Title
Compression of Binary Facsimile Images by Preprocessing and Color Shrinking

Journal
IEEE Trans. on Communications, 38

Authors
Algazi, Ralph
Kelly, Phillip L.
Estes, Robert R.

Publication Date
1990

Peer reviewed

Compression of Binary Facsimile Images by Preprocessing and Color Shrinking*

V. R. Algazi, P. L. Kelly and R. R. Estes

August 16, 1994

Abstract

The compression of binary facsimile images has now become widespread and has led to the adoption of standard codes implemented in VLSI chips. In this paper, we reexamine the heuristic and formal basis for the techniques used in compressing such data. This leads us to suggest a decomposition into the successive operations of preprocessing, image segmentation or color shrinking, and modeling and coding of color blocks. Such a decomposition allows a reevaluation of the comparative performance of standard codes and leads us to propose a new simple code, the PCSE code, which, using preprocessing and color shrinking prior to encoding, outperforms the READ code by from 7 to 42% for the standard CCITT test images with a very small change in image quality.[1]

1 Introduction

The data compression of images is based on identifying and removing redundancy and irrelevancy [JN84]. Redundancy in facsimile images is present in two forms: 1) most images are predominantly white, and 2) adjacent and nearby pixels are highly correlated. Well known techniques based on removing these redundancies have led to compression ratios of about 10-20 for typed letters [HH75, HR80, Ste82] without any loss in quality.

Perceptual irrelevancy in images is a function of resolution, the quantization from a grey level to a binary image, and image quality. Except for increasing resolution, improving the quality in facsimile images by using perceptual information has received little attention, and removing perceptual irrelevancy has been neglected.

To articulate and motivate our work, we outline and discuss the physical and conceptual operations involved in digitally transmitting or storing an image (Figure 1), which are digitization, preprocessing, modeling, signal representation, source encoding, and channel encoding. In this paper, we shall not consider either digitization or channel encoding.

The digitized images we studied are the set of CCITT (Comite Consultatif International de Telegraphie et Telephonie) images used to standardize the comparison of compression techniques. The sampling densities are 200 pixels/inch and 200 lines/inch. Each image consists of 1 bit per pixel, 1728 pixels per line, and 2376 lines per image for a total of 4,105,728 bits per image.

*This research was supported in part by Pacific Bell, Hewlett Packard, Xerox, Digital Equipment Corporation and the research program MICRO of the University of California.

[1] Compression ratio is defined as the number of bits in the original representation divided by the number of bits in the encoded representation.

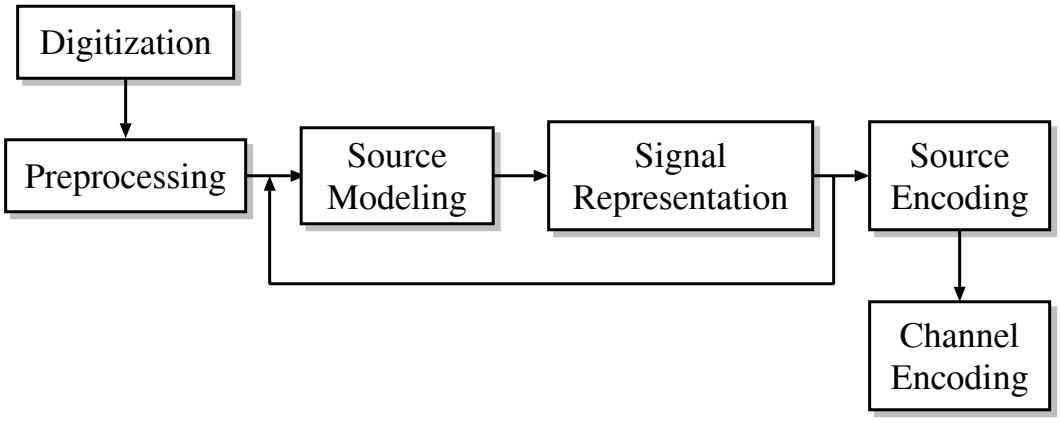

Figure 1: Steps in digital transmission and storage

By preprocessing we mean to alter intentionally the input image to improve quality. Preprocessing can be used to remove distortion introduced when an image is digitized, or to improve an original that is of poor quality. The CCITT binary images we are studying are of poor quality because they contain isolated pixels (that we view as noise), and the edges of lines and characters are noticeably distorted, or jagged.

Source modeling, signal representation, and the source encoder are usually lumped together and called the coding algorithm.

Effective and simple models for binary images identify the edges, or boundaries, of consecutive black and white pixel runs. More complex models identify pixel patterns, either statistically by Markov models, or deterministically by template matching. For binary images, boundary points can be represented by run lengths, contours, or an edge point labeling scheme that we shall propose.

Once representations have been chosen, the encoder will produce the bit stream to be transmitted or stored. For typical binary images, we consider successive modeling and representation steps from the global to the local levels to suggest alternative encoding strategies and possible performance improvement.

2 Outline

We shall propose and examine the performance of new algorithms for preprocessing, global modeling and segmentation, local modeling and representation, and binary encoding.

Preprocessing addresses the problem of perceptual irrelevancy and quality. We examine briefly some binary morphological operations that may be used to improve the quality and increase the compressibility of binary images.

We next consider the modeling of binary images. Models for binary images, on a global level and at the pixel level have been proposed, but little work has been done to integrate the two. The purpose of global modeling is to represent a source as a union of smaller local sources. Some work has been done on segmenting images based on characters [CDW78], but the primary focus has been on white skipping algorithms [HH75]. These techniques try to capitalize on the predominance of white areas in images. We propose a region growing algorithm that identifies global structure. At the local or pixel level, three source modeling techniques are used extensively: Markovian models, run length codes (which are based

on a geometric distribution model of run lengths), and relative address codes (two dimensional extensions of run length codes). We propose a regenerative transformation (our edge point labeling scheme) that is fundamentally related to these techniques and is easily implemented.

Since a complete encoding algorithm may encompass all the steps above, and necessarily the last two, we examine the effectiveness of the preprocessing and global modeling operations on their own, by combining them with "standard" encoders. We shall consider as "standard" codes the modified Huffman code (MHC) in one dimension and the READ code in two dimensions [HR80, RAR78].

Finally we incorporate proposed techniques into a set of comprehensive encoding schemes, denoted the PCSE codes, and demonstrate their effectiveness in compressing the CCITT binary images.

There are eight CCITT documents used as standard facsimile test images. Key features of these CCITT test documents have been extracted and combined in one image to simplify preliminary studies and illustrate results. This composite image, Figure 2, is made up of 256x256 sections from CCITT images 1, 2, 5, and 7. CCITT image 1 is a business letter, image 2 is a hand drawn circuit diagram, image 5 is a portion of a text book, and image 7 is a document written with Kanji characters. In the final evaluation of performance, we encode each separate image to allow comparison with other techniques.

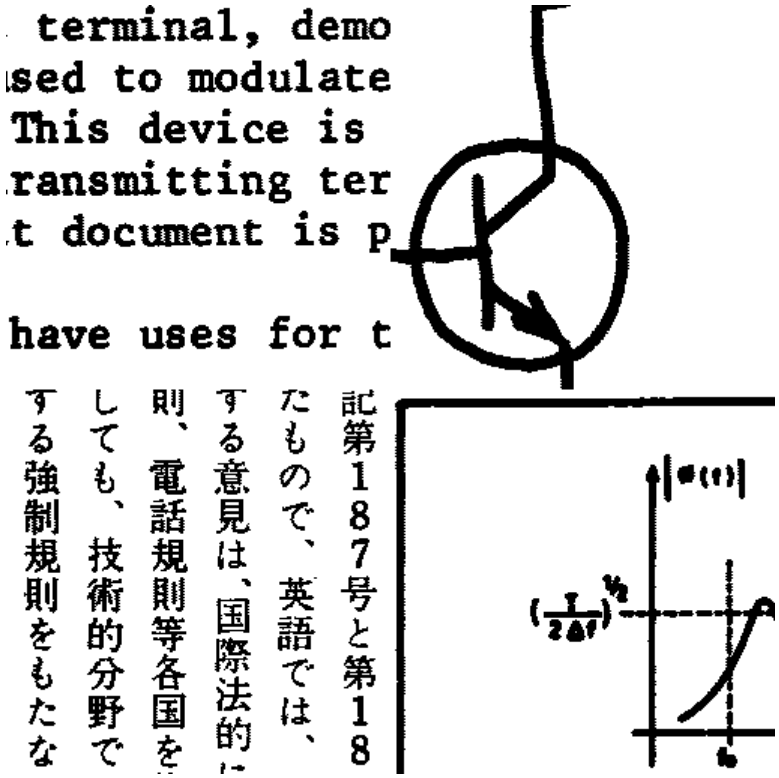

Figure 2: CCITT composite image

3 Preprocessing

Facsimile images contain two unwanted anomalies, isolated pixels and quantization noise. Both of these detract from the subjective quality and introduce unnecessary detail that decreases compression performance. Morphology is a mathematical formalism and tool that we suggest can be used to remove these anomalies without a corresponding perceptual degradation of the image.

The theory of mathematical morphology was introduced by Matheron and Serra [Ser82], and has recently been applied to digital image processing [MS86, Ste82]. We shall apply some simple morphological operations which were designed to remove jaggedness in horizontal and vertical edges. These operations will improve compressibility and often have no effect on the perceived quality of the image.

Morphological operations are "local"; they can be described by a kernel structuring element, or pattern, in a limited neighborhood around each pixel and an If-Then-Else statement. At every pixel in the image, a check is made for the pattern. Based on whether the pattern is found, or not, one pixel value may be changed. Alternative descriptions can be based on convolution using binary arithmetic, or as set operations [MS86, Ste82]. The kernels we used are shown in Figure 4.

The basic Remove kernel is number 3, and the corresponding operation is defined as:

If the pattern is found
Then change the current pixel from black to white
Else do nothing.

It is used to remove a single pixel extending beyond a right edge. To perform the complementary Fill operation, the basic Fill kernel (4) is used. The order of the operations is significant. If the Fill operation is performed first the image tends to grow, while if the Remove operation is processed first the image tends to shrink. These effects are especially apparent for the more detailed Kanji text. After careful examination of the resulting images, we decided that using the Remove kernels first resulted in best overall image quality.[2]

To handle pairs of pixels, the images are first processed by kernels 1 and 2 to remove and fill, respectively, one of the two pixels. Then, kernels 3 and 4 are used to remove and fill the other pixel in the pair as well as all single pixels. In each case, all four 90 degree rotations of the kernels are used.

Figure 3, our composite example, illustrates the quality of typical digitized facsimile images. Figure 3 shows the same text after preprocessing using morphological operations. Comparing Figure 2 and Figure 3, we see that the differences are minor, but the additional structure improves the compressibility without substantially degrading the quality. Improvement in quality also occurs as seen in straight line segments, which appear smoother and more definite, particularly in the graph located in the lower right hand corner of the figures. The reduction in jaggedness or randomness leads to increased compression. The text, both English and Kanji, is also smoothed.[3] This either improves the quality or detracts very little from it.* A numerical evaluation of preprocessing operations on the performance of the READ code ($k = 4$) is summarized in Table 1. Images containing horizontal and vertical line segments show the most improvement. The abundance of detail caused by the pixel variations along the many straight line segments is viewed by the encoding algorithm to be of high information content while to the viewer it is viewed as noise. The circuit schematic (CCITT Document 2) has few straight line segments so that the percent improvement is not as large as with the other images.

[2] In the terminology of mathematical morphology, our "Remove" and "Fill" operations match specific one-zero patterns, and are thus specialized "Hit or Miss" transforms. The classical "Erosion" and "Dilation", as well as the operations of "Opening" and "Closing" derived from them, perform a broader smoothing of shapes and lead to objectionable degradation of the quality of the CCITT documents.

[3] The readability of Kanji was verified by a native reader.

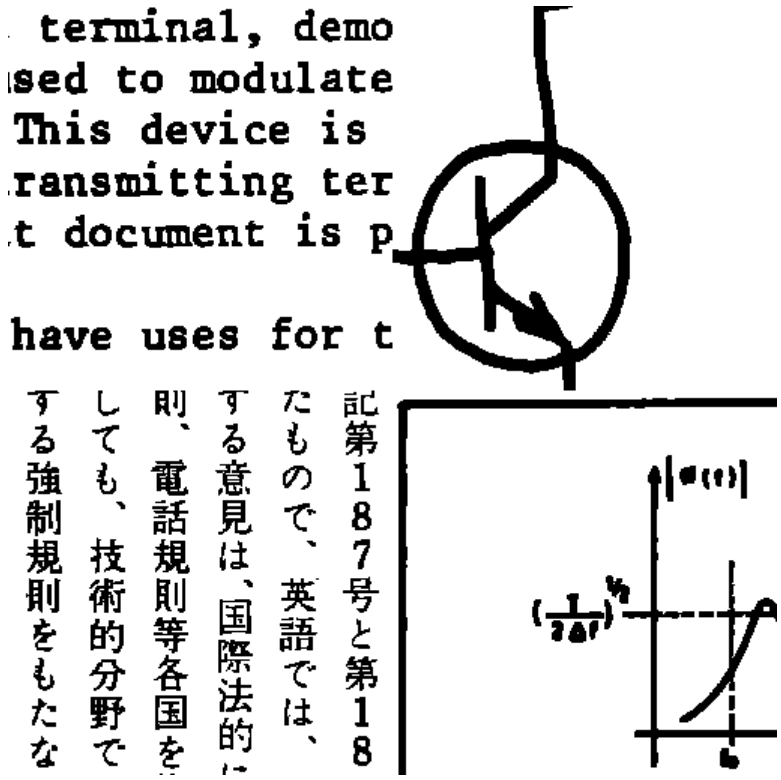

Figure 3: Morphologically preprocessed image

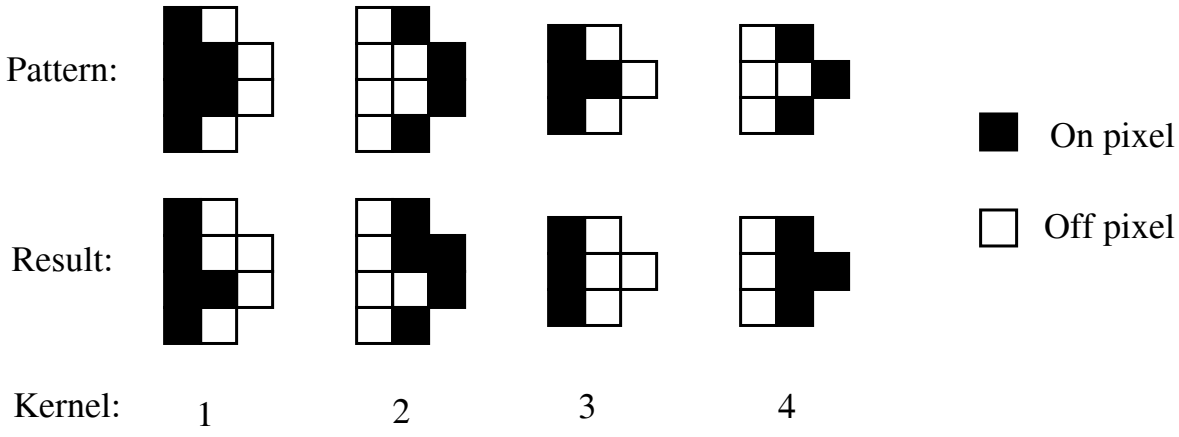

Figure 4: Morphological operations

CCITT document number	Compression ratio READ code ($k = 4$)		Percent improvement
	original	preprocessed	
1	19.77	21.65	9.50
2	26.12	27.54	5.40
3	12.58	14.26	13.29
4	6.27	7.12	13.40
5	11.62	13.06	12.34
6	18.18	20.14	10.79
7	6.30	6.91	9.59

Table 1: Effects of morphological preprocessing on compression ratio

This preliminary examination of the effects of morphological preprocessing on binary images could be complemented by a study of morphological postprocessing which would improve the visual quality before display. Such a study is beyond the scope of this paper which is focused on source encoding. The preprocessed images are used throughout the rest of the paper.

4 Color Shrinking

Most facsimile images are predominantly white which normally suggests that facsimile images be segmented into white regions and regions of high activity. Thus it is not surprising that some of the early facsimile coding algorithms are based on efficiently encoding large white areas. The goal of these white block skipping algorithms [HH75, Hua77, dCK74, KJ80] is to decompose white areas into large one or two dimensional blocks of fixed size, which can be simply described and encoded.

In one dimension, a run length representation is a very effective segmentation technique of white and black regions. In extending the encoding of run lengths to two dimensions, the READ code uses a simple line to line difference of the ends of runs. Thus, the vertical structure of the image is represented and encoded fairly crudely.

Color shrinking is the process of segmenting the binary image into color blocks, which are two dimensional subimages containing both black and white pixels, and white subimages which contain no information. The premise of color shrinking is that image segmentation into a fairly detailed composite source of white areas and color blocks can be performed at a small cost, in bits. Further, color shrinking should result in a relatively small fraction of the image within the color blocks.

4.1 Theoretical Analysis of Color Shrinking

Assume a simple block coding scheme where all the source elements are two dimensional binary blocks of fixed size that we shall denote source symbols, or just symbols. Assume further that these symbols are submultiples of color blocks, i.e., an integer number of code blocks fit within each color block. Let N be the total number of symbols and W be the total number of white symbols in the image. Let $p_1, \ldots, p_n$ be the probabilities of these symbols. Color shrinking will remove W_1 white symbols with $W_1 \leq W$. We shall assume that color shrinking will remove these W_1 symbols at no cost. Thus, we assume that encoding the location of the color blocks can be done with negligible overhead. We now evaluate the

limiting performance of color shrinking by making use of the entropies of the original and color shrunk images.

Encoding of the original image

The expected number of bits required to encode the original image, or source S, is $B = NH(S)$, where

$$H(S) = -\sum_k p_k \log p_k$$

$$\text{with } p_k = \frac{N_k}{N} \text{ and } p_1 = \frac{N_1}{N} = \frac{W}{N} \triangleq w. \tag{1}$$

N_k is the number of occurrences of the k^{th} symbol, and where we have taken p_1 to be the probability of a white symbol.

Encoding of the color shrunk image

Because we remove W_1 white symbols from the image, the probabilities of all resulting source symbols have to be reevaluated for the color shrunk source S'. For colored symbols, $p_k = N_k/N$ becomes

$$p'_k = \frac{N_k}{N - W_1} = \frac{N}{N - W_1}\frac{N_k}{N} = \frac{1}{1 - w_1}p_k = \alpha p_k, \qquad k \neq 1, \tag{2}$$

$$\text{with } w_1 \triangleq \frac{W_1}{N} \text{ and } \alpha \triangleq \frac{1}{1 - w_1}. \tag{3}$$

For white symbols $w = W/N$ becomes

$$w' = p'_1 = \frac{W - W_1}{N - W_1} = \alpha(w - w_1). \tag{4}$$

The number of bits in the colorshrunk image becomes

$$B' = (N - W_1)H(S) \tag{5}$$

with

$$H(S') = -\sum_k p'_k \log p'_k = -\alpha(w - w_1)\log(\alpha(w - w_1)) - \sum_{k \neq 1} \alpha p_k \log(\alpha p_k). \tag{6}$$

After some manipulation, we find that

$$\begin{aligned} H(S') &= \alpha(H(S) + (1 - w_1)\log(1 - w_1) + w \log w - (w - w_1)\log(w - w_1)) \\ &= \alpha(H(S) + F(w, w_1)) \end{aligned} \tag{7}$$

with

$$F(w, w_1) \triangleq (1 - w_1)\log(1 - w_1) + w \log w - (w - w_1)\log(w - w_1) \tag{8}$$

from which we can evaluate the ratio B'/B

$$\frac{B'}{B} = \frac{H(S) + F(w, w_1)}{H(S)}. \tag{9}$$

Limiting Case

In general, we cannot remove all the white symbols by color shrinking and thus $w_1 < w$. But, since

$$\lim_{w_1 \to w} (w - w_1) \log(w - w_1) = 0 \text{ and } \lim_{w_1 \to w} F(w, w_1) \triangleq -H(w) \tag{10}$$

where $H(w)$ now represents the entropy function of the scalar w, we have

$$\lim_{w_1 \to w} \frac{B'}{B} = \frac{H(S) - H(w)}{H(S)} \triangleq \frac{B'_L}{B}. \tag{11}$$

For any actual color shrinking method, its efficiency can be measured by

$$\eta = \frac{B'_L}{B'} = \frac{H(S) - H(w)}{H(S) + F(w, w_1)}. \tag{12}$$

We shall propose a color shrinking scheme which removes a significant fraction of white code blocks at a small increase in the number of bits. Using (12), we shall show that a good efficiency is achieved by such a scheme.

4.2 A Color Shrinking method

The blocking of activity into fairly simple regions, or color blocks, was performed in a number of ways, with fairly comparable results. The method described here is based on the following premises:

1. We wish to block typewritten text, such as CCITT document 1, into isolated words. For a 12 pitch font at 200 dpi, this led us to connect all objects within 8 pixels of each other.
2. Because of the variety of documents of interest, from English and Kanji text, to handwritten and hand drawn diagrams, we wish to devise a scheme which performs effectively on both horizontal and vertical structures.
3. For simplicity in processing, we reset the squaring operation, described later, on 64 bit boundaries, corresponding to 1/3" x 1/3" areas of the document.

The blocking process is divided into three distinct operations: local connectivity, squaring by projections, and removing vertices, as illustrated in Figure 5 and Figure 6. Figure 7 shows the final color blocks generated by this scheme for the composite image of Figure 2.

1. *Local connectivity:* In this stage, we connect all black pixels within 8 white pixels of each other, either horizontally or vertically, by eliminating all horizontal and vertical runs of white pixels ¡ 8 in length. This process generates "color strips."
2. *Squaring by projections:* To carry the blocking process further, we compute the horizontal and vertical projections of color strips within each 64x64 pixel block of the image. The intersections of these projections generate all possible color bounding rectangles as illustrated in Figure 5a. This process can be viewed as a means of finding the "natural" horizontal and vertical boundaries between objects. All color will be included within these color rectangles, but some of them are empty and can be eliminated while others are not full and can be reduced to tight bounding rectangles as illustrated in Figure 5b.
3. *Removing vertices:* At this point, color blocking is quite effective, but, due to the forced resets on 64 bit boundaries, it is not as efficient as it could be. The final process consists of searching the image for a few key corner configurations (Figure 6) to determine if further squaring is justified.

A color block is simplified if the number of bits required to code the incremental area defined by 3 or 4 vertices is less than the number of bits required to code the vertices. As an example, consider case 2 in Figure 6. If the shaded area is included in the color block, we will save 2 vertices, but the shaded area will now require coding. For an effective compression ratio of 4 and a coding overhead of 15 bits/vertex, we simplify the color block if the number of pixels in the shaded area is less than 2x15x4 = 120, or approximately 60 pixels per removed vertex (we actually used 64). This final stage of processing leads to a slight improvement in compression ratio and reduces the number of vertices by more than 30%, on average.

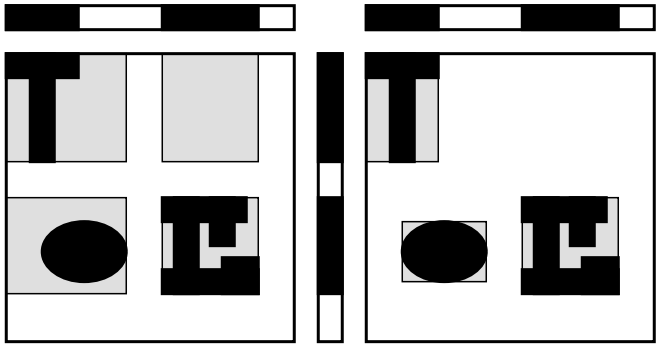

Figure 5: Squaring by projections (a) Step 1: Color bounding rectangles (b) Step 2: Tight bounding rectangles

4.3 A Preliminary Evaluation of Color Shrinking

We now make use of the theoretical analysis of color shrinking presented in Section 4.1 to evaluate the performance of the color shrinking method described in Section 4.2.

Theoretical Evaluation

We measure the effectiveness of color shrinking in removing white symbols by considering 3×3 blocks as the information source. For these results we assume that the code blocks are aligned on a fixed 3×3 grid. The quantities of interest are:

w, the fraction of white blocks in the original documents,
w_1, the fraction of white blocks removed by color shrinking,
$H(S)$, the entropy of the original documents, and
η, the effciciency of the color shrinking method.

For the theoretical evaluation presented here, we assume that the number of bits required to encode the vertices is negligible.

From the results, tabulated in Table 2, we observe that the color shrinking method presented in Section 4.2 removes from 66% - 90% of the original data, that being the data that lies outside the

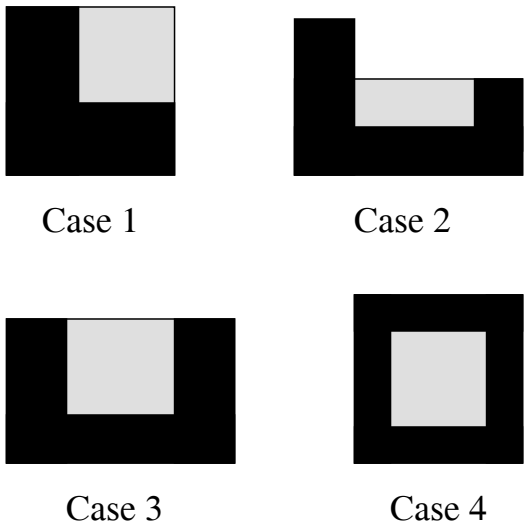

Figure 6: Removing extra vertices

color blocks. The efficiency of the color shrinking scheme ranges from 63% - 77% and is related to the rectangular structure of the documents. Thus, it is highest, and quite consistent, for documents consisting principally of text using either the English or Kanji character sets.

CCITT document number	White blocks in original w (%)	White blocks removed w_1 (%)	Bits/block $H(S)$	Efficiency η (%)
1	94.1	90.3	0.614	75.6
2	94.3	88.8	0.509	63.4
3	88.3	81.8	1.018	74.4
4	79.4	66.4	1.826	77.2
5	88.5	80.8	1.050	74.1
6	92.8	82.5	0.668	63.4
7	83.9	70.3	1.613	76.8

Table 2: Theoretical analysis of color shrinking

A Simple Encoding Scheme

A very simple scheme, encoding the vertices of the color blocks and using the raw binary data within them, yields a simple form of white block skipping. We assume that coding the vertices of the color blocks will require, on the average, 15 bits each. This is because coding each vertex, once the adjacent vertex is known, requires only incremental information, either $(\delta x, y)$ or $(x, \delta y)$, where x and y are zero. The increments δx and δy require at most 12 bits, therefore 15 bits should be sufficient for the pair.

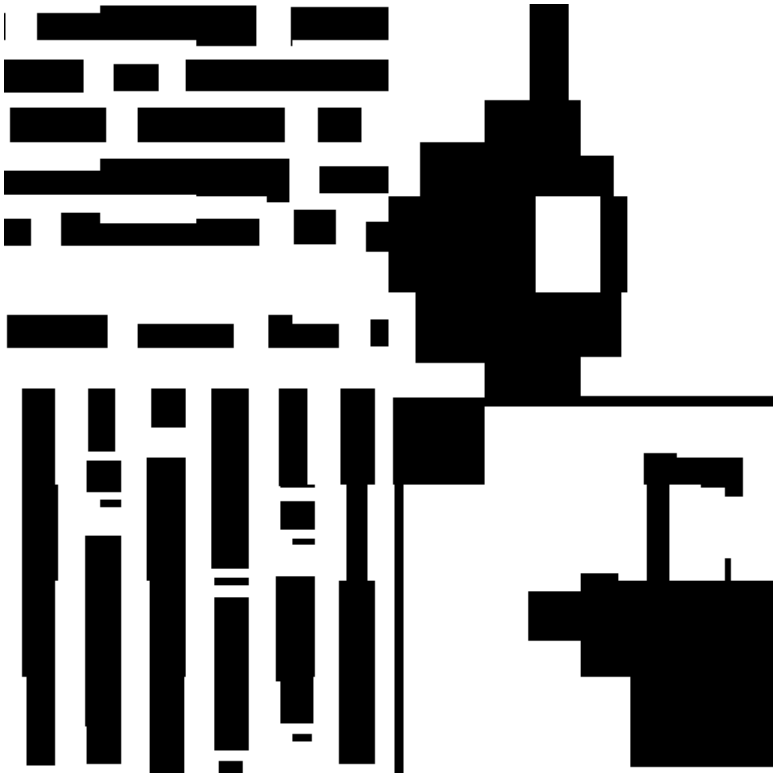

Figure 7: Color shrunk image

Devising a better encoding scheme for the vertices is not critical, since the largest number of bits per pixel to encode the vertices is 0.016 for CCITT document 7. The results using this simple encoding scheme on the morphologically preprocessed images are shown in Table 3.

CCITT document number	Number of vertices	Compression ratio
1	1174	10.94
2	960	9.31
3	1524	5.97
4	4186	3.20
5	2384	5.58
6	1410	6.02
7	4434	3.57

Table 3: Compression ratio due to color shrinking

5 Source Models and Color Blocks

A substantial gain in representation and thus in the coding of the documents has already been achieved using color shrinking alone. We will achieve further gains by encoding within the color blocks. We shall consider several coding schemes which use different source models. Thus, we discuss briefly, in this section, the alternate source models which are pertinent to devising such encoding schemes. In particular, we differentiate the use of run lengths from the use of edge or transition points as a source model.

We view facsimile images as the output of a raster scan device. The simplest source model assumes no dependence between pixels, in which case source extensions can be used to represent the source more efficiently. The approach next in complexity is to use a Markov model, in which case the transition probabilities must be determined. A first order Markov model has been proposed for facsimile images [Cap59, Oli52] and leads to good results. The key feature in line by line models is that the beginning and end of runs of the same color contain all the information.

5.1 Run Lengths Representation and Coding

In run length coding the number of consecutive white pixels and black pixels are transmitted. Capon proposed a first order Markov model for facsimile images and showed, based on this model, that run lengths are independent and that both black and white run lengths are geometrically distributed [KJ80, Cap59]. Run length coding can be considered as the representation of a one dimensional binary source by an alternate source which generates run lengths. One problem is that the run lengths can be very long, i.e. there are too many words in the source alphabet which makes implementation of these codes impractical for typical facsimile images. Of the many variations proposed [JN84], the run length code accepted as a standard by the CCITT is the modified Huffman code. While the maximum possible run length is 1728, which corresponds to one line of a facsimile image, because long run lengths occur with small probability, in the modified Huffman code run lengths longer than 64 are expressed by two code words. The first code word is a prefix which represents a group for the run length. This group is the

integer obtained by dividing the run length by 64. No prefix is used for run lengths less than 65. The second code word represents the remainder.

Since run length codes do not take advantage of the correlation between lines, several extensions of run lengths to two dimensions have been considered [HR80, Hua72, Yas80, MP77]. In all of these schemes a comparison is made between encoding the current pixel with reference pixels, where one reference pixel is on the same scan line and another reference pixel is on the previous scan line. The scheme accepted by the CCITT is the modified READ (relative address designate) code. Because of the two dimensional nature of these codes, transmission errors can be devastating [MP77]. Therefore, associated with the READ code is a factor k which designates that every kth line be transmitted using the one dimensional modified Huffman code. Naturally, the best performance is obtained when k is large. In practice k=4 is used.

We see that, in two dimensional codes, the relative address or run length has now become a new representation of the source. The key to the success of these encoders, is in the efficient representation of the data as runs.

An alternate representation is to label the transition pixels (the end of the runs) and consider separately encoding these new source symbols consisting of the transition pixels.

5.2 One-dimensional Run Length Coding of Color Blocks

Our brief discussion of run length coding suggests that a simple one dimensional coding within color blocks may have merit and may out perform the one dimensional CCITT standard MHC. As we discussed, because of implementation constraints, designing a Huffman code for facsimile images is often not practical, therefore a modified Huffman code is used. Color shrinking limits the maximum length and reduces the average length of the white runs. Furthermore, since the color blocks have more homogeneous statistics than the original image, coding of the white runs may be more efficient. The results of such a scheme are shown in Table 4. Comparing these results with those obtained using the READ code with k = 4 (Table 1), it is worthwhile to note that the performance of color shrinking followed by one dimensional run length coding within the color blocks is close to that of the READ code for the original images, particularly for the business letter (CCITT document 1).

CCITT document number	Compression ratio modified Huffman code		Percent improvement
	original	color shrunk	
1	13.72	16.73	22.0
2	14.94	19.09	27.8
3	7.89	8.89	12.6
4	4.75	4.75	0.1
5	7.51	8.12	8.1
6	10.03	11.93	18.9
7	4.82	4.54	-5.8

Table 4: Compression Ratios using the Modified Huffman Code and Color Shrinking

5.3 Edge Point Labeling

We now consider a scheme for representing edge points which is not limited in scope to two scan lines. The scheme is similar to [SK86]. First, black runs in one direction are determined and the edge points saved. Next, runs of these edge points in another direction are determined and the new edge points saved. Horizontal and vertical line segments are the most appropriate choices here. A problem occurs if a line segment is a single (isolated) pixel since then one edge point needs to mark the beginning and end of a run. This requires that more than two symbols be used to describe an edge point image. Note that we are introducing a new representation that labels the edges so that the process is reversible. (The mapping is one-to-one if there are no isolated pixels after the first step.) An example of this edge point representation for the composite image of Figure 2 is shown in Figure 8. Figure 8 is ternary; isolated pixels in the horizontal direction were discarded, and isolated pixels in the vertical direction were marked using a third symbol, also shown in black in the Figure. This processing is shown in detail in Figure 9.

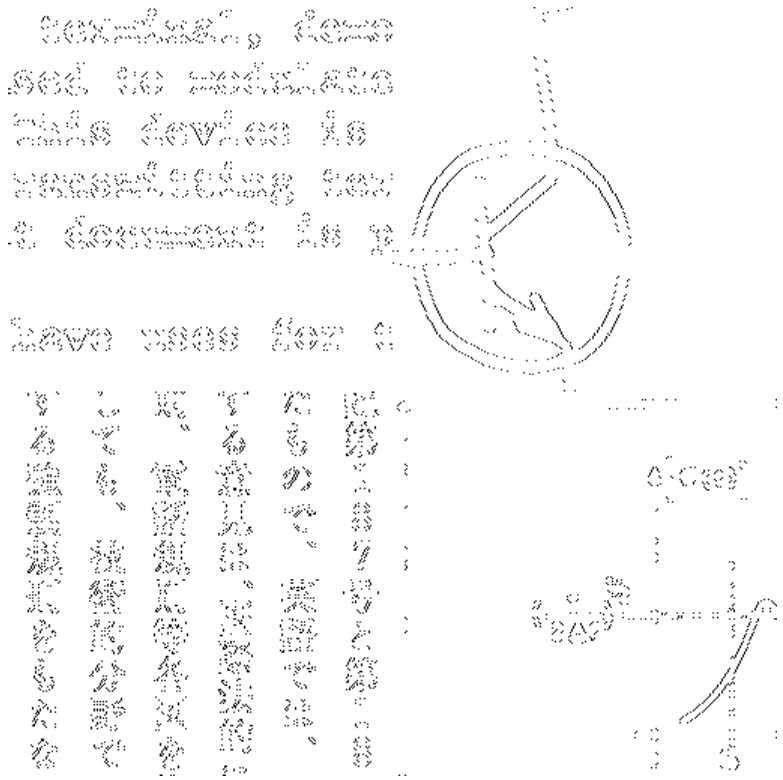

Figure 8: Edge point label of Figure 3.

To extend this idea, the end points of line segments extending in more than two directions or the end points of curves could be marked. As the complexity of this approach increases, the edge point labeling becomes a combination boundary tracing and run length scheme.

Such a scheme now maps the original binary image into a very sparse ternary image. The direct

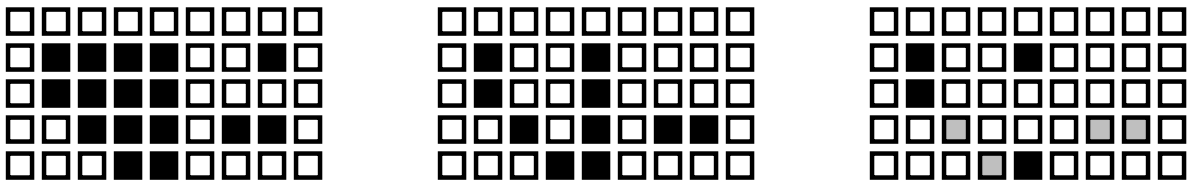

Figure 9: Edge point labeling representation a) Binary input image. b) After processing in horizontal direction; isolated points are discarded. c) After processing in vertical direction; isolated points are denoted by a third symbol.

block coding of such a new source would be rather inefficient, because a small block cannot capture efficiently the large white contiguous areas. But such a scheme, combined with color shrinking leads to an interesting encoding scheme. This is because color shrinking captures quite efficiently the large white areas, and within the color blocks, the non-zero source symbols are relatively frequent.

6 PCSE Codes

The preprocessing, color shrinking, and edge point labeling are now combined to form a new class of data compression algorithms (PCSE codes). In PCSE codes, we label the transition pixels, or ends of runs, and then perform a block coding of the labels. Thus the two dimensional structure of the transition pixels are preserved within each block, but no relative or differential addresses are determined. We consider and compare several labelling schemes:

PCSE1 The one dimensional ends of runs are labelled within the color blocks - generated by color shrinking the morphologically preprocessed images - and then coded using a 4×4 block code.

PCSE2 The two dimensional ends of runs are labelled within the color blocks - generated by color shrinking the morphologically preprocessed images - and then coded using a 3×3 block code.

PCSE3 Same as PCSE2, except that the order of the color shrinking and edge point labelling are reversed, leading to additional color shrinking and more vertices.

Note that we consider only 3×3 block codes for PCSE2 and PCSE3, while using a 4×4 block code for PCSE1. For PCSE1, all black code blocks are no longer possible and many of the symbols are mapped into white symbols. Thus, instead of $2^{16} = 65,536$ source symbols, only 28,561 are possible and only 4740 actually occur in the first seven CCITT documents. For PCSE2 and PCSE3, in addition to the cases above, vertical black runs are not possible. However, we now have a ternary label at each pixel. Still, instead of $3^9 = 19,683$ source symbols, only 1465 are possible and only 961 actually occur (compare with $3^{16} = 43,046,721$, if 4×4 blocks are used.)

In each case, we have evaluated the performance of a Huffman code devised from the statistics of the individual documents. In addition, we designed a generic Huffman code by summing the histograms of the seven documents. The results of our tests are shown in Table 5, where we have used 15 bits to code each vertex, as discussed earlier. For comparison, we show the percent improvement of the generic PCSE1 code over that of the READ code (k=4). The READ code results shown in the table are based on the original image; if the morphologically preprocessed results are used, the percent improvement figures would be reduced by about 10.

CCITT document number	READ code (k=4)	Compression ratio generic Huffman code			% Improvement of PCSE1 over READ (k=4)
		PCSE1	PCSE2	PCSE3	
1	19.77	25.18	23.94	21.38	27.3
2	26.12	37.04	32.29	23.87	41.8
3	12.58	16.54	16.30	15.26	31.4
4	6.27	6.71	6.32	6.17	6.9
5	11.62	14.00	13.53	12.90	20.4
6	18.18	24.24	23.12	23.73	33.4
7	6.30	7.15	6.82	6.51	13.4

Table 5: Compression ratios using PCSE coding

We have observed that the generic Huffman code performs only 1-2% worse than the specific Huffman codes, and that the specific codes are within 1% of the theoretical performance based on the entropy. Thus, a universal coding scheme that applies to all documents seems feasible. It is also interesting to note that the PCSE3 code seldom outperforms the PCSE2 code due to the increased number of vertices created by this strategy. It is only in cases where the documents have very definite vertical structures, such as CCITT document 6, that improvement is possible.

We also observe that the percent improvement of the PCSE2 code with respect to the READ code (k=4) ranges from a low of 7.0% for document 4 to a high of 41.8% for document 2. Thus, very significant improvement in performance is possible using the PCSE codes, especially for sparse documents, since color shrinking is most effective in such cases.

7 Conclusion

In this paper, we have considered several preprocessing and processing steps singly, and in combination, which result in alternative, high performance, encoding schemes for binary documents.

Morphological preprocessing of binary facsimile images produces some improvement in the performance of standard compression codes. Since these codes exploit the horizontal and vertical structure of the images, the preprocessor was designed to smooth irregularities in these directions. We believe that for subsets of images, such as predominantly printed English text, the preprocessor could be more precisely designed to systematically improve quality.

The central theme of color shrinking was examined in some detail and a theoretical measure of its effectiveness was proposed. The specific color shrinking scheme described was predicated on the expected structure of documents scanned at 200 dpi.

Color shrinking has also resulted in a reconsideration of simple coding schemes within the color blocks and documented that good overall performance can be achieved using the composite PCSE codes. In particular, once the image is segmented into homogeneous color blocks, localized codes can be quite effective since their major drawback as a global code is their inefficiency in large white areas, which are now removed by color shrinking.

Because it segments the documents into "natural" regions, color shrinking has other possible uses besides compression, including previewing and editing. For example, it leads naturally to a progressive set of image quality standards using the color blocked image as a preview. We have briefly examined this progressive coding scheme for a range of quality standards which extends to 300 dpi, scanned documents

with grey scale [AEFK90].

Extension of our preprocessing and color shrinking schemes to other scanning resolutions and multi-level graphics coding appears to be straightforward and is currently being considered.

References

[AEFK90] V. R. Algazi, R. R. Estes, G. E. Ford, and P. L. Kelly. Progressive color block coding of bilevel scanned documents. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2221–5, 1990.

[Cap59] J. Capon. A probabilistic model for run-length coding of pictures. *IRE Transactions on Information Theory*, pages 157–63, December 1959.

[CDW78] W. Chen, J. L. Douglas, and R. D. Widergren. Combined symbol matching, a new approach to facsimile data compression. *Proceedings of the SPIE*, 149:2–9, 1978.

[dCK74] F. de Coulon and M. Kunt. An alternative to run-length coding for black-white facsimile. *Proc. International Zurich Conference on Digital Communications*, pages C4.1–4, 1974.

[HH75] T. S. Huang and A. B. S. Hussain. Facsimile coding by skipping white. *IEEE Transactions on Communications*, 23(12):1452–66, December 1975.

[HR80] R. Hunter and A. H. Robinson. International digital facsimile coding standards. *Proceedings of the IEEE*, 68(7):854–67, July 1980.

[Hua72] T. S. Huang. Run-length coding and its extensions. In T. S. Huang and O. J. Tretiak, editors, *Picture Bandwidth Compression*, pages 233–64. Gordon and Breach, 1972.

[Hua77] T. S. Huang. Coding of two–tone images. *IEEE Transactions on Communications*, 25(11):1406–24, November 1977.

[Huf52] D. A. Huffman. A method for the construction of minimum redundancy codes. *Proceeding of the IRE*, 40:1098–1101, September 1952.

[JN84] N. S. Jayant and P. Noll. *Digital Coding of Waveforms*. Prentice–Hall, Inc., 1984.

[KA84] P. Kelly and V. R. Algazi. Two step color shrinking algorithm for the encoding of graphics. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 48.3.1–48.3.4, December 1984.

[KJ80] M. Kunt and O. Johnson. Block coding of graphics: a tutorial review. *Proceedings of the IEEE*, 68(7):770–86, July 1980.

[MP77] H. G. Musmann and D. Preuss. Comparison of redundancy reducing codes for facsimile transmission of documents. *IEEE Transactions on Communications*, 25(11):1425–33, November 1977.

[MS86] P. A. Maragos and R. W. Schafer. Applications of morphological filtering to image analysis and processing. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2067–70, April 1986.

[Oli52] B. M. Oliver. Efficient coding. *The Bell System Technical Journal*, 31:724–50, July 1952.

[RAR78] U. Rothgordt, G. Aaron, and G. Renelt. One-dimensional coding of black and white facsimile pictures. *ACTA Electronica*, 21(1):21–37, 1978.

[Ser82] J. Serra. *Image analysis and mathematical morphology*. Academic Press, 1982.

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.

[SK86] I. Song and S. A. Kassam. A new noiseless coding technique for binary images. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34(4):944–51, August 1986.

[Ste82] S. R. Sternberg. Biomedical image processing. *IEEE Computer*, pages 23–34, January 1982.

[Yas80] Y. Yasuda. Overview of digital facsimile coding techniques in Japan. *Proceedings of the IEEE*, 68(7):830–45, July 1980.