.
8. Held, G. *Data Compression: Techniques and Applications*. Wiley, New York, 1984. Explains a number of ad hoc techniques for compressing text.
9. Hester, J.H., and Hirschberg, D.S. Self-organi[illegible]near search. *ACM Comput. Surv. 17*, 3 (Sept. 1985), 295–3[illegible] analysis of the technique used in the present arti[illegible]ess to an array of dynamically changing fre[illegible]
10. Huffman, D.A. A meth[illegible]imum redundancy code[illegible] (Sept. 1952), 1098–1101. Th[illegible] introduced his famous co[illegible]
11. Hu[illegible]al digital facsimile coding [illegible]g. 68, 7 (July 1980), 854–867 [illegible]ding to compress run lengths in

[illegible] W.N. *Computational Analysis of Present-Day* [illegible]wn University Press, Providence, R.I., 1967 [illegible]d English is often used for computer experiments [illegible]g some of the evaluation reported in the present

[illegible]duction to arithmetic coding. *IBM J. Res. Dev.*

16. Rissanen, J.J. Generalized Kraft inequality and arithmetic coding. *IBM J. Res. Dev. 20* (May 1976), 198–203. Another early exposition of the idea of arithmetic coding.
17. Rissanen, J.J. Arithmetic codings as number representations. *Acta Polytech. Scand. Math. 31* (Dec. 1979), 44–51. Further develops arithmetic coding as a practical technique for data representation.
18. Rissanen, J., and Langdon, G.G. Arithmetic coding. *IBM J. Res. Dev. 23*, 2 (Mar. 1979), 149–162. Describes a broad class of arithmetic codes.
19. Rissanen, J., and Langdon, G.G. Universal modeling and coding. *IEEE Trans. Inf. Theory IT-27*, 1 (Jan. 1981), 12–23. Shows how data compression can be separated into *modeling* for prediction and *coding* with respect to a model.
20. Rubin, F. Arithmetic stream coding using fixed precision registers. *IEEE Trans. Inf. Theory IT-25*, 6 (Nov. 1979), 672–675. One of the first papers to present all the essential elements of practical arithmetic coding, including fixed-point computation and incremental operation.
21. Shannon, C.E., and Weaver, W. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Ill., 1949. A classic book that develops communication theory from the ground up.
22. Welch, T.A. A technique for high-performance data compression. *Computer 17*, 6 (June 1984), 8–19. A very fast coding technique based on the method of [23], but whose compression performance is poor by the standards of [4] and [6]. An improved implementation of this method is widely used in UNIX systems under the name *compress*.
23. Ziv, J., and Lempel, A. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory IT-24*, 5 (Sept. 1978), 530–536. Describes a method of text compression that works by replacing a substring with a pointer to an earlier occurrence of the same substring. Although it performs quite well, it does not provide a clear separation between modeling and coding.

CR Categories and Subject Descriptors: E.4 [Data]: Coding and Information Theory—*data compaction and compression*; H.1.1 [Models and Principles]: Systems and Information Theory—*information theory*
General Terms: Algorithms, Performance
Additional Key Words and Phrases: Adaptive modeling, arithmetic coding, Huffman coding

Received 8/86; accepted 12/86

[illegible]. Neal, and John [illegible]ty of Calgary, [illegible]4.

[illegible]ithout fee all or part of this material is granted [illegible] that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of [illegible] Machinery. To copy otherwise, or to

[illegible]dations for Information Systems
[illegible] for Related Computer Science Programs in Vocational-[illegible]d Junior Colleges and Health Computing

Information available from [illegible], 309

540 *Communications of the ACM* *June 1987 Volume 30 Number 6*

Fig. 10. Image "Black hand".

The method works well with typical facsimile images, because both uniformly white and uniformly black areas are benefit in the form of an increased speed. Test results show that the compression/decompression times are only circa 40% of the times with the pure QM-coder. The worst case for the algorithm is an image where each block is of the mixed type, consisting of both white and black pixels. While such images are rare, they can still be compressed in the same time as with the QM-coder.

Appendix A

In Figs. 6–10 images 1–4 and "Black hand" are given.

References

[1] K.R. McConnel, D. Bodson and R. Schaphorst, *FAX: Digital Facsimile Technology and Application*, Artech House, Boston, 1992, pp. 1–338.

[2] P. Fränti and O. Nevalainen, "A two-stage modelling method for compressing binary images by arithmetic coding", *The Comput. J.*, Vol. 36, No. 7, 1993, pp. 615–622.

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

[4] ISO/IEC Committee Draft 11544, Coded Representation of Picture and Audio Information – Progressive Bi-level Image Compression, April 1992.

[5] M. Kunt and O. Johnsen, "Block coding: A tutorial review", *Proc. IEEE*, Vol. 68, No. 7, July 1980, pp. 770–779.

[6] J.L. Mitchell and K.L. Anderson, "Speedup mode", ISO/IEC JTC1/SC2/WG8 JPEG-241, January 1989.

[7] A. Moffat, "Two-level context based compression of binary images", *Proc. Data Compression Conf.*, Snowbird, Utah, 1991, pp. 382–391.

[8] W.B. Pennebaker and J.L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, New York, 1993.

[9] I.H. Witten, R.M. Neal and J.G. Cleary, "Arithmetic coding for data compression", *Comm. ACM*, Vol. 30, No. 6, June 1987, pp. 520–539.