Content-Adaptive Bi-Level (Facsimile) Image Coding

by

Neil H. Tender

S.B., Massachusetts Institute of Technology (1993)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Author 5/16
Department of Electrical Engineering and Computer Science
May 13, 1994

Certified by ....
David H. Staelin
Professor of Electrical Engineering
Thesis Supervisor

Certified by 5/11
Dr. Forrest Tzeng
Comsat Laboratories
Company Supervisor

Accepted by ....
Frederic R. Morgenthaler
Committee on Graduate Students

Content-Adaptive Bi-Level (Facsimile) Image Coding

by

Neil H. Tender

Submitted to the Department of Electrical Engineering and Computer Science
on May 13, 1994, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

The high bandwidth requirements of facsimile communication can make it very costly or even infeasible in environments where these resources are limited. The existing CCITT Recommendation T.4 standard uses lossless Group 3 compression to reduce the number of bits by a factor of 6 to 12, depending upon the contents of the document. However, with the rapidly increasing use of facsimile equipment, a large number of communications services could benefit greatly from an additional reduction in bandwidth.

This thesis describes the development of such an improved coding technique, called Content-Adaptive Facsimile Coding (CAFC). It uses a more sophisticated page model that better represents the types of documents that are typically transmitted via facsimile. Three different coding techniques (Symbol Matching and Substitution, Optimized 2D Run-Length Coding, and non-compressing Direct Coding) are adaptively applied to different parts of the page, followed by a stage of arithmetic coding. CAFC achieves compression ratios that outperform Group 3 by an average of almost 2:1 for most documents and 3:1 for documents consisting predominantly of typed text (25% improvement over JBIG for text). In addition, preliminary estimates show that by using concepts from JBIG to replace the run-length coding, there is the potential for an additional 2:1 improvement for most non-text documents. Although the algorithm is lossy, there is little perceivable distortion introduced into the reconstructed images.

In this research, the target application for CAFC is secondary facsimile compression within Digital Circuit Multiplication Equipment (DCME). The methods developed have the potential to double the capacity of DCME equipment for facsimile transmissions at the expense of a very small amount of image distortion. The amount of additional hardware that would be needed to implement CAFC on DCME facsimile channels is believed to be of the same order of magnitude as that used on existing speech channels.

Thesis Supervisor: David H. Staelin
Title: Professor of Electrical Engineering

Company Supervisor: Dr. Forrest Tzeng
Company: Comsat Laboratories

Acknowledgments

There are so many people to thank, I don't know where to begin! I did my thesis work at both Comsat and MIT and interacted with so many people; it is hard to select only a few to acknowledge. Let me start by thanking the numerous people whose names cannot fit in this small space. You know who you are!

Mom and dad, I cannot begin to count the number of ways which you helped me get through it all. You have truly always been there when I needed you, ready to do anything in your power to make things go well for me. There are no words powerful enough to express my gratitude. And Kim, whenever I start feeling old, you make me feel young again by reminding me our our experiences growing up together.

I know that my beloved Grandma Saire, wherever she may be right now, is watching over me saying "That's my MIT boy!" I could never imagine a grandmother more proud, and there is no question that it is her dignity that has kept me going over the years. She will never be forgotten.

Jon and Jason, we've been together all along, from our freshman "quad" to the completion of our masters theses. It's hard to believe just how many things have changed and even harder to believe how many have stayed the same. We have gone through so much together; thanks for being there when I needed you!

To my many other friends from MIT: Steve, who is constantly reminding me how nice Maryland is; Karl and Dave, who always enjoy a good "jam" and have kept the music flowing; Marcie, Teresa, and Tracy, who keep track of whose birthday we need to celebrate next. You have all been an important part of my college years.

When working on a project intensely, it is important to take breaks. For helping me occasionally forget about my thesis altogether, I have to give credit to my friends who are not from MIT, particularly Arthur, Duane, Paul, Becky, and Jenisa. Thank you for reminding me that work is not the only part of life.

Sometime back in the Spring of 1993 I cracked open a fortune cookie that promised "You will soon get something you have always wanted." Well, that fortune came true. During my final eight month 6A assignment, I learned about something far more important than fax compression – physical fitness. I started running regularly and eating healthy and got in shape for the first time in my life. I would like to thank all of my running partners who helped me make one of the biggest changes in my life: Dad, Frank, Udaya and Rod, Jeff, and more recently, Jon.

I would like to thank all of those who made the whole 6A experience possible. My advisor at MIT, Professor Staelin, and all of my former Comsat advisors, Forrest Tzeng, Rod Ragland, and Udaya Bhaskar, have helped make my assignments incredible learning experiences. Thanks also to the many people at Comsat who have contributed in their own ways. Finally, I would like to thank Professor Papadopoulos, my undergraduate thesis advisor, who always makes everything so fun and exciting, reminding me just why I went into engineering in the first place.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Facsimile (fax) communication has become increasingly popular over the past few years. As the number of fax pages transmitted each year continues to rise at an astonishingly high rate, the technique used for encoding the images becomes extremely important.

The biggest problem inherent to facsimile communication is that it requires the transmission of a tremendous number of data bits. A single bi-level page of a fine resolution CCITT fax consists of close to four million pixels. Without any source coding, this transmission could tie up a 4800 bit/s channel for over 13 minutes. To reduce this burden, a facsimile image compression technique is employed. Most of the more popular facsimile machines and personal computer (PC) plug-in cards support the CCITT Recommendation T.4 Group 3 [1] standard for document transmission. The Group 3 standard employs a modified form of Huffman run-length coding to reduce the transmission time by a factor of 6 to 12, depending upon the contents of the document.

Although the CCITT facsimile protocols were initially designed to be used over the Public Switched Telephone Network (PSTN), the increasing demand for facsimile communications has made it available in a more diverse set of environments. Many of these communications media are very expensive or have severely limited bit-rates, making them economically infeasible or impractical for facsimile even with the existing compression. For example, in 1992, the transmission of a single page over a 4800 bit/sec Inmarsat-M mobile satellite channel would have taken several minutes and cost between $10 and $20 [2].

This thesis describes the development of a secondary facsimile compression algorithm, intended to further reduce the number of bits per page and thus decrease these bandwidth requirements even more. The proposed Content-Adaptive Facsimile Compression (CAFC) technique consists of a more aggressive approach than T.4 Modified Huffman Run-Length Coding, using a sophisticated page model that is better-suited for the types of documents that are typically transmitted via facsimile. Unlike Group 3, which applies a single coding scheme uniformly over the entire page, CAFC makes use of three different techniques, each applied where it would be most effective to minimize the number of bits needed to represent the page. The initial goal is to achieve a compression ratio of 20:1, about a factor of 2 greater than that attained by Group 3, with no degradation in the reconstructed image quality.

Chapters 2-6 explain the fundamental concepts behind CAFC and describe the encoding and decoding techniques in detail. Chapter 7 discusses the procedures developed for optimizing the various adjustable parameters of the algorithm. Chapters 8 and 10 summarize the results of extensive simulations and provide suggestions for future development work. The remainder of this chapter and Chap. 9 discuss the initial target application for CAFC, facsimile compression for Digital Channel Multiplication Equipment.

1.1 Facsimile Compression for DCME

Although CAFC could conceivably be used to improve the efficiency of any facsimile image transmission or storage system, this thesis focuses on its application to Digital Circuit Multiplication Equipment (DCME) [3]. DCME multiplexes hundreds of analog voice, data, and fax channels into a single high-speed digital channel for transmission over a satellite link. To maximize the number of channels that can be operating simultaneously, some form of bandwidth reduction is applied to each channel prior to multiplexing. Voice and data channels are digitized and then passed through suitable coders that achieve compression ratios of 2 or 4. Fax channels are actually demodulated to a digital baseband signal and then passed directly into the multiplexor. This provides substantial gains compared to transmitting the same signal in the voiceband domain, but unlike encoded voice and data, Group 3-encoded facsimile is extremely sensitive to bit errors. In some DCME channels, the bit error rate can approach 10^{-3}, high enough to severely distort any fax page. In order to make DCME viable for facsimile communication, it is necessary to use some degree of Forward Error Correction (FEC). This technique adds redundant bits to the data prior to transmission so that the receiver can detect and correct most of the errors. FEC virtually eliminates distortion to the page introduced by channel errors, but it has the undesirable effect of increasing the number of bits that must be transmitted and thus the required bandwidth.

The proposed solution to this problem is to add a secondary compression stage using CAFC prior to the multiplexor and FEC stages. The extra compression would decrease the required DCME channel bandwidth to a level at or below what it is without FEC and secondary compression, effectively reversing the negative effects of FEC. Figure 1-1 contains a block diagram of the envisioned configuration. The fax pages are scanned and encoded with CCITT T.4 Group 3 coding and then modulated within the facsimile terminal equipment. The DCME equipment then demodulates the fax signals, applies the secondary compression and FEC, multiplexes the channels, and transmits everything over a high-speed digital satellite link. The DCME on the receiving end splits the high-speed input into signals for each of the incoming channels. The facsimile channels are then passed through an FEC error-correcting stage and the resulting CAFC-encoded signals are decoded and converted back to Group 3, removing the secondary compression. Finally, these signals are modulated and routed to the receiving facsimile terminal equipment, ready to be demodulated, decoded, and printed.

The secondary compression consists of two stages, as shown in Fig. 1-2. At the transmitting end, the T.4-encoded document must first be decoded with an inverse Modified Huffman Run-Length coder ($T.4^{-1}$) into a raw bitmap, increasing the total number of bits of approximately a factor of 10. Then, the image is compressed using CAFC, reducing the number of bits by approximately a factor of 20, a net improvement of 2:1. At the receiving end, the process is reversed. The CAFC-encoded image is expanded to a raw bitmap by an inverse CAFC coder ($CAFC^{-1}$), and then the bitmap is encoded using T.4 so that the fax terminal equipment can receive it, an overall increase of 2:1 in the number of bits. In each

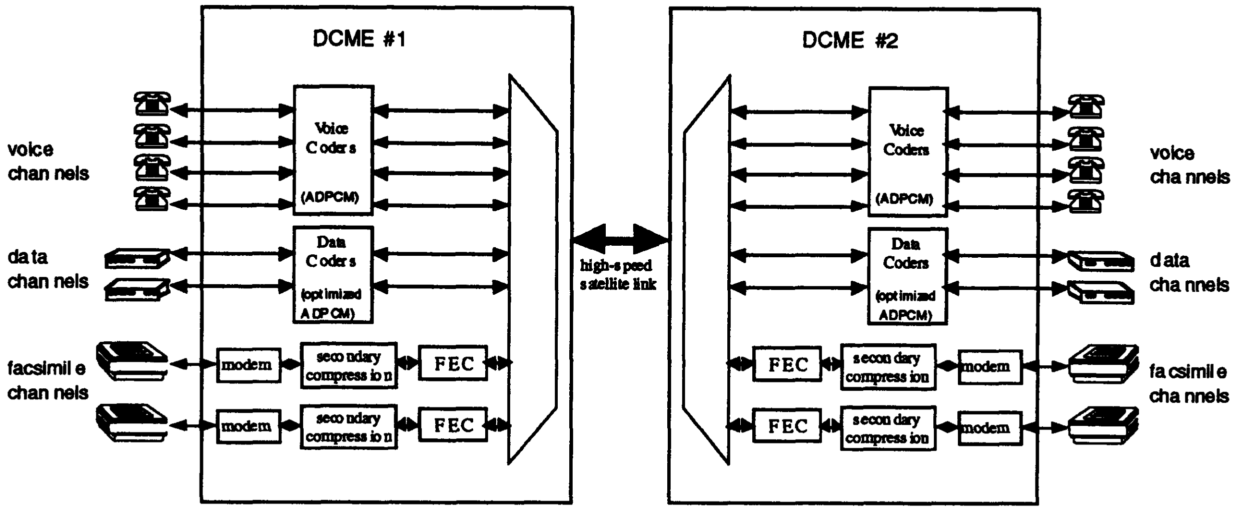


Figure 1-1: Proposed DCME configuration with secondary fax compression.

direction, a sufficient amount of buffering is required to store the intermediary raw bitmap format. However, because this is a real-time implementation, only a small portion of the page need be stored as a raw bitmap at any given time. This is desirable feature of the Group 3 and CAFC algorithms, since it allows the memory requirements and propagation delay to be minimized. Section 9.2 discusses the issue of propagation delay in real-time fax implementations.

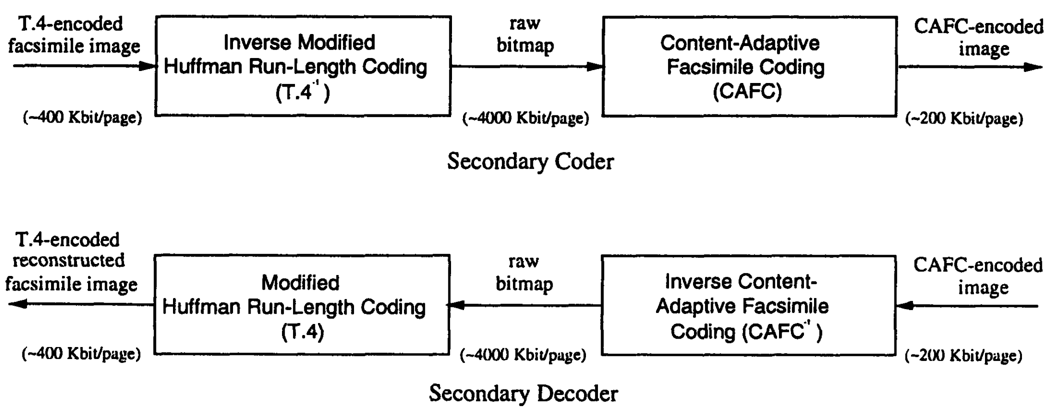


Figure 1-2: Secondary facsimile compression over DCME.

Chapter 2

Overview of CAFC

Bi-level (facsimile) image coding consists of a page model and a coding technique that is applied to the information contained in the model. Facsimile compression involves applying a better model and/or coding technique so as to represent the image in fewer bits, effectively eliminating the redundant information. CCITT Recommendation T.4 Group 3 Modified Huffman Run-Length Coding [1] models the page as a sequence of black and white horizontal run-lengths. These values are coded using a form of Huffman variable-length coding, achieving a compression ratio of approximately 10:1. Group 3 is effective because typical fax documents consist of strings of black or white pixels with unbalanced run-length distributions that can be efficiently entropy-coded with variable-length codewords (see Sect. 4.2).

It may seem that a compression algorithm designed to work well for "typical" documents but not for all documents would be undesirable since there is a loss of generality and lower degree of reliability. For example, it is actually possible for a Group 3-encoded image to contain more bits than the corresponding raw bitmap representation. Unfortunately, what appears to be a design flaw is actually an essential requirement. It is theoretically impossible to achieve compression without the use of a model that makes some assumptions about the contents of the page. In order for a compression algorithm to be effective, it is essential that this page model be a good match for the documents that are to be compressed. In fact, this is true when compressing any type of data, not just bi-level images.

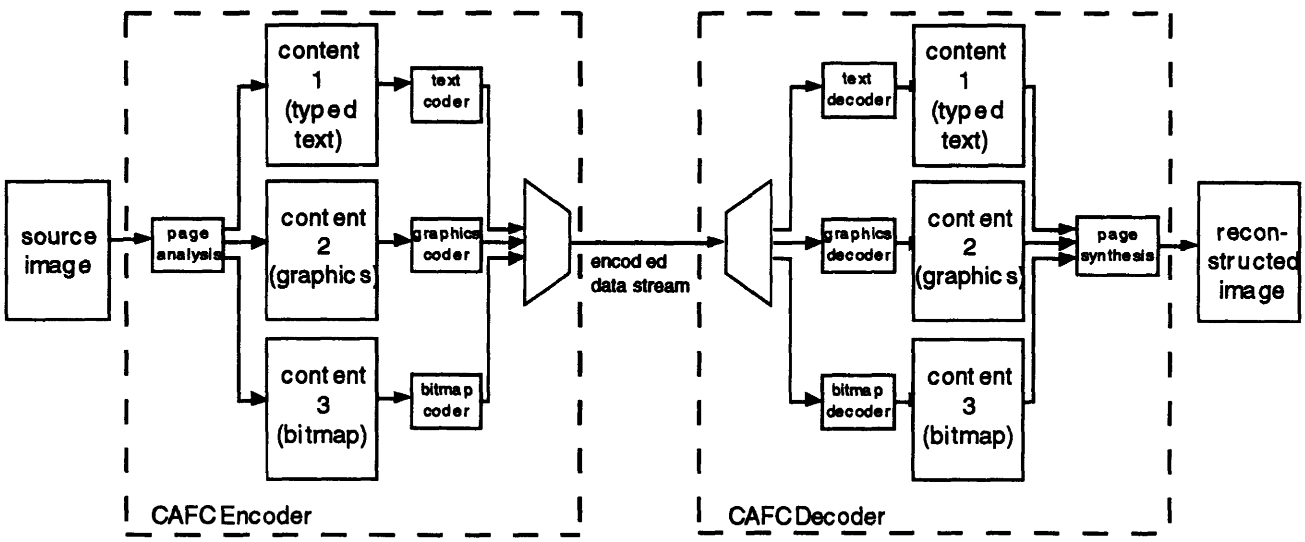


Figure 2-1: CAFC block diagram.

It should be evident that the assumptions that can be made about a source of data can best be determined by the means with which they were created. Many effective speech compression algorithms, for example, use a model of the human vocal tract to decompose speech into a set of filter coefficients and some additional excitation information. While it is not always possible to completely characterize a source, it is often acceptable to make sensible use of some known properties and develop a conservative model that performs extremely well when the assumptions are good yet is not completely ineffective when they are not.

This approach is the basis for Content-Adaptive Facsimile Coding. Facsimile documents typically consist of a large amount of typed text and some simple line graphics (diagrams), handwriting, and possibly some dithered bitmaps (grey-scale images converted to bi-level). CAFC uses a page model that classifies each of these elements as a different type of "foreground content." This representation is useful because each type of foreground content has its own unique properties and can be modeled and coded most efficiently in a distinct manner. CAFC encodes a document by breaking the page down into its different foreground contents, encoding each with a technique optimized for the properties of that content, and then multiplexing all of the encoded data into a single data stream. To decode the image, the data stream is separated back into its different content-specific parts, and then each content is decoded and combined together to form the reconstructed page. Figure 2-1 illustrates the general flow of data during each of these stages.

The important feature of CAFC is that the individual foreground contents are processed independently of one another, allowing the use of completely different coding methods. Each type of content is effectively modeled and coded differently, where the model is adaptively chosen by some local properties of the source image (hence the name Content-Adaptive Facsimile Coding). This differs fundamentally from Group 3 coding, which applies the same model and coding technique to the entire page. It is observed that the various foreground contents that appear in facsimile documents have significantly different properties, and it is therefore expected that an adaptive coder will achieve significantly higher compression.

2.1 Contents and Coding Techniques

For small and detailed image material, Group 3 achieves significantly lower compression ratios than it does on most other documents. Yet, typed text, which fits into this category, is the most prevalent foreground content in typical facsimile documents. For this reason, the primary focus of this thesis is the development of a sophisticated Symbol Matching and Substitution algorithm optimized for compressing typed text. It is expected that this technique alone will provide most of the compression for CAFC. On the other hand, Group 3 compresses larger and coarser image material very well. For contents with these properties, such as handwriting and graphics, CAFC applies a run-length coding technique that is very similar to Group 3, but optimized for the somewhat different run-length probability distributions present in these contents. The CAFC method also takes into account the two-dimensional redundancies on the page and employs an entropy coding technique known as arithmetic coding (AC) to provide additional compression over Group 3. Finally, for dithered bitmaps, Group 3 is an extremely ineffective coding technique; often, the image is actually expanded. To prevent this, CAFC directly encodes the pixels as individual bits without employing any entropy coding.

The CAFC content classifications and associated coding schemes, summarized in Table 2.1, were carefully selected to not only provide as high a compression ratio as possible, but to also comprise a compression system that is practical and reliable. All of the algorithms developed for CAFC can be readily implemented on reasonably inexpensive off-the-shelf hardware platforms. They are designed to produce reconstructed images of high quality,

Foreground Content	Coding Technique
Typed Text	Symbol Matching and Substitution
Graphics	Optimized 2D Run-Length Coding
Handwriting	Optimized 2D Run-Length Coding
Dithered Bitmaps	Direct Coding

Table 2.1: Foreground content types and associated coding techniques.

probably indistinguishable from the originals. The use of a run-length coding variant as one of the coding schemes guarantees that the compression ratio achieved by CAFC for any particular document will not be any lower than it would be with Group 3 coding. Likewise, the use of Direct Coding where appropriate guarantees that CAFC will never produce a coded image that is larger than the source. It is possible that a different set of foreground contents and coding techniques could produce a higher compression ratio and/or better image quality. Those listed in Table 2.1 were selected based upon the observed performance of existing facsimile compression techniques on a variety of documents.

Finally, it is necessary to mention that none of the components of Content-Adaptive Facsimile Coding are completely original. A number of papers have been written about coding schemes similar to Symbol Matching and Substitution [4] [5] [6]. An even larger number of algorithms have been conceived for the efficient lossless two-dimensional coding of facsimile [7] [8] [9] and for the encoding of dithered grey-scale images [10]. In fact, several standards have been introduced since Group 3, including two-dimensional Group 3, Group 4, and JBIG [11]. The content-based nature of CAFC is a relatively new idea that was taken from some preliminary research performed at Comsat Laboratories [12]. However, that report describes a hypothetical compression technique at a high level and does not go into sufficient detail to completely define an algorithm.

The work described in this thesis is an attempt to intelligently combine a number of these existing ideas into a practical working system that can improve the efficiency of facsimile transmissions. Each component is individually redesigned and optimized for use in an image compression system that could be incorporated into a real-time DCME platform. CAFC is the unique combination of Symbol Matching and Substitution, Optimized 2D Run-Length Coding, Direct Coding, and Arithmetic Coding in a content-based facsimile coding scheme.

Chapter 3

Page Modeling and Analysis

As described in the previous chapter, a facsimile page is modeled as a combination of several different types of foreground content – typed text, graphics, handwriting, and dithered bitmaps. To keep things simple, graphics and handwriting are grouped together as a single content because they have similar properties and share the same coding technique, Optimized 2D Run-Length Coding. To encode a document, the CAFC encoder must scan the page (a raw bitmap) and divide it into a large number of small regions, each containing an occurrence of a single content. The CAFC page model specifies exactly what constitutes a region of a particular content so that the encoder's analysis algorithm can efficiently and systematically break down the page into its components.

The content classification is performed progressively, as shown in Fig. 3-1. The encoder first tries to detect instances of typed text, the content with the most efficient associated coding technique. The basic element of typed text is the *symbol*, defined as a cluster of black pixels that is completely surrounded by white pixels. A symbol is essentially a single typed character: a letter, a digit, a punctuation mark, or part of a character in cases where the character consists of two or more segments, such as the percent sign (%). The encoder uses a unique symbol detection/isolation algorithm to locate all such clusters and subsequently codes them using the Symbol Matching and Substitution technique.

Once all of the symbols have been detected and isolated, the remainder of the page should

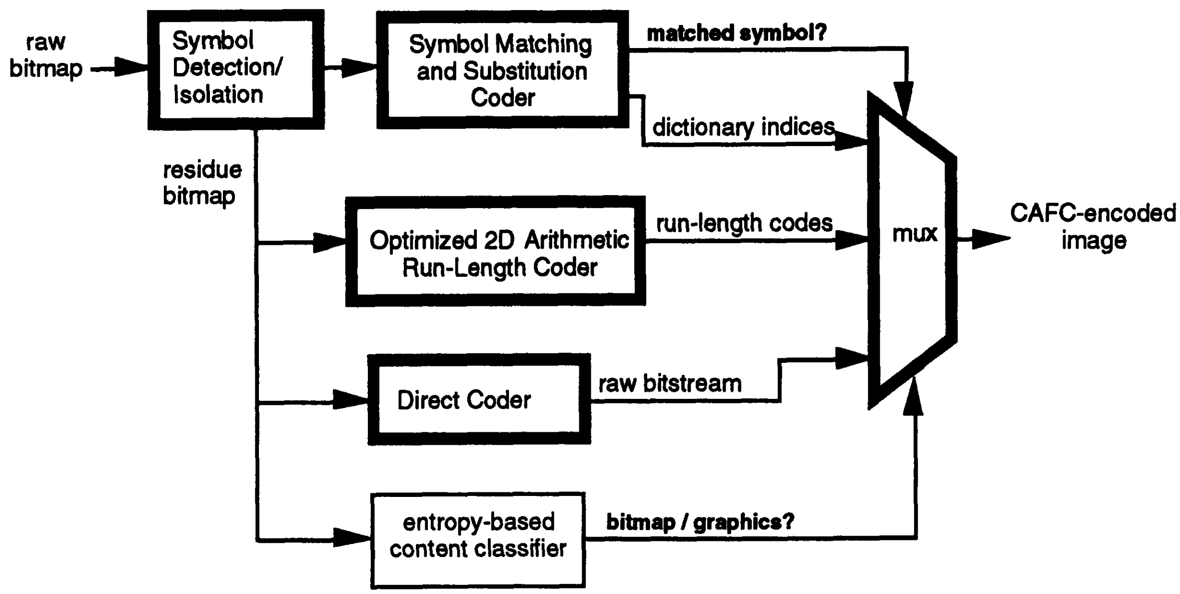


Figure 3-1: CAFC encoder block diagram.

be free of typewritten text. An entropy-based content classifier examines each scan line of this image and searches for segments that would have unusually high entropies (information contents) when represented with the two-dimensional run-length model. These scan line segments are classified as dithered bitmap fragments because their run-length distributions indicate that they contain a large number of very short run-lengths, a property of dithered bitmaps. For now, Direct Coding simply inserts these pixels into the encoded bit-stream without any attempts at compression. All other segments are classified as the handwriting/graphics content and are coded with the entropy-based Optimized 2D Run-Length Coder.

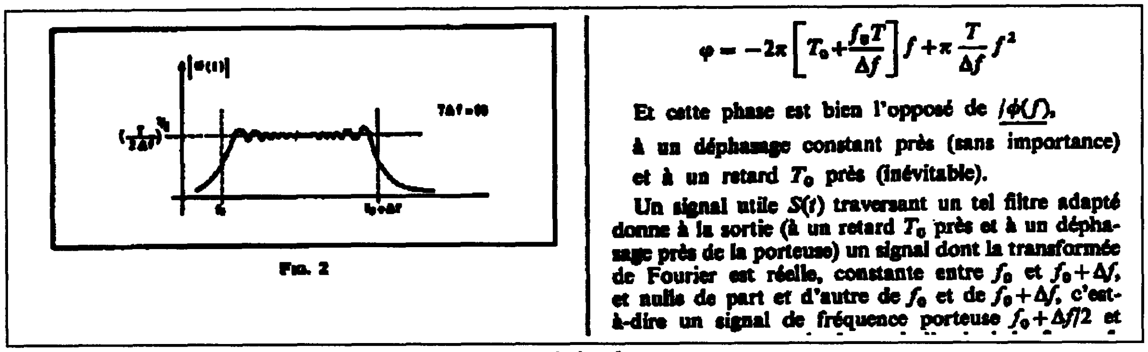
It is important to realize that these specifications were chosen to be practical and that the CAFC encoder may not always be successful at correctly recognizing and categorizing each instance of a foreground content. For example, a detected symbol will not always turn out to be a typewritten character; a small handwritten number or part of a bitmap could easily qualify. Likewise, portions of the page with dense text or graphics might have high enough run-length entropies that they are classified as bitmaps. However, it is not particularly important that the CAFC criterion for typed text, handwriting, graphics, and bitmaps exactly match the perceptual significance of these contents. The primary objective for CAFC is efficient and reliable compression, not character recognition or accurate content-based modeling. As long as the page model exploits the common properties of the majority of facsimile images, it is accomplishing this objective. Designing a page model that is as close to the perceptual level as possible is beneficial because it indirectly takes advantage of these particular properties. In CAFC, this approach works especially well since the characteristics that are sought out during the content classification are the same ones that are used to perform the actual coding. For example, the symbol-based model that is used for typed text allows for very efficient coding via Symbol Matching and Substitution. Similarly, the criterion that is used for locating dithered bitmap fragments also guarantees that a classification leading to the optimal coding method is made.

The progressive nature of CAFC is a very important aspect of this approach. Of the three coding techniques used in CAFC, the Symbol Matching and Substitution coding technique provides the greatest compression gains. For this reason, the encoder first attempts to detect symbols in hopes that typewritten text will be discovered, so that Symbol Matching and Substitution can be applied. Then, after the symbols have been removed, entropy-based content classifier scans the page for instances where Direct Coding would be most efficient, most likely dithered bitmaps. Finally, Optimized 2D Run-Length Coding, the "default" technique, is applied to the remainder of the page, which is assumed to consist of handwriting and graphics. Thus, by performing the content detection algorithms in this specific order, the maximal compression can be achieved.

3.1 Symbol Detection and Isolation

The symbol isolator has the specific task of extracting all symbols from an image, making them available for coding with Symbol Matching and Substitution. The objective is to detect all isolated instances of contiguous black pixels (clusters of black pixels surrounded entirely by white pixels) that meet some predetermined minimum and maximum size constraints. Figure 3-2 contains an example of a portion of an an image where all of the symbols have been isolated and removed. In this case, the minimum allowed symbol size (*widthxlength*) was 2×3 pixels and the maximum allowed symbol size was 40×60 pixels.

The restriction on the maximum size of detected symbols is an important requirement of



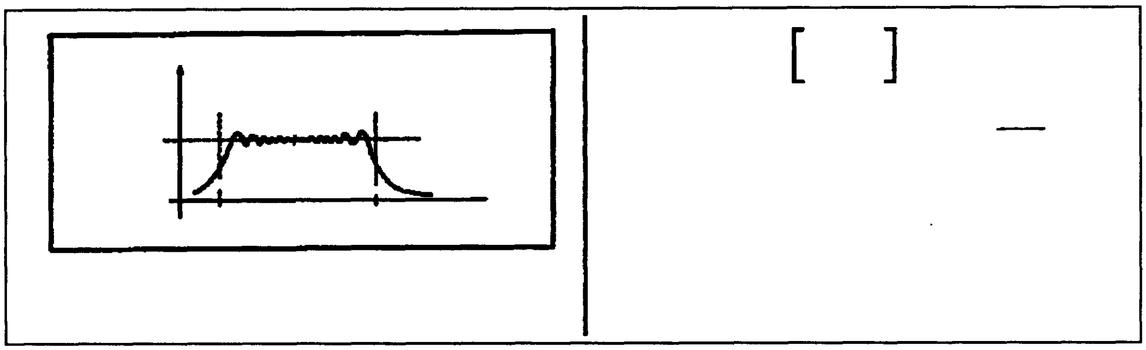
$$\varphi = -2\pi\left[T_0+\frac{f_0T}{\Delta f}\right]f+\pi\frac{T}{\Delta f}f^2$$

Et cette phase est bien l'opposé de $/\phi(f)$, à un déphasage constant près (sans importance) et à un retard T_0 près (inévitable).

Un signal utile $S(t)$ traversant un tel filtre adapté donne à la sortie (à un retard T_0 près et à un déphasage près de la porteuse) un signal dont la transformée de Fourier est réelle, constante entre f_0 et $f_0+\Delta f$, et nulle de part et d'autre de f_0 et de $f_0+\Delta f$, c'est-à-dire un signal de fréquence porteuse $f_0+\Delta f/2$ et

FIG. 2

original



typed text removed

Figure 3-2: Example of symbol isolation.

the symbol isolation algorithm. It is required for a number of reasons, but primarily to minimize the amount of propagation delay in CAFC so that it may be incorporated into a real-time facsimile transmission system (see Sect. 9.2). If, while investigating a possible symbol, the maximum allowable horizontal or vertical dimensions are exceeded, the isolation is abandoned and the next potential symbol is pursued. Each of the three different isolation techniques has a unique method for detecting this condition. A minimum symbol size is also imposed, since Symbol Matching and Substitution would not efficiently code very small symbols. The maximum and minimum symbol sizes are fixed and determined in advance.

Whenever the symbol isolator successfully locates a symbol, it is immediately removed from the page (so that it will not be detected again) and is then passed on to the coder. When the symbol isolation procedure is complete, what remains on the page consists of white space and clusters that are too small or too large to be symbols.

When encoding an image, the CAFC encoder systematically scans the image from left to right and from top to bottom until it encounters a black pixel. The coordinates of this pixel are used as the starting point for the detection and isolation of a symbol. Because it is a fairly involved process, three different methods for performing symbol isolation have been developed: symbol filling, symbol tracing, and symbol windowing. The approaches differ in terms of their computational burden, memory requirements, and overall ability to detect all of the symbols in an image. Section 7.1 describes the procedure for evaluating the performance of each technique and selecting the best one for CAFC. The following sections explain simplified versions of each of the three approaches. The actual real-time algorithms that were developed for CAFC are omitted from this discussion because they are considerably more tedious.

3.1.1 Symbol Filling

With *symbol filling*, the isolator examines each of the starting pixel's eight adjacent neighbors and selects only those which are black. Then, each of the selected pixels are checked for adjacent black pixels in the same manner. This procedure is applied recursively until the entire cluster of contiguous black pixels has been isolated. When a black pixel is detected, it is also marked so that the isolator will not detect it again and get stuck in an infinite loop. If the rectangular region spanned by the cluster of marked pixels is within the permitted range of sizes, a symbol has been detected. Otherwise, the pixels are left behind for subsequent coding by one of the other coding techniques.

Figure 3-3 contains an example of symbol filling. The numbers on each arrow indicate the order in which the black pixels are detected and marked. In this case, the isolator examines the neighboring pixels in the clockwise direction, starting with the pixel immediately to the right. Thus, if the first black pixel is at coordinate (x,y), the isolator examines the eight neighboring pixels in the following order: (x+1, y), (x+1, y-1), (x, y-1), (x-1, y-1), (x-1, y), (x-1, y+1), (x, y+1), and (x+1, y+1).

Symbol filling is capable of detecting all of the symbols on the page correctly. However, it has the disadvantage that it must inspect every black pixel in the symbol, requiring a lot of temporary storage and processor cycles.

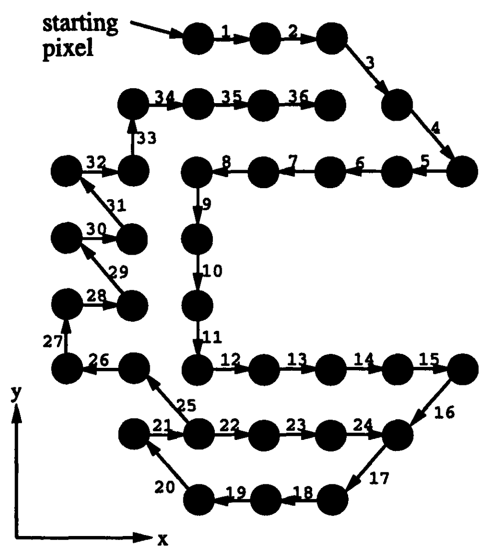


Figure 3-3: Symbol filling.

3.1.2 Symbol Tracing

The particular order in which the source page is scanned for symbols guarantees that the initial black pixel will always be on the boundary of a potential symbol. In *symbol tracing*, this pixel is used as the starting point for a contour trace. The isolator examines the pixel's neighbors in a specific order to determine which is the next boundary pixel, moving in a clockwise direction. The trace continues until the first pixel is encountered once again. If the size of the traced region is within the permitted range, a symbol is detected, consisting of all of the black pixels within the boundary. Otherwise, these pixels are marked as non-symbols.

Figure 3-4 illustrates the procedure for symbol tracing. Once again, the numbers on each arrow indicate the order in which the contour is traced. At each stage of the trace, the search for the next boundary pixel begins at the pixel one step clockwise from the previous pixel and proceeds in the clockwise direction. If the current pixel is at coordinates (x, y) and the search for the next boundary pixel goes past (x+1, y), pixel (x+1, y) is marked. Likewise, if the search goes past (x-1, y), pixel (x, y) is marked. In the figure, the marked pixels are designated with an X. After the trace is complete, the detected symbol consists of all black pixels contained within the horizontal segments formed by the marked pixels.

This technique requires less processing power and memory than symbol filling, but it does not always produce exactly the same results as symbol filling. The isolated symbol consists

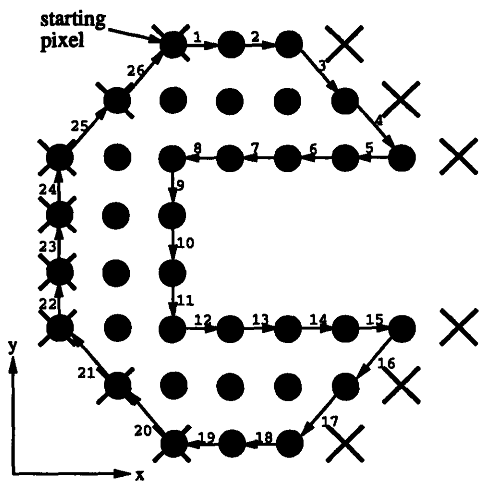


Figure 3-4: Symbol tracing.

of all black pixels enclosed by the boundary, and it is possible to have scenarios where the pixels are not all contiguous. This will occur when a smaller cluster, completely surrounded by white pixels, is enclosed by a contour of black pixels, such as in the character ©. Effectively, a smaller symbol is contained within the larger one. Despite this inconsistency with the strict definition of a symbol, the performance of the symbol matching and substitution should not necessarily be any worse than it would with a filling isolator.

3.1.3 Symbol Windowing

Finally, with *symbol windowing*, a rectangular window centered around the first detected black pixel is used to surround the symbol. Initially the size of a single pixel, the rectangle is gradually expanded in the horizontal and vertical directions until all of the pixels on its four edges are white, or until the window is larger than the maximum allowed symbol size. At this point, the isolation is complete; if the window is within the permitted size constrints, a detected symbol is enclosed. Figure 3-5 illustrates an example of this process.

This approach is conceptually the most simple and straightforward and has very small processing and storage demands. It is similar to symbol tracing in that it can sometimes isolate "symbols within symbols", but this is not a problem from the coding efficiency perspective. The difficulty with symbol windowing is that a symbol can only be detected if it can fit inside of a white rectangular outline. For most typed text this should pose no problem

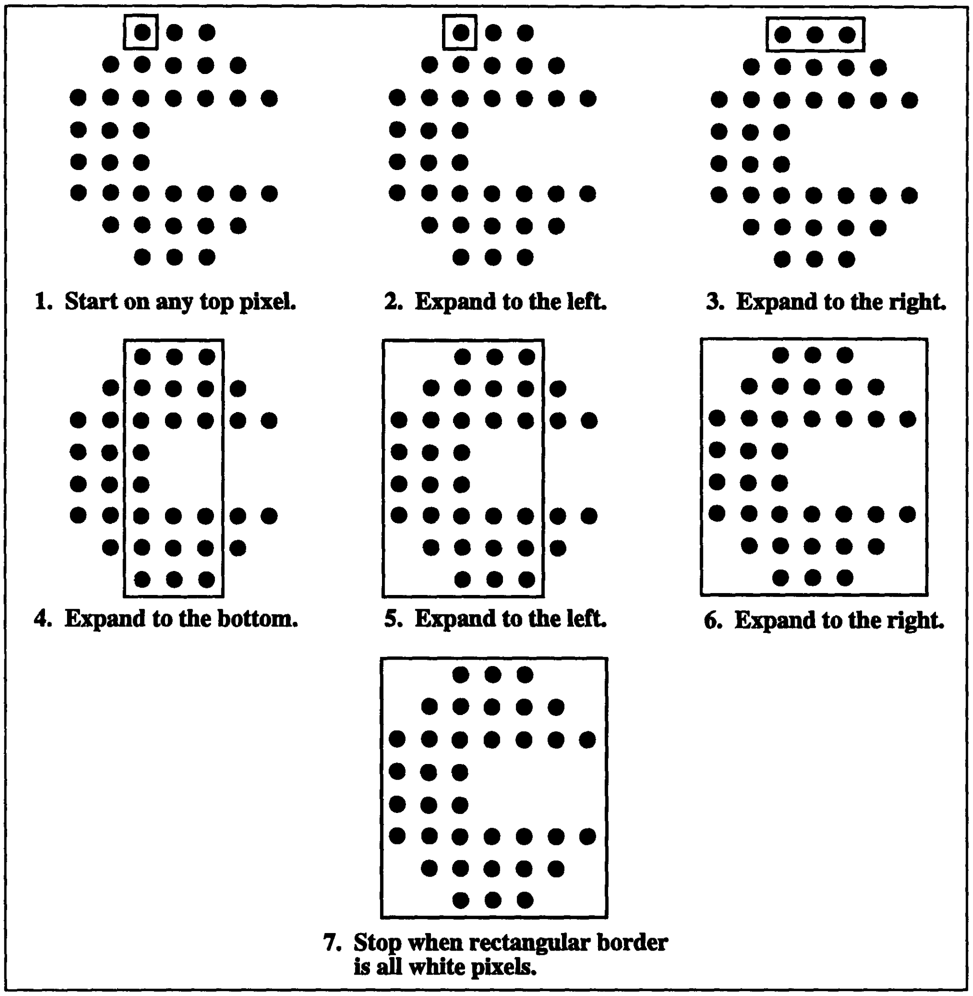


Figure 3-5: Symbol windowing.

because each typed character is completely contained within its own rectangular region, but there may be some fonts or styles (such as italics) where there can be overlap. The isolator could overlook many of the symbols or possibly group multiple symbols together, resulting in less efficient coding.

3.2 Dithered Bitmap Detection

Once all symbols have been isolated and removed, the remainder of the page is scanned by an entropy-based content classifier to detect instances of dithered bitmap fragments. The classifier is designed to locate portions of a scan line that would actually require more bits to represent with Optimized 2D Run-Length Coding than with no compression at all

(Direct Coding). To do this, it makes use of the same run-length statistics as the Optimized 2D Run-Length Coder (see Sect. 4.2) and estimates the entropy of each horizontal run in the image. Then, it looks for portions of scan lines where the average entropy per pixel is greater than one. These horizontal segments do not fit the run-length model very well and are therefore classified as dithered bitmap fragments which pass through the CAFC coder uncompressed.

Chapter 4

Coding Techniques

The following sections describe in detail the three coding techniques employed in Content-Adaptive Facsimile Coding. None of these methods are entirely novel approaches to bi-level image compression, but rather variations of previously developed methods that have been improved and optimized for a particular content. They are all designed to produce decoded content regions that in an error-free environment are either identical to the original or nearly indistinguishable, so that the reconstructed facsimile images are of high quality.

In terms of compression efficiency, each of the content coders serves a different role. The Optimized 2D Run-Length Coder, the "default" method, is intended to outperform the CCITT Recommendation T.4 Group 3 run-length coder in almost every scenario, providing a high degree of reliability. The other two techniques are used whenever possible to provide additional compression over Group 3. Symbol Matching and Substitution is especially efficient for typed text. Direct Coding is used on dithered bitmaps where run-length coding is especially ineffective.

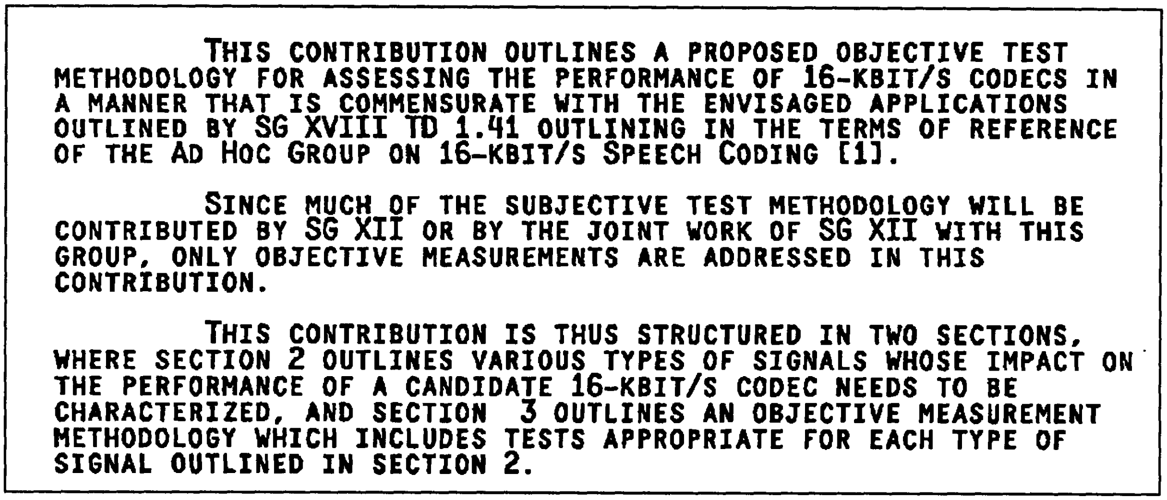
4.1 Symbol Matching and Substitution

Typewritten text consists of symbols from a fixed set of alphanumeric characters and punctuation marks, and each individual character is typically repeated numerous times on the same facsimile page. Although it is possible for small variations to occur as a result of differing scanner alignments, the symbols representing the multiple occurrences of a particular character are nearly identical. The resolution of facsimile images is high enough that these differences are difficult or impossible perceive, so that from the perspective of the person reading the document, the symbols appear exactly the same. These will be referred to as *matching* symbols.

The Symbol Matching and Substitution encoder takes advantage of this type of redundancy by maintaining a library of all unique symbols that are encountered on the page. Whenever a particular symbol is detected for the first time, it is added to the library but is left on the page to be encoded with Optimized 2D Run-Length Coding. However, when the symbol is recognized as a good match of one that is already in the library, it is considered to be a duplicate and only the library index need be transmitted. The decoder maintains an identical library so that when it receives such a message, it can decode it by simply substituting the matching symbol from the library into the image. Since the library index requires a much smaller number of bits to represent than the symbol itself, a considerable compression gain can be realized with this method.

The matching of two symbols, described later in this section, is performed through a comparison of their bitmap representations and not their association with a particular letter, number, or punctuation mark. This makes the process much simpler and does not restrict it to a particular font, style, orientation, or language. A symbol could even be something other than a conventional typewritten character, such as a logo, a very small picture, or a portion of a graphic. And, since items such as these are often repeated in facsimile documents as well, compression is still possible, making this approach very versatile; the only requirement is that symbols be repeated on page. Of course, on documents with little or no repeated characters, or where the text is in many different font sizes, styles, or orientations, it will fail to detect many matches, and the less efficient Optimized 2D Run-Length Coding technique will be used instead. It is likely that CAFC will provide the poorest compression gains for facsimile source images of this nature.

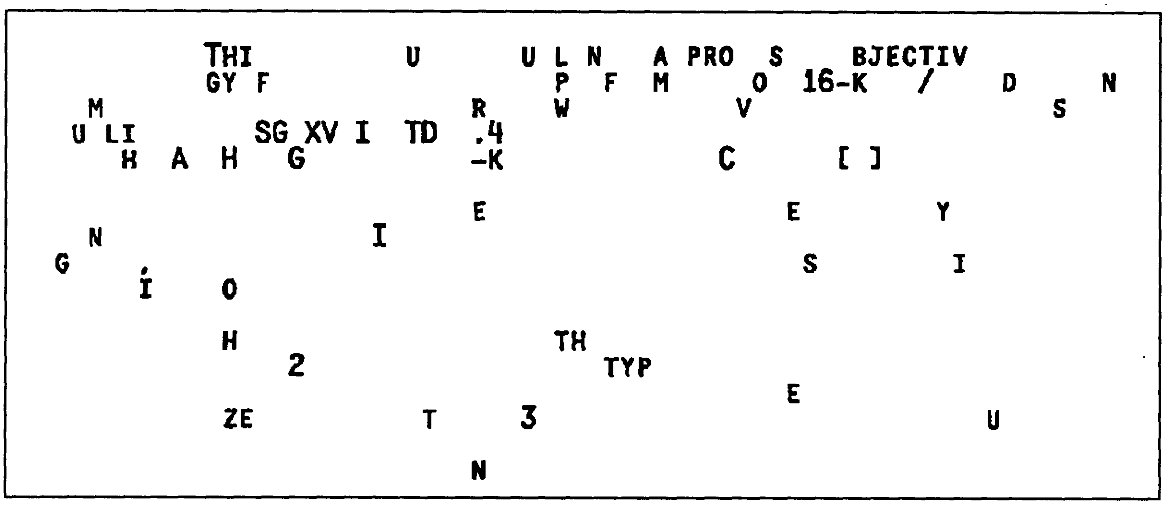
Figure 4-1 contains an example of the effect of symbol matching. It contains a source image and the corresponding image after all but the first instances of matching symbols have


THIS CONTRIBUTION OUTLINES A PROPOSED OBJECTIVE TEST METHODOLOGY FOR ASSESSING THE PERFORMANCE OF 16-KBIT/S CODECS IN A MANNER THAT IS COMMENSURATE WITH THE ENVISAGED APPLICATIONS OUTLINED BY SG XVIII TD 1.41 OUTLINING IN THE TERMS OF REFERENCE OF THE AD HOC GROUP ON 16-KBIT/S SPEECH CODING [1].

SINCE MUCH OF THE SUBJECTIVE TEST METHODOLOGY WILL BE CONTRIBUTED BY SG XII OR BY THE JOINT WORK OF SG XII WITH THIS GROUP, ONLY OBJECTIVE MEASUREMENTS ARE ADDRESSED IN THIS CONTRIBUTION.

THIS CONTRIBUTION IS THUS STRUCTURED IN TWO SECTIONS, WHERE SECTION 2 OUTLINES VARIOUS TYPES OF SIGNALS WHOSE IMPACT ON THE PERFORMANCE OF A CANDIDATE 16-KBIT/S CODEC NEEDS TO BE CHARACTERIZED, AND SECTION 3 OUTLINES AN OBJECTIVE MEASUREMENT METHODOLOGY WHICH INCLUDES TESTS APPROPRIATE FOR EACH TYPE OF SIGNAL OUTLINED IN SECTION 2.

original



residue

Figure 4-1: Example of symbol matching.

been removed, known as the *residue*. Note that towards the top of the residue, most of the characters remain intact, while near the bottom, almost all have been removed. This is because the image is encoded from top to bottom. At the top, the symbol library is initially empty, so most symbols encountered are new and are added to the library but left on the page. Towards the bottom, the library is full, so most of the symbols can be successfully matched and are removed from the image.

Figure 4-2 contains a block diagram of the Symbol Matching and Substitution encoder with the symbol isolation and detection stage included. The dashed lines indicate the flow of data and the solid lines indicate the flow of control. The source image, a raw bitmap, is scanned by the symbol isolator and the portions of the page that cannot be classified as symbols are placed in the residue. Then, in a two level comparison process, the encoder attempts to match each detected symbol with the existing library entries. The first stage is a crude comparison, where high-level properties or "features" of the symbol are used to help eliminate unlikely candidates from the search. A feature extraction procedure performs a few simple operations to obtain these properties, and then a feature matching algorithm uses them to try and "match" the symbol with the library entry. If the symbol is rejected because no matches can be made, it is added to the library for future comparisons and is also appended to the residue so that it may be encoded by one of the other methods. Otherwise, it enters the second screening phase, a more rigorous template matching process. Here, an accurate alignment and cross-correlation algorithm is used to compare the source symbol with the library entry. As before, if no match can be found, the symbol is added to the library and placed in the residue. On the other hand, if an entry with an equivalent symbol is located, then the encoding is performed with the corresponding library index number. This value is passed onto the arithmetic coder/multiplexor, the next stage of the CAFC encoder.

Symbol Matching and Substitution achieves high compression gains for typed text because it allows most of the typed characters to be represented as indices into a table rather than as a two-dimensional arrangement of pixels. Unlike most compression techniques, the redundancy that is detected and eliminated is based upon macroscopic properties of the image. That is, rather than searching for correlations between local regions of neighboring pixels, the encoder examines the entire page for essentially identical occurrences of the same pattern. This is a fairly involved task, but it is expected to prove worthwhile because it takes advantage of these previously untapped resources.

4.1.1 Feature Extraction and Matching

When processing documents containing many symbols, a large number of comparisons need to be made between newly isolated symbols and those already in the library. This can be a fairly time-consuming process that requires a significant amount of processing power. Feature matching alleviates this problem by quickly eliminating the unlikely candidates

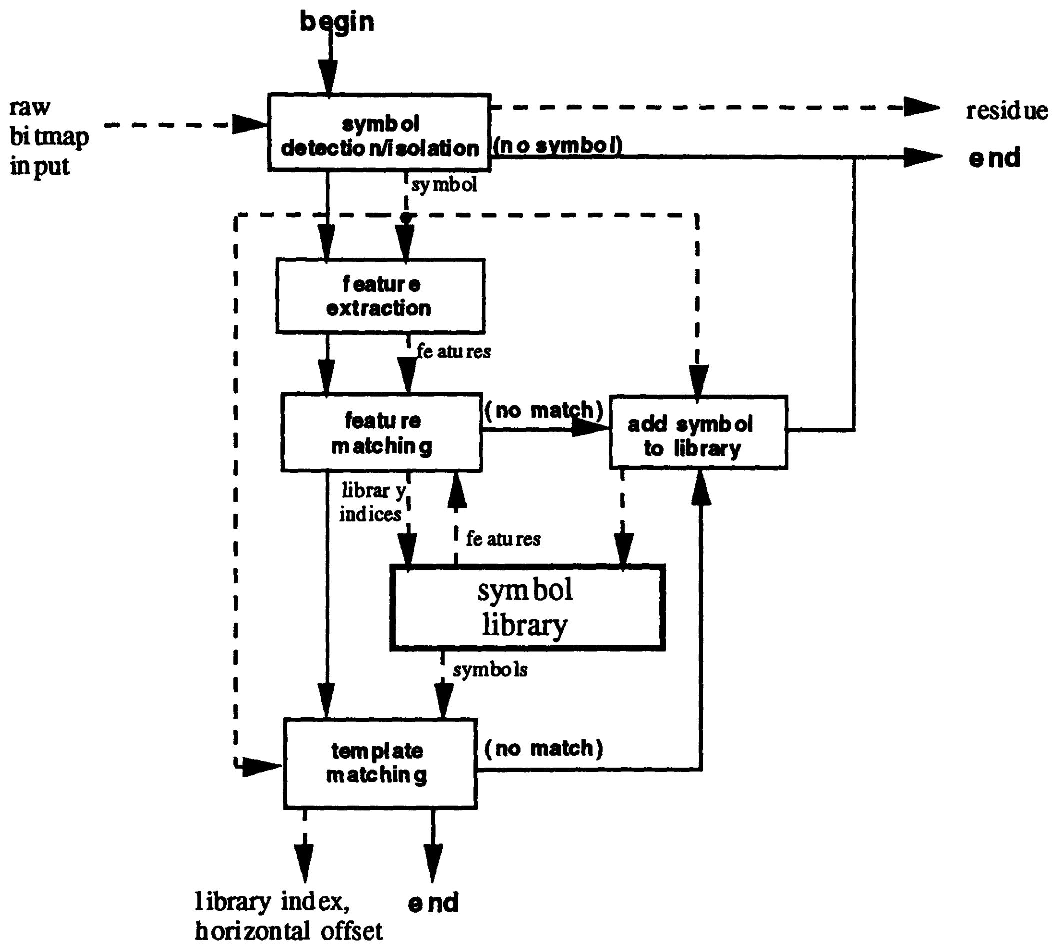


Figure 4-2: Symbol matching and substitution encoder (with symbol isolator).

in the library based upon some high-level features of the symbols. Table 4.1 lists some examples of properties of symbols that can be used during the feature matching process. In practice, a subset of these are selected for use in the encoder (see Sect. 7.2.

Feature extraction is the process of obtaining the value of a feature on a given symbol. It is useful to introduce a notation for referring to these values. For symbol s, the value of the nth feature is denoted as $F_n(s)$. For example, using the numbering in Table 4.1, the number of black pixels in symbol s_1 would be $F_3(s_1)$.

Feature matching is the process of comparing the feature values extracted from two symbols to determine if they are likely to not match. In order to do this, feature matching makes use of the "absolute difference" between two symbols for a given feature, defined as follows:

$$D_n(s_1, s_2) = \mid F_n(s_1) - F_n(s_2) \mid \quad .$$

#	Feature Name
1	Width
2	Height
3	Number of Black Pixels
4	Number of White Pixels
5	Number of Horizontal Run-Lengths
6	Number of Vertical Run-Lengths
7	Horizontal Moment (Center of Mass)
8	Vertical Moment (Center of Mass)
9	Average Width

Table 4.1: Potential features for feature matching.

For a given feature, the absolute difference between the two symbols is compared with a rejection threshold, r_n. If $D_n(s_1, s_2) \geq r_n$, then symbols s_1 and s_2 are considered mismatches and no more comparisons are necessary. Otherwise, the process continues with the remaining features until there is a mismatch or the list is exhausted. In the latter case, feature matching is unable to differentiate s_1 and s_2, so template matching procedure is applied.

The features that are chosen must be very simple so that they can be extracted easily and rapidly, yet diverse enough so that they are effective at eliminating as many non-matching library entries as possible. Section 7.2 describes a procedure that has been devised for selecting the optimal set of features and corresponding rejection thresholds. It is important to note that feature matching by itself does not contribute to the process of image compression. Rather, it provides a mechanism for bypassing many of the symbol comparisons serviced by template matching, thereby decreasing the necessary computational resources. Thus, feature matching has an unessential but very practical role in CAFC.

4.1.2 Template Matching

When an isolated symbol and a library entry symbol pass through all the stages of feature matching process, the template matching algorithm is applied to ensure that the symbols truly do appear identical before they are officially declared a match. Since CAFC's only lossy compression technique is Symbol Matching and Substitution, its ability to produce extremely high quality reconstructed images relies heavily on this final screening process.

First, template matching adjusts for any slight variation in the positions of the two symbols that may have occurred during scanning by computing their "centers of mass", defined for symbol s as follows:

$$c_x(s) = \frac{\sum_{x=0}^{W-1} \sum_{y=0}^{L-1} x s(x,y)}{\sum_{x=0}^{W-1} \sum_{y=0}^{L-1} s(x,y)} \ ,$$

$$c_y(s) = \frac{\sum_{x=0}^{W-1} \sum_{y=0}^{L-1} y s(x,y)}{\sum_{x=0}^{W-1} \sum_{y=0}^{L-1} s(x,y)} \ .$$

where s(x, y) represents the pixel at coordinate (x, y) within symbol s (1 = black, 0 = white). W and L are the larger width and length, repectively, of the two symbols; pixels referenced outside the boundries of a symbol are assumed to be white. Next, the template matcher computes the square of the cross-correlation, λ^2, between the two symbols:

$$\lambda^2(s_1, s_2) = \frac{\sum_{x=0}^{2W-1} \sum_{y=0}^{2L-1} s_1([x/2],[y/2]) s_2([x/2 - c_x(s_1) + c_x(s_2)],[y/2 - c_y(s_1) + c_y(s_2)])}{16(\sum_{x=0}^{W-1} \sum_{y=0}^{L-1} s_1(x,y))(\sum_{x=0}^{W-1} \sum_{y=0}^{L-1} s_2(x,y))} \ .$$

This formula takes into account the fact that it is possible for two symbols to be misaligned by a fraction of a pixel by working within a grid with twice the horizontal and vertical resolution of the source image. Since a translated coordinate can be fractional as well, each component is rounded; the notation $[x]$ is used to represent the integer closest to x.

If the cross-correlation distortion $\lambda^2(s_1, s_2)$ is above some established threshold, r_t, the source and library symbols are considered a match, and the efficient coding can take place. Otherwise, residue coding is necessary.

4.1.3 Library Maintenance

The symbol library is a large data structure that is used to store all of the unique symbols that have occurred so far in the image. It is initially empty at the top of the page and gradually fills up as newly-encountered symbols are added. The information stored in each library entry is listed in Table 4.2. Obviously the most important item is the bitmap representation of the symbol itself, necessary for performing comparisons with future potential matching symbols and for decoder substitution. Also needed is an arithmetic coding element number, a unique identifying integer used during the arithmetic coding and decoding processes (described in Chap. 5).

bitmap representation of symbol
features of symbol
arithmetic coding element number
total number of appearances.

Table 4.2: Contents of a library entry.

Two additional items may be included in order to reduce the amount of computational overhead associated with Symbol Matching and Substitution. The first is the numerical value of all of the symbol's features. Since these will have already been computed before any new symbol is added to the library, they are readily available for storage. By retaining this information, it is not necessary to perform feature extraction on these symbols in the future when comparing them with newly-isolated symbols. The other useful item is the number of times a symbol has appeared so far on the page. This information can be used to keep the library sorted in such a manner that the most frequently occurring symbols are always searched first. This decreases the expected number of comparisons necessary to match a symbol and decreases the probability of a mismatch.

The library structure is able to support two basic operations: adding a new symbol to the library and fetching the information from the library back one entry at a time. The same library structure is shared by both the CAFC encoder and decoder.

4.2 Optimized 2D Run-Length Coding

Run-length coding is a well-known technique that is used in many forms of image compression, especially facsimile. It is based on the observation that for any given scan-line on a page, there tend to be long "runs" of black and white pixels. These strings of pixels occur because in typical fax documents, black pixels are clustered together to form items in the foreground while contiguous white pixels fill the background regions. In run-length coding, a scan line is encoded as a series of numbers representing the lengths of these runs rather than as individual pixels, thus resulting in significant compression gains. Figure 4-3 contains an example run-length coding on a portion of a facsimile image scan line. The

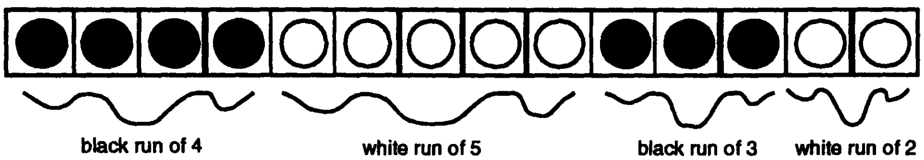


Figure 4-3: Run-length coding of a portion of a scan line.

encoded run-lengths alternate between white and black runs across the scan line.

Another important characteristic of facsimile images is that some run-lengths appear more frequently than others. For example, one would expect a significantly larger number of short black run-lengths than long black run-lengths because of the many thin lines (pen strokes) on the page. A similar effect can be observed for white run-lengths, which occur most frequently in-between black pen strokes or in association with blank image scan-lines. Figure 4-4 shows the run-length distributions for both white and black runs on the set of test documents in Appendix D.

Because the run-length distributions are not flat, a coding scheme that gives an equal number of bits to each run-length would have some statistical redundancy and would therefore be sub-optimal. Huffman coding is a technique which takes advantage of these unbalanced statistics by assigning a variable-length codeword to each run-length. Runs that have a high probability of occurring are assigned shorter codewords, while runs with appear infrequently are given longer codewords. The result is a much more efficient coding scheme. The Group 3 standard uses a slightly modified version of Huffman run-length coding that is easier to implement on limited hardware platforms. It is capable of achieving compression gains on the order of 6 to 12, depending upon the run-length distributions of the particular image.

Because of its effectiveness, run-length coding was chosen as CAFC's coding technique for handwriting and graphics. However, two additional modifications were made to increase the compression ratio even further. The first is the replacement of the Huffman variable-length coding with a newer technique known as arithmetic coding. Described in more detail in

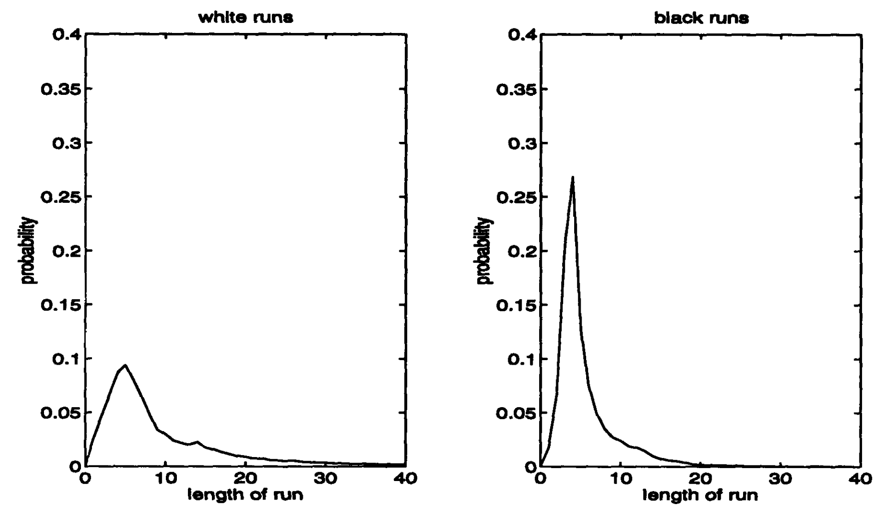


Figure 4-4: Run-length statistics for a sample set of images.

Chap. 5, arithmetic coding overcomes some of the limitations of Huffman coding, allowing it to achieve higher compression gains.

The second improvement on conventional run-length coding is the extension of its model into two dimensions. Run-length coding is a one-dimensional scheme because it only operates on runs in the horizontal direction only. In the majority of documents, however, there are significant correlations between adjacent scan lines in the vertical direction as well. A number of algorithms have already been developed to exploit these properties [7] [8] [9]. However, since the primary focus of this work has been the development of efficient coding for typed text, a relatively simple approach has been chosen for this preliminary version. Rather than run-length coding the residue directly, CAFC run-length codes the *difference* between the pixels in adjacent scan-lines. The difference between two pixels is equal to 0 if the pixels are of the same color and 1 if they are different. Instead of parsing a scan-line into runs of black and white, it uses information from the scan-line immediately above to parse the scan-line into runs of 0s ("same" runs) and 1s ("different" runs).

Figure 4-5 shows the 2D run-length distributions for the same set of documents. Clearly, the peaks are much sharper, indicating that there is a higher degree of redundancy in the images with this model than with one-dimensional run-length coding. Because of this, the run-lengths will take fewer bits to represent after passing through the entropy coding stage. Using the difference between adjacent scan-lines is therefore an extremely simple way of

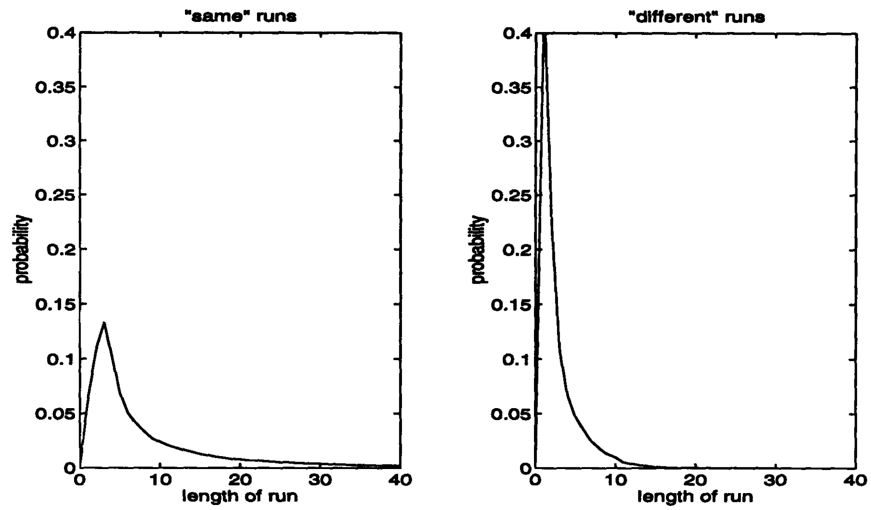


Figure 4-5: 2D run-length statistics for a sample set of images.

exploiting the two-dimensional redundancies in images.

The run-length statistics used in Optimized 2D Run-Length Coding are considerably different from those used in Group 3 for another reason. Because the images processed by the 2D run-length coder are residues, they are void of typed text and dithered bitmaps, which have already been removed and encoded by the other methods. A page without these contents has somewhat different run-length statistics than those of a complete page.

For both of the above reasons, a new training set is generated consisting of residues of the original training set. The entropy coding can then be performed more efficiently, taking advantage of the vertical correlations as well as reduction in content diversity. Even better, the arithmetic coder used in CAFC is adaptive and automatically updates these run-length statistics based upon the statistics of the particular image being processed. This is explained in detail in Sect. 5.3.

Run-length coding achieves compression by eliminating redundancy on a microscopic level, focusing only on small regions of pixels at a time. This is a good choice for handwriting and graphics which have little or no repetition throughout a page but have very predictable local properties.

4.3 Direct Coding

Direct coding is the straightforward conversion of pixels to bits for dithered bitmap fragments. It does not actually perform any compression and is so simple that it is hardly a coding scheme at all. Instead, it is used as a preventative measure for unusual instances when the distribution run-lengths present is so abnormal that the fragment would be otherwise expanded. Scan-line segments that meet this criterion are likely to contain a large number of very short runs so that it does not fit the run-length model well.

Chapter 5

Multiplexing and Arithmetic Coding

The final stage of the CAFC encoder uses arithmetic coding to combine the individual outputs from the three different content coders into a single stream of output data. Section 5.1 describes the relationship between the CAFC page model and the order in which contents are combined in the encoded output stream. Sections 5.2 and 5.3 provide a brief tutorial on arithmetic coding. Finally, Sect. 5.4 puts these ideas together to explain the specifics of how arithmetic coding is used to multiplex the individual contents in CAFC.

5.1 Content Multiplexing

According to the CAFC page model, an image consists entirely of the following objects: symbols, dithered bitmap fragments, and "same" and "different" runs. These page components fill up the entire area of the image in the form of horizontal portions of a scan line. Dithered bitmap fragments and "same" and "different" runs are naturally horizontal segments. In the case of symbols, the model specifies that the bitmap representation fills up no space on the page. Instead, the symbol is superimposed over the presumably white space below it. Since by definition, a symbol must be surrounded by white pixels, this area

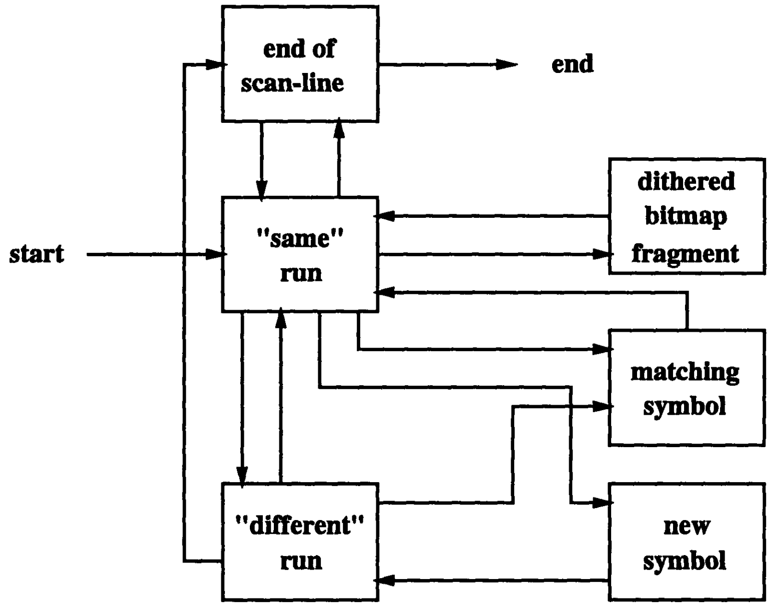


Figure 5-1: CAFC multiplexing state diagram.

would most likely consist of "same" runs that go "under" the black symbol.

The CAFC encoder scans the source document from left to right and then top to bottom, breaking the page down into its content components. After an object is classified, it is encoded with the appropriate compression scheme and passed on to the multiplexing stage, which combines the components of the image into a single data stream in the same order that they were read from the page.

Figure 5-1 contains the state diagram that is used when multiplexing data from the three individual coding algorithms. The beginning of each scan-line is assumed to begin with a "same" run. This is true most of the time because the majority of documents have a white border on the left side of the page. If the line begins with a symbol or dithered bitmap, a "same" run of length zero is encoded first. Following a "same" run can be a "different" run, a new symbol, a matching symbol, or a dithered bitmap fragment. It could also be followed by another "same" run if and only if the run ends at the end of the scan-line (triggering the beginning of a new line). New symbols are always followed by "different" runs, because the detected starting pixel is always black and the pixel above it must always be white (since symbols are surrounded by white pixels). Matching symbols, on the other hand, are always followed by a "same" run, because white pixels run underneath and above the symbol. Finally, dithered bitmap fragments are always followed by a "same" run. The end of the page is reached after the last "same" run on the last scan-line.

The CAFC decoder's demultiplexor uses the same model to separate the data stream back into three separate components. The Symbol Matching and Substitution decoder can correctly update its library because new unique symbols are detected in the same order that they are presented to the receiver, and repetitions of a previous symbol will never appear before it has been added to the library. Also, real-time facsimile communications is possible because the page is transmitted from top to bottom, and only a few scan lines of buffering are required for symbols.

5.2 Arithmetic Coding

This section provides a brief introduction to arithmetic coding (AC). For a more in-depth explanation, refer to one of the many publications on this topic [13] [14]. Enough background is presented here to explain the basic principles of arithmetic coding and how it is used by CAFC to multiplex and entropy-code the encoded contents. In this description, the term "element" is used to refer to what most of the literature on model-based entropy-coding defines as a "symbol." This is to avoid the obvious confusion with the definition of "symbol" that has been used up to this point.

An entropy coder takes as input a stream of *elements* taken from a fixed alphabet and converts them to a stream of output bits. A *model* consists of all of the possible input elements and their respective probabilities. The encoder applies a model to each input element to produce an output with the minimal number of bits. Suppose a model contains N elements named e_1 through e_N with probabilities P_1 through P_N. It is a necessary condition that the probabilities add up to unity:

$$\sum_{n=1}^{N} P_n = 1 \ .$$

The optimal number of bits necessary to represent element e_n is equal to $-\log P_n$ where the logarithm is taken to base 2. The optimal number of bits necessary to encode a stream of M elements x[1] through x[M] is just the sum over all elements:

$$\text{optimal number of bits} = \sum_{m=1}^{M} -\log P_{x[m]} \quad .$$

For example, if a model consists of the letters **a**, **b**, and **c** with probabilities 0.5, 0.3, and 0.2, respectively, then the message "**abacab**" would require a minimum of -log 0.5 - log 0.3 - log 0.5 - log 0.2 - log 0.5 - log 0.3 = 8.796 bits.

Since each element occurs with a probability of P_n, the average number of bits required to encode an element is equal to the following expression:

$$\sum_{n=1}^{N} -P_n \log P_n \quad .$$

The most straightforward way to convert a stream of elements into bits is to use a unique fixed-length codeword to represent each possible element. If the coding model contains an alphabet of N elements, then each codeword would require at least $\log N$ bits. In the above example, this would be $\log 3 = 1.58$ bits, which would have to be rounded up to 2. If **a** is encoded as 00, **b** as 01, and **c** as 10, the above message would be encoded as 000100100001, a total of 12 bits. This approach does not take into account the probability of each element and therefore achieves no compression whatsoever.

A better approach is Huffman coding, which assigns a variable-length codeword to each element. The codewords are selected so that the shorter codes refer to the most probable elements and the longer codes refer to the least probable ones. An optimal Huffman coding scheme for the above example would be to encode **a** as 0, **b** as 10, and **c** as 11. The above message would be encoded as 010011010, a total of 9 bits. While this is significantly better, the reductions that Huffman coding achieves can never approach the theoretical limit. This is due to the restriction that each codeword must be of an integral length. Most of the time, however, the ideal number of bits for representing a particular run-length falls in-between two consecutive integers. Because of this, small compromises must be made when the Huffman codewords are assigned, leading to suboptimal coder performance.

Arithmetic coding avoids this limitation by abandoning the notion of codewords altogether. Instead, a message is represented by an interval of real numbers between 0 and 1. Based upon the coding model, each element is assigned a unique range within this interval [0,1] in such a manner that none of the ranges overlap. Table 5.1 lists the model and associated

element	probability	range
a	0.2	[0, 0.2]
b	0.3	[0.2, 0.5]
c	0.5	[0.5, 1.0]

Table 5.1: Example fixed model for alphabet [**a**, **b**, **c**].

subintervals for the previous example.

Initially, the range for the message is the entire interval [0,1]. As each element is encoded, the range is narrowed to a smaller interval based upon the range of the element. For example, with the above model, encoding the element **a** would reduce the range to [0, 0.2]. Encoding another **a** would further reduce it to [0, 0.04]. In general, if the interval was previously [x_1, x_2] and the element to be encoded has the associated range [y_1, y_2], the new interval is [$(x_2\text{-}x_1)y_1 + x_1$, $(x_2\text{-}x_1)y_2 + x_1$]. Encoding the entire message **abacab** produces the following results:

Initially		[0,	1]
After seeing	**a**	[0,	0.2]
	b	[0.04,	0.1]
	a	[0.04,	0.052]
	c	[0.046,	0.052]
	a	[0.046,	0.0472]
	b	[0.04624,	0.0466]

Decoding this message is fairly straightforward. The decoder simply compares the interval with the ranges in the model to determine which was the first element in the message. It then "removes" this element from the message by computing a new interval [$(x_1\text{-}y_1)/(y_2\text{-}y_1)$, $(x_2\text{-}y_1)/(y_2\text{-}y_1)$] and the process repeats.

As it turns out, the entire message can be uniquely decoded by any number within the calculated interval. The longer the message, the more bits it takes to represent this number. For a large number of elements, the number of bits approaches the theoretical minimum. Thus, arithmetic coding is an optimal technique for encoding a stream of elements if their associated probabilities are known.

The implementation of arithmetic coding in a practical system is a bit more complicated. On a computer system, real numbers are best represented with floating point variables. These offer fairly limited precision, making them useless for encoding and decoding long messages. It is much more desirable to use integer arithmetic if possible. In addition, operating on intervals in the above manner would require an amount of storage proportional to the length of the message, since the number of bits required to represent the interval is equal to the *total* number of encoded bits. Again, this is not feasible for long messages. Fortunately, methods have been developed for performing arithmetic coding of arbitrarily long messages using only integer arithmetic. CAFC utilizes such techniques, which are described adequately elsewhere [13] [14].

5.3 Adaptive Arithmetic Coding Models

At any stage of the coding process, the arithmetic encoder takes as input the element to encode, a coding model, and the present state of the coder and generates as an output a stream of bits. The corresponding arithmetic decoder takes as input the stream of bits to decode, the same coding model, and the present state of the decoder to reconstruct the element. This structure is extremely flexible because it does not restrict the model to be fixed over time. After each element is encoded, is possible to revise the model with an updated alphabet of elements or associated probabilities. As long as the decoder has access to the new model at each stage, it can generate the correct stream of output elements.

In fact, because there is no coding delay associated with arithmetic coding, the new model can even be a function of the elements transmitted. In an *adaptive* arithmetic coder, after each element is encoded, the probability of that element is increased in the coding model. That way, the next time the same element appears, it will require fewer bits to encode. The arithmetic decoder updates its model in the exact same manner based upon the decoded elements, so that it is always in accordance with the encoder. Adaptive arithmetic coding works well for encoding streams of elements where the relative probabilities of each element are fairly constant but unknown, because it eventually "adapts" to the appropriate statistics after a large enough set of elements has been encoded.

5.4 Arithmetic Coding Model for CAFC

The arithmetic coder in CAFC serves two major functions. Most importantly, it is the mechanism used to actually merge the three independent data streams from Symbol Matching and Substitution, Optimized 2D Run-Length Coding, and Direct Coding. Furthermore, it provides additional compression by exploiting the relative probabilities each run-length or symbol.

As explained in the previous sections, arithmetic coding efficiently represents a sequence of items from a given alphabet by using information from a model consisting of the alphabet and probability of occurrence of each entry. In CAFC, there are actually three different arithmetic coding models, all shown in Table 5.2, where W represents the width of the image in pixels and N indicates the total number of symbols in the symbol library. The particular model and element that are used depend upon the state of the encoder and the content that is to be encoded. *AC Model0* is used to encode "same" runs, matching symbols, or codes to indicate a new symbol (**new-symbol**), a dithered bitmap fragment (**bitmap**), or the end of the page (**escape**). *AC Model1* provides the capability of encoding "different" runs, matching symbols, or a code to indicate a dithered bitmap fragment. Finally, *AC Model2* encodes bitmap fragments as well as a code to signify the end of one (**last-pixel**).

The complete arithmetic coding state diagram for CAFC is shown in Fig. 5-2. The dashed boxes represent the contexts of the three different coding models. As the encoder moves from one state to the next, the element shown in italics along the indicated path is encoded using the appropriate model. In addition, this is an adaptive coder, and the model(s) indicated in a typewriter font are updated with an increased probability for the encoded element. Also, when a new symbol is detected, a new element is created in *AC Model0* and *AC Model1* for encoding future instances of matching symbols (after the **new-symbol** element is encoded).

Performing both the multiplexing and entropy-coding with a single arithmetic coder is desirable because of its simplicity and flexibility. With the exception of dithered bitmap fragments, all possible image components are contained within a single alphabet, eliminating the need for headers or tags to be incorporated into the data stream. The entropy-coding

AC Model0		AC Model1		AC Model2	
#	element	#	element	#	element
0	"same" run of 0	0	"different" run of 0	0	a single white pixel
1	"same" run of 1	1	"different" run of 1	1	a single black pixel
2	"same" run of 2	2	"different" run of 2	2	**last-pixel**
.		.			
.		.			
.		.			
W	"same" run of W	W	"different" run of W		
W+1	**escape**	W+1	------------		
W+2	**new-symbol**	W+2	------------		
W+3	**bitmap**	W+3	**bitmap**		
W+4	symbol #1	W+4	symbol #1		
W+5	symbol #2	W+5	symbol #2		
W+6	symbol #3	W+6	symbol #3		
.		.			
.		.			
.		.			
W+N +3	symbol #N	W+N +3	symbol #N		

Table 5.2: CAFC's three arithmetic coding models.

of run-lengths is no longer integrated with the run-length coding process, as it is in Group 3 Modified Huffman Run-Length Coding. This approach can also take advantage of the redundancy associated with the relative probabilities of each of the unique symbols contained in the symbol library. For example, on a typical typewritten document, the letter "e" appears far more frequently than the letter "q." Because CAFC's arithmetic coder is adaptive, the encoded "e"s will end up taking fewer bits than the encoded "q"s.

Adaptive coders often perform quite poorly at the beginning of the encoding process because a representative set of statistics has not yet been generated. To help alleviate this problem, the CAFC elements for run-lengths are initially "weighted" based upon the statistics of a large collection of sample images. The encoder eventually adapts to the real statistics of the image (and performs better) after enough of the page has been encoded. More importantly, it does not initially perform any worse than a non-adaptive run-length coder, which uses the same predefined statistics for the entire document.

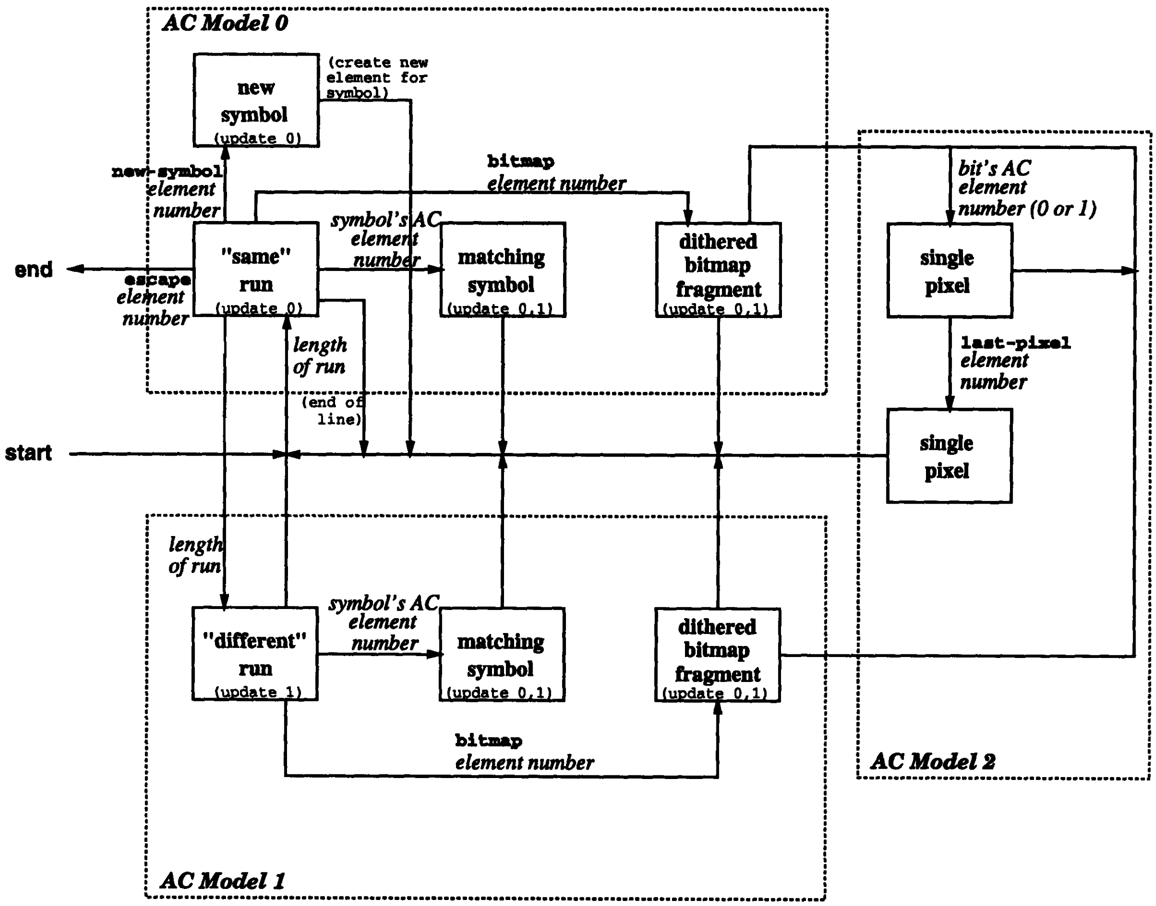


Figure 5-2: Arithmetic coding state diagram.

Chapter 6

CAFC Decoder

The CAFC decoder reverses the compression process, converting an encoded representation of a facsimile page back into the form of an image. This process is fairly straightforward, following directly from the design of the encoder.

The various building blocks of the image reconstruction algorithm are depicted in Fig. 6-1. A content splitter first separates the encoded data stream back into the basic elements of its three constituent contents: symbols, "same" or "different" runs, and dithered bitmap fragments. These are then individually converted back into bitmap form by the appropriate content decoder, either Symbol Substitution, Optimized 2D Run-Length Decoding or Direct Decoding. Finally, an Image Constructor combines the decoded image objects together to form a single reconstructed output page.

The Symbol Matching and Substitution decoder converts a stream of library index numbers (encoded as AC element numbers) back into symbols to insert into the destination image. In order to do this, it must maintain its own identical copy of the symbol library. Since at any given point in time the library consists of symbols which have already been encoded by another method, this process can be performed adaptively. The CAFC encoder multiplexes the data from each of the three coding techniques in such a manner that a causal system exists between the source page and the library. The decoder can capture all new symbols from the partially generated destination image by applying the same symbol isolation algorithm

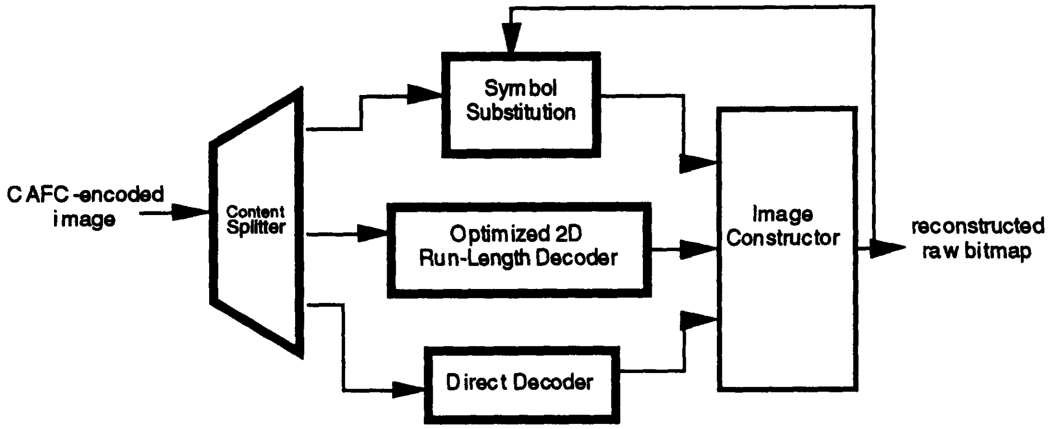


Figure 6-1: CAFC decoder block diagram.

as the encoder. Because both libraries are updated in exactly the same manner based upon identical images, they should be in accordance at all times. The actual decoding process is then very straightforward and involves nothing more than a simple table lookup operation. For each library index number in the encoded data stream, the appropriate symbol bitmap is extracted from the library and superimposed directly onto the reconstructed output page.

Because the content multiplexing is performed inherently in the arithmetic coding process, the content splitter is directly implemented with an arithmetic decoder. Each decoded data item is then passed into one of three different reconstruction algorithms, depending upon its content. The Symbol Substitutor takes as an input the library index number of an encoded symbol and performs a table lookup into the symbol library, producing a two-dimensional bitmap representation of the symbol. The Optimized 2D Run-Length Decoder converts a given run-length into a run of "same" or "different" pixels to be inserted into the output image. The Direct Decoder generates a horizontal segment of pixels from a stream of input bits using each bit as the representation for a single pixel.

All generated image elements are combined together by the Image Constructor. Since in the CAFC page model, the basic element of each content is a portion of a scan line, the Image Constructor simply fills up the output page with these non-overlapping segments. The only exception is typed text symbols, which are two-dimensional and therefore extend below the scan line segment. Symbols are incorporated into a reconstructed image with a pixel-wise inclusive OR function that preserves black pixels. They are effectively "placed on top of" whatever occupies the space where they belong, which should be a region of white pixels. The resulting image is the final output of the CAFC decoder, a reconstructed version of the original source page. If the encoding was done well, the two should appear almost identical.

Chapter 7

CAFC Parameter Optimization

The two most important performance measures of any compression algorithm are the amount of compression it can achieve and the quality of the output generated by the decoder. There is, of course, an inherent tradeoff between these properties, and the objective is usually to develop a technique that meets some standard for one of them while maximizing the performance of the other. In the case of DCME facsimile communications, it is more important that the quality of the reconstructed image be high (though not necessarily perfect). The goal then is to maximize the compression ratio under this constraint.

There are a number of different adjustable parameters in Content-Adaptive Facsimile Coding which need to be preset in advance. In some cases, the values that are used are not particularly critical. However, most of the time the overall compression ratio and/or reconstructed image quality depend very heavily on the selections that are made.

This chapter discusses the various choices that are available in the design of a CAFC coding system and explains the criteria and procedures that were developed for optimizing the performance of the coder based upon these parameters. Within each section, the values that were chosen for the preliminary version of the algorithm are summarized.

7.1 Selection of Symbol Isolation Technique

Section 3.1 described the process of symbol isolation and detection for extracting typed text symbols from a source image. For CAFC, three different approaches were developed, symbol filling, symbol tracing, and symbol windowing. Since only one symbol isolation algorithm is needed, the different techniques need to be evaluated so that the best one can be selected for CAFC.

In terms of compression performance, the most important property of a symbol isolator is its ability to detect and isolate all of the symbols on a page. Symbol filling and symbol tracing can both do this (though the results may differ slightly), while symbol windowing cannot. On the other hand, it is desirable to be able to implement facsimile compression on limited hardware platforms so as reduce the cost and increase the mobility of such systems. Symbol windowing requires very little in the way of computational resources, while the other two techniques require storage space and processor cycles that grow linearly or quadratically with the size of the symbol.

Table 7.1 summarizes the space and processing power requirements of each of the three algorithms, as well as their performance at isolating symbols from a large collection of test images (from Appendix D). As expected, symbol filling was able to detect the most symbols while symbol windowing detected the fewest. However, the disparity was very small, indicating that all three algorithms did a perfect or nearly perfect job. And since symbol windowing requires the smallest amount of storage space, it appears to be the best choice for symbol isolation.

It is interesting to note that the execution times of the simulations did not vary considerably when different isolation techniques were used. This indicates that symbol isolation does not contribute significantly to the total amount of processing power required by CAFC.

Name	complexity	Total # Symbols Detected	time	space
Symbol Filling	medium	11011	$O(L \times W)$	$O(L \times W)$
Symbol Tracing	high	10999	$O(L+W)$	$O(L+W)$
Symbol Windowing	low	10898	$O(L+W)$	$O(1)$

Table 7.1: Comparison of symbol isolation techniques.

7.2 Feature Selection and Matching Criteria

The feature matching component of Symbol Matching and Substitution has the important task of detecting probable symbol mismatches before they are passed onto the more computationally demanding template matcher. As explained in Sect. 4.1.1, this involves the extraction of a number of high-level features from the symbols. For each feature F_n, the absolute difference D_n is compared against a rejection threshold r_n to determine whether or not to reject a symbol during a library search, where

$$D_n(s_1, s_2) = \mid F_n(s_1) - F_n(s_2) \mid \quad .$$

In the development of a feature matching algorithm, it is necessary to select a set of features and a corresponding set of rejection thresholds. These parameters should be selected so that the feature matcher is effective at eliminating differing symbols and passing matching symbols. While it is certainly undesirable for the feature matcher to incorrectly pass differing symbols, it is absolutely critical that it not reject matching symbols. The former would merely result in the need to perform template matching, costing processor time, but not affecting the compression gain. The latter would lead to the misinterpretation of the two matching symbols as differing symbols, resulting in the creation of redundant library entries and severe reduction in the amount of compression that is achieved.

For this reason, the feature matcher should be designed to be very conservative, with rejection thresholds set high enough to pass the overwhelming majority of matching symbols. If $P(D_n(M_1, M_2) = d)$ is the probability that the absolute difference between feature n of any two matching symbols M_1 and M_2 is equal to d, then the probability of false rejection for feature n, Pf_n, is equal to the following expression:

$$1 - \sum_{d=0}^{r_n - 1} P(D_n(M_1, M_2) = d) \quad .$$

After all N_F features have been extracted and tested, the overall probability of false rejection for the feature matching system, Pf, can be easily computed:

$$Pf = 1 - \prod_{n=1}^{N_F} (1 - Pf_n) \ . ^*$$

To make the symbol matching of CAFC as robust as possible, Pf is selected to be very low, 0.005, so that only 1 in 200 pairs of matching symbols should be falsely considered a mismatch. To select the 5 best features, the probability of false detection for each feature should therefore be targeted at 0.025. Using these assumptions, the rejection thresholds are chosen to be the values that satisfy

$$0.005 = 1 - \sum_{d=0}^{r_n - 1} P(D_n(M_1, M_2) = d) \ .$$

Once the rejection thresholds have been established with this procedure, the chances that a pair of matching symbols will be falsely rejected should be the same for all of the features. It is then possible to evaluate the features based upon their ability to correctly reject two differing symbols. If $P(D_n(S_1, S_2) = d)$ is the probability that the absolute difference between feature n of any two differing symbols S_1 and S_2 is equal to d, then the *effectiveness*, e_n, of feature n is defined as follows:

$$e_n = 1 - \sum_{d=0}^{r_n - 1} P(D_n(S_1, S_2) = d) \ ,$$

where e_n is just the probability that the feature matcher will correctly reject two differing symbols. The overall effectiveness of the symbol matcher, e, is a function of the effectiveness of each feature:

$$e = 1 - \prod_{n=1}^{N_F} (1 - e_n) \ . ^*$$

For CAFC, the features are selected from the list of nine easily-extracted features listed earlier in Table 4.1. In order maximize the overall effectiveness of the feature matcher in CAFC, the five features that are selected are those with the five highest e_n's.

*These equations assume mutual statistical independence between all of the features of a symbol. This is a crude and inaccurate model, but is still useful for estimating the overall sensitivity and effectiveness of feature matching.

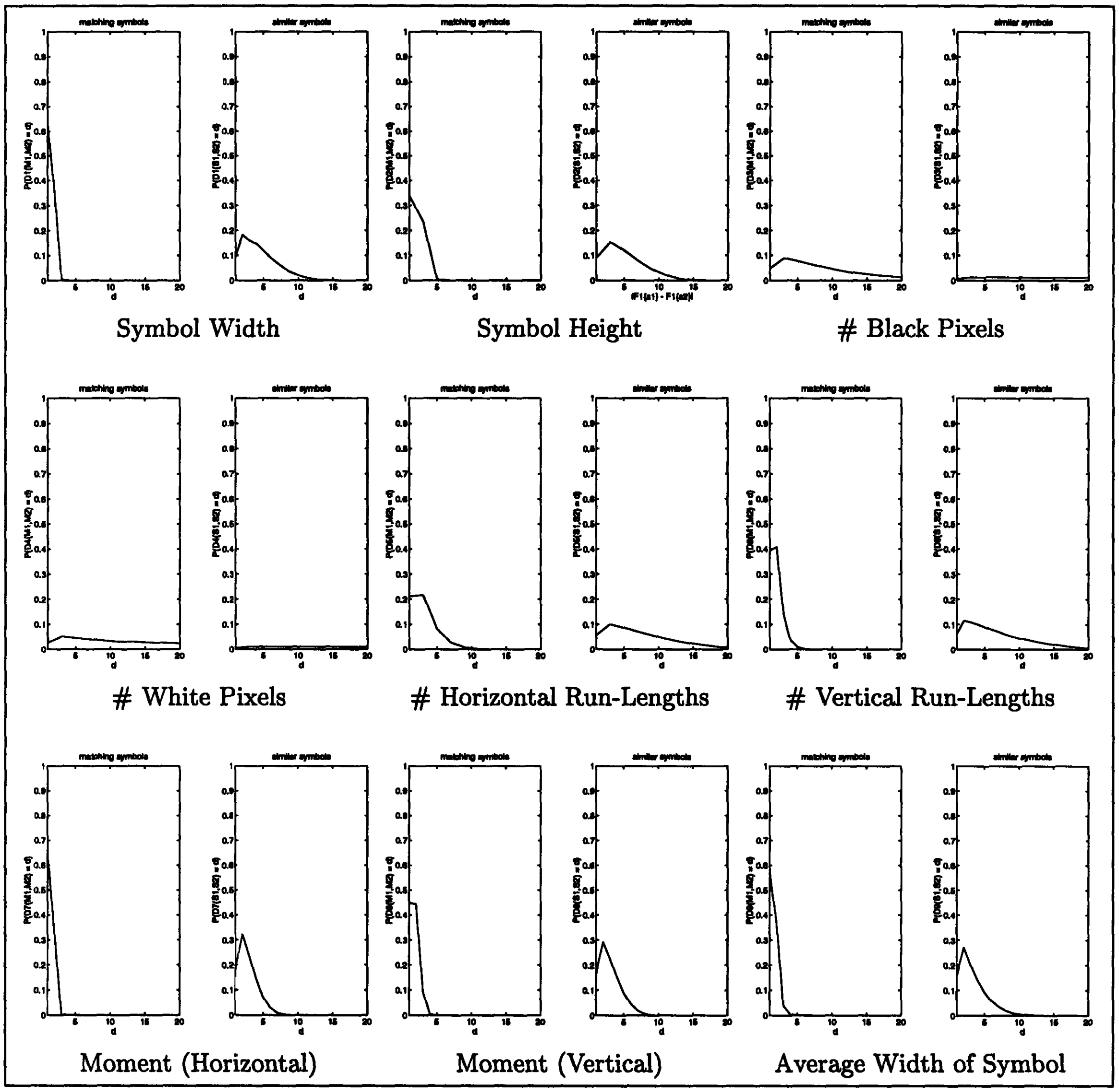


Figure 7-1: Statistics of features on test symbols.

In order to determine $P(D_n(M_1, M_2) = d)$ and $P(D_n(S_1, S_2) = d)$, a set of typed text test images was generated and scanned. The eight pages shown in Appendix C contain 8 repetitions of 78 different characters in 3 fonts, 3 styles, and 3 sizes. To obtain $P(D_n(M_1, M_2) = d)$, all instances of the same symbol are compared with one another using each of the nine features as a basis. Likewise, $P(D_n(S_1, S_2) = d)$ is generated by comparing all of the "similar" symbols with one another in the same manner. "Similar" symbols are differing symbols that are likely to be confused with one another, such as the same character in two different fonts or styles. Figure 7-1 contains the graphs of $P(D_n(M_1, M_2) = d)$ and $P(D_n(S_1, S_2) = d)$, determined from data compiled from these tests.

# (n)	Feature Name	r_n	e_n
1	Width	2	0.72
2	Height	4	0.50
3	Number of Black Pixels	30	0.66
4	Number of White Pixels	53	0.52
5	Number of Horizontal Run-Lengths	8	0.37
6	Number of Vertical Run-Lengths	4	0.62
7	Horizontal Moment	2	0.49
8	Vertical Moment	3	0.32
9	Average Width	3	0.37

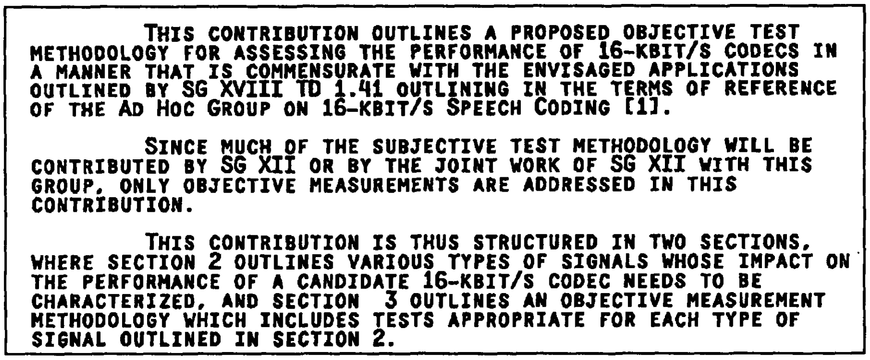
Table 7.2: Feature statistics – rejection threshold and effectiveness.

Using these results, the procedure described above was performed to determine the optimal rejection threshold and the effectiveness of each feature. As can be seen in Table 7.2, the five most effective features of a symbol (in order) are its width, the number of black pixels, the number of vertical runs, the number of white pixels, and its height, with an overall effectiveness of 0.991. These are the features that were selected for CAFC.

7.3 Template Matching Criteria

The template matching procedure, described in Sect. 4.1.2, also uses a rejection threshold to determine whether or not two symbols are matches. Unlike in feature matching, however, it is extremely important that the template matcher does not falsely match any differing symbols. This is because template matching is the final stage in symbol matching, and any errors made here would result in the incorrect symbol being substituted into the reconstructed image.

Figure 7-2 shows a sample portion of a page processed with the rejection threshold r_t set at three different values. When r_t is on the low side at 0.6, the results are somewhat embarrassing; the symbol mismatches are numerous and obvious, and almost appear as typos. When r_t is raised to 0.8, only a few errors appear. Finally when r_t is close to 1, there are no errors.

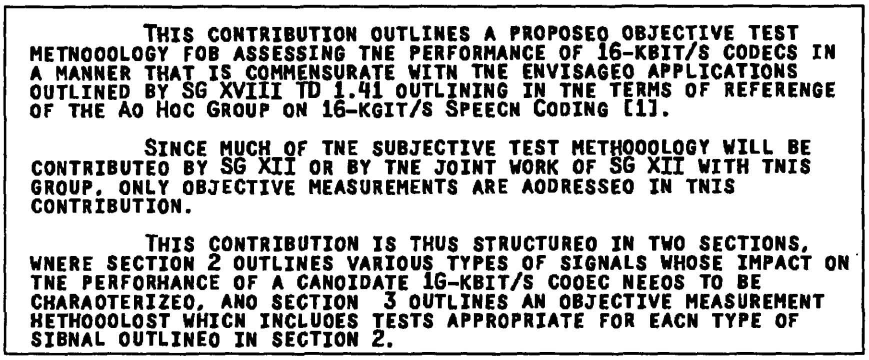


THIS CONTRIBUTION OUTLINES A PROPOSED OBJECTIVE TEST METHODOLOGY FOR ASSESSING THE PERFORMANCE OF 16-KBIT/S CODECS IN A MANNER THAT IS COMMENSURATE WITH THE ENVISAGED APPLICATIONS OUTLINED BY SG XVIII TD 1.41 OUTLINING IN THE TERMS OF REFERENCE OF THE AD HOC GROUP ON 16-KBIT/S SPEECH CODING [1].

SINCE MUCH OF THE SUBJECTIVE TEST METHODOLOGY WILL BE CONTRIBUTED BY SG XII OR BY THE JOINT WORK OF SG XII WITH THIS GROUP, ONLY OBJECTIVE MEASUREMENTS ARE ADDRESSED IN THIS CONTRIBUTION.

THIS CONTRIBUTION IS THUS STRUCTURED IN TWO SECTIONS, WHERE SECTION 2 OUTLINES VARIOUS TYPES OF SIGNALS WHOSE IMPACT ON THE PERFORMANCE OF A CANDIDATE 16-KBIT/S CODEC NEEDS TO BE CHARACTERIZED, AND SECTION 3 OUTLINES AN OBJECTIVE MEASUREMENT METHODOLOGY WHICH INCLUDES TESTS APPROPRIATE FOR EACH TYPE OF SIGNAL OUTLINED IN SECTION 2.

original

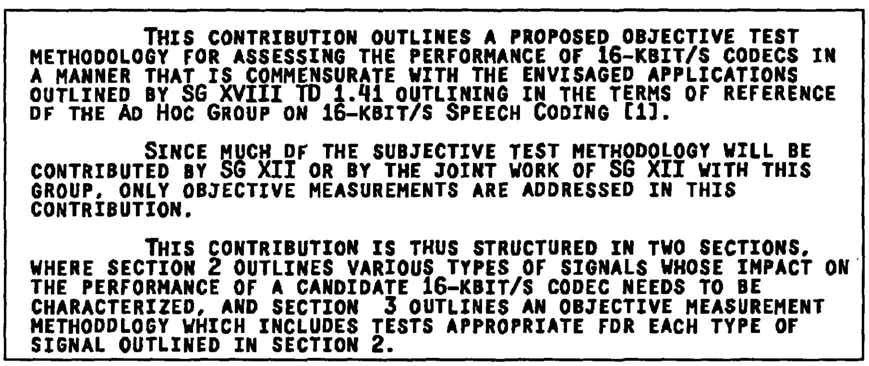


THIS CONTRIBUTION OUTLINES A PROPOSEO OBJECTIVE TEST METNOOOLOGY FOB ASSESSING TNE PERFORMANCE OF 16-KBIT/S CODECS IN A MANNER THAT IS COMMENSURATE WITN TNE ENVISAGEO APPLICATIONS OUTLINED BY SG XVIII TD 1.41 OUTLINING IN TNE TERMS OF REFERENGE OF THE AO HOC GROUP ON 16-KGIT/S SPEECN CODING [1].

SINCE MUCH OF TNE SUBJECTIVE TEST METHOOOLOGY WILL BE CONTRIBUTEO BY SG XII OR BY TNE JOINT WORK OF SG XII WITH TNIS GROUP, ONLY OBJECTIVE MEASUREMENTS ARE AODRESSEO IN TNIS CONTRIBUTION.

THIS CONTRIBUTION IS THUS STRUCTUREO IN TWO SECTIONS, WNERE SECTION 2 OUTLINES VARIOUS TYPES OF SIGNALS WHOSE IMPACT ON TNE PERFORHANCE OF A CANOIDATE 1G-KBIT/S COOEC NEEOS TO BE CHARAOTERIZEO, ANO SECTION 3 OUTLINES AN OBJECTIVE MEASUREMENT HETHOOOLOST WHICN INCLUOES TESTS APPROPRIATE FOR EACN TYPE OF SIBNAL OUTLINEO IN SECTION 2.

correlation rejection threshold=0.6

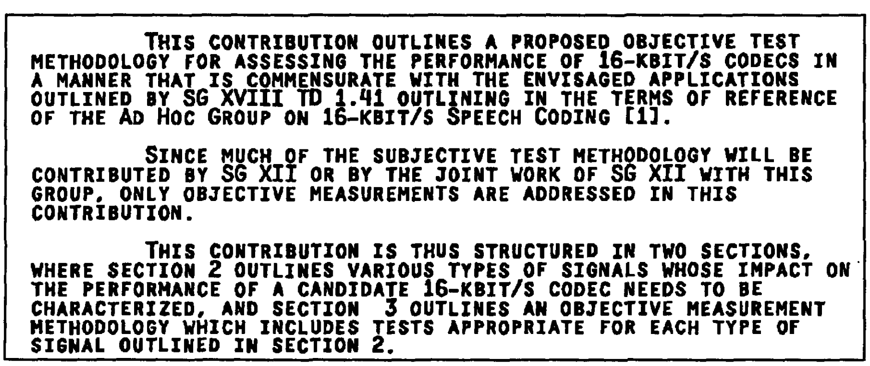


THIS CONTRIBUTION OUTLINES A PROPOSED OBJECTIVE TEST METHODOLOGY FOR ASSESSING THE PERFORMANCE OF 16-KBIT/S CODECS IN A MANNER THAT IS COMMENSURATE WITH THE ENVISAGED APPLICATIONS OUTLINED BY SG XVIII TD 1.41 OUTLINING IN THE TERMS OF REFERENCE DF THE AD HOC GROUP ON 16-KBIT/S SPEECH CODING [1].

SINCE MUCH DF THE SUBJECTIVE TEST METHODOLOGY WILL BE CONTRIBUTED BY SG XII OR BY THE JOINT WORK OF SG XII WITH THIS GROUP, ONLY OBJECTIVE MEASUREMENTS ARE ADDRESSED IN THIS CONTRIBUTION,

THIS CONTRIBUTION IS THUS STRUCTURED IN TWO SECTIONS, WHERE SECTION 2 OUTLINES VARIOUS TYPES OF SIGNALS WHOSE IMPACT ON THE PERFORMANCE OF A CANDIDATE 16-KBIT/S CODEC NEEDS TO BE CHARACTERIZED, AND SECTION 3 OUTLINES AN OBJECTIVE MEASUREMENT METHODDLOGY WHICH INCLUDES TESTS APPROPRIATE FDR EACH TYPE OF SIGNAL OUTLINED IN SECTION 2.

correlation rejection threshold=0.8



THIS CONTRIBUTION OUTLINES A PROPOSED OBJECTIVE TEST METHODOLOGY FOR ASSESSING THE PERFORMANCE OF 16-KBIT/S CODECS IN A MANNER THAT IS COMMENSURATE WITH THE ENVISAGED APPLICATIONS OUTLINED BY SG XVIII TD 1.41 OUTLINING IN THE TERMS OF REFERENCE OF THE AD HOC GROUP ON 16-KBIT/S SPEECH CODING [1].

SINCE MUCH OF THE SUBJECTIVE TEST METHODOLOGY WILL BE CONTRIBUTED BY SG XII OR BY THE JOINT WORK OF SG XII WITH THIS GROUP, ONLY OBJECTIVE MEASUREMENTS ARE ADDRESSED IN THIS CONTRIBUTION.

THIS CONTRIBUTION IS THUS STRUCTURED IN TWO SECTIONS, WHERE SECTION 2 OUTLINES VARIOUS TYPES OF SIGNALS WHOSE IMPACT ON THE PERFORMANCE OF A CANDIDATE 16-KBIT/S CODEC NEEDS TO BE CHARACTERIZED, AND SECTION 3 OUTLINES AN OBJECTIVE MEASUREMENT METHODOLOGY WHICH INCLUDES TESTS APPROPRIATE FOR EACH TYPE OF SIGNAL OUTLINED IN SECTION 2.

correlation rejection threshold=0.9

Figure 7-2: Reconstructed images at various template matching thresholds.

r_t	total symbols	unique symbols	size(bytes)
0.6	689	84	4028
0.8	689	216	7215
0.9	689	468	11740

Table 7.3: CAFC coding performance at various template matching thresholds

The value of r_t also has a significant effect on the performance of Symbol Matching and Substitution and on the overall compression performance of CAFC. Table 7.3 compares the total number of unique symbols detected and total number of encoded bytes for the above image at the different threshold values. The simulation shows that as r_t is increased, the performance drops rapidly. It is therefore important to choose a value for r_t that is just high enough to eliminate any symbol matching errors, but no higher. Through extensive trial-and-error on a number of different source images, a value of 0.82 was chosen.

7.4 2D Run-Length Coding Initial Model

In order for 2D Optimized Run-Length Coding to be effective, it is important the that arithmetic coder take full advantage of the disparity in the relative probabilities of each of the run-lengths through entropy-coding. The arithmetic coder used by CAFC is adaptive and automatically does this by developing a model containing run-length statistics on the fly. However, at the top of the page, the arithmetic coder has only collected a very small amount of data and does not have such an accurate model available.

To compensate for this, the CAFC arithmetic encoder and decoder start off with models containing run-length statistics that are intended to be representative of the majority of facsimile documents. These are generated by encoding the training set images in Appendix D with 2D Optimized Run-Length Coding. Once collected, the probabilities are scaled down by a factor of 4 and placed in the initial arithmetic coding model for CAFC. With a scale factor of 4, the initial run-length statistics are used exclusively at the top of the page and the adaptively-determined statistics begin to dominate only after half of the page has been processed, at which point a reasonably good model will have developed.

Chapter 8

Analysis and Evaluation

This chapter discusses the overall performance of Content-Adaptive Facsimile Coding as determined by a number of different measures. Through the use of the software implementation of CAFC described and listed in Appendix E, a large number of simulations were performed to quantify several important properties of CAFC. Unfortunately, the detection and coding of dithered bitmaps is missing from this implementation and could therefore not be evaluated. However, none of the test images used in these tests appear to contain any instances of this content anyway.

Of primary interest is the amount of compression that is achieved by the algorithm. Without a reasonably high enough compression ratio, it would be difficult to justify the introduction of the added complexity of CAFC into a facsimile communication system. Equally important is the quality of the reconstructed images that are generated by the decoder. To analyze the performance of CAFC for these measures, a set of test documents is passed through the encoder and decoder stages so that both the compression ratios and image quality can be evaluated.

The remaining two properties that are examined here directly affect CAFC's capability of being incorporated onto a real-world hardware platform. The requirements of CAFC in terms of computational resources are a direct measure of the cost of implementing it in hardware. In order for CAFC to be economically feasible, this cost must be low compared to the savings obtained in channel bandwidth. Finally, the amount of coding delay introduced by CAFC must not be too high or it would be technically impossible to use with the existing facsimile protocols.

8.1 Compression Gains

The overall compression gain for CAFC was measured by encoding the set of eight standard CCITT documents included in Appendix A. Table 8.1 lists the size of each of the CAFC-compressed images in bytes as well as those reported from a number of existing bi-level image compression algorithms, including Group 3, two-dimensional Group 3, Group 4, and JBIG. The compression ratios achieved are shown in Table 8.2, and a direct comparison is made with CCITT one-dimensional Group 3.

The compression ratios for CAFC varied significantly, from roughly 6:1 to 27:1 depending upon the document. As expected, it outperformed Group 3 in every case by an average of almost 2:1. Compared to the remaining compression algorithms, however, CAFC did not fare so well. With the exception of CCITT Image #4, which consists predominantly of typed text, CAFC performed about as well as G3D2, slightly worse than Group 4, and about a factor of 2 worse than JBIG. However, this is to be expected, because the majority of compression gains in CAFC are obtained from Symbol Matching and Substitution, optimized for typed text. On CCITT Image #4, for example, CAFC outperformed the JBIG standard by over 25%. The Optimized 2D Run-Length Coding in CAFC is not nearly as sophisticated as the two-dimensional approaches in Group 4 or JBIG, so it is no surprise

Source Image	Raw	G3D1	G3D2	G4	JBIG	CAFC
CCITT #1	513216	37423	25967	18103	14715	18816
CCITT #2	513216	34367	19656	10803	8545	20980
CCITT #3	513216	65034	40797	28706	21988	38194
CCITT #4	513216	108075	81815	69275	54356	39862
CCITT #5	513216	68317	44157	32222	25877	36903
CCITT #6	513216	51171	28245	16651	12589	27494
CCITT #7	513216	106420	81465	69282	56253	83604
CCITT #8	513216	62806	33025	19114	14278	34640

Table 8.1: Compressed file sizes in bytes for various coding algorithms.

Source Image	CAFC:raw Compression Ratio	CAFC:G3D1 Compression Ratio
CCITT #1	27.3:1	2.0:1
CCITT #2	24.5:1	1.6:1
CCITT #3	13.4:1	1.7:1
CCITT #4	12.9:1	2.7:1
CCITT #5	13.9:1	1.9:1
CCITT #6	18.7:1	1.9:1
CCITT #7	6.13:1	1.3:1
CCITT #8	14.8:1	1.8:1

Table 8.2: Relative compression ratios for CAFC

that CAFC cannot compete on documents where this form of coding is predominantly used.

However, the results of this analysis lead to a promising conclusion. It appears that if the Optimized 2D Run-Length Coder were to be replaced by an approach similar to JBIG, CAFC could perform as well as JBIG for non-text documents and better than JBIG for typed documents. Consider the estimates shown in Table 8.3. The eight CCITT documents are processed with JBIG, CAFC, and CAFC Optimized 2D Run-Length Coding. In addition, the residues generated from CAFC are processed with Optimized 2D Run-Length Coding. The compressed file sizes are used to determine how much of each image is encoded using Symbol Matching and Substitution and how much was encoded using Optimized 2D Run-Length Coding. Then, based upon the JBIG:CAFC(RL coding only) ratio for each image, the number of bytes that would be necessary to encode the residue with JBIG is estimated. This value is added to the bytes required for Symbol Matching and Substitution to obtain the estimated compressed file size for the revised version of CAFC.

Source Image	JBIG	CAFC	CAFC (RL only)	CAFC (RL on residue)	Potential CAFC & CAFC:G3D1 ratio
CCITT #1	14715	18816	27651	16906	10907 (3.4:1)
CCITT #2	8545	20980	20995	20861	8609 (4.0:1)
CCITT #3	21988	38194	42150	36754	20613 (3.2:1)
CCITT #4	54356	39862	95189	30484	25850 (4.2:1)
CCITT #5	25877	36903	47887	33993	21279 (3.2:1)
CCITT #6	12589	27494	28273	27069	12478 (4.1:1)
CCITT #7	56253	83604	96038	75910	52157 (2.0:1)
CCITT #8	14278	34640	34489	34209	14593 (4.3:1)

Table 8.3: Estimated file sizes for CAFC with suggested modification.

The preliminary calculations suggest that the modified CAFC could either match or beat the performance of JBIG for almost all documents. This same approach would beat Group 3 by an average of approximately 3.5:1.

8.2 Reconstructed Image Quality

Both the Optimized 2D Run-Length Coder and Direct Coder components of CAFC are lossless techniques and do not introduce any distortion into the document. Image degradation is only possible with Symbol Matching and Substitution. It is therefore appropriate to examine reconstructed images containing typed text when evaluating this property of CAFC.

There are basically two types of distortion that can be introduced into an image by Symbol Matching and Substitution. The first is the error that is associated with the false matching two symbols that are actually different. The observable consequences of such a mistake are the appearance of the wrong character on the page. The second type of distortion results from errors in the placement of symbols on the reconstructed output image.

Appendix B contains the eight CCITT documents after having been encoded and decoded with CAFC. Occurrences of the first type of degradation are extremely rare, and most often occur when the font size is very small and the two mismatched symbols are scanned in poorly. In such cases, it is difficult for even a human set of eyes to differentiate the symbols, and such errors are likely to be overlooked. It is also possible, but difficult, to observe slight variations in the placement of symbols. Careful inspection of the lines of text reveals that they sometimes "weave" up and down by one or two pixels. This probably occurs because of the slight differences in the width and height of symbols that have been scanned in from different portions of the page. Suggestions are made in Sect. 10.2 to correct this problem.

Overall, the quality of the reconstructed images is excellent and is believed to be acceptable for use in commercial systems.

8.3 Computational Resources

Compared to Group 3, CAFC is a fairly sophisticated compression algorithm. While Group 3 can be easily coded on a very inexpensive microcontroller without concerns of memory or speed limitations, CAFC has relatively more demanding hardware requirements.

The most compute-intensive component of the CAFC algorithm is the search through the symbol library for matching symbols, and in particular template matching. These operations grow linearly with the size of the library. Thus, images with the largest number of unique symbols require the most processing time, and images with few unique symbols require the least. The majority of the memory required by CAFC is needed to store the bitmap representations of the symbols in the library. Again, documents with many unique symbols impose the greatest demand.

All simulations were performed on a Hewlett Packard 9000/720 (57 MIPS, 32 Mbyte of RAM) workstation using the C code contained in Appendix E. Compressing the majority of documents took approximately 30 seconds, while compressing documents with many unique symbols (such as CCITT #7) required about 90 seconds. Decoding the images took approximately half as long. It is believed that a substantial portion of the execution time is spent reading and writing the image files off of the disk.

The times required to process the images on the workstation are certainly not unreasonable; they are approximately as long as it takes presently to transmit a page over Group 3 facsimile terminal equipment. And if the code were rewritten in assembly language on a high-speed digital signal processor (DSP), the execution times would drop substantially. As the cost of DSPs and memory chips continues to drop, the feasibility of implementing CAFC as a hardware add-on to DCME increases. In fact, many of the speech coders presently used in DCME are of comparable complexity to DCME. It is believed that by 1997, when the next generation of DCME's are phased in, high-performance facsimile compression will offer a significant cost advantage.

8.4 Coding Delay

Finally, it is important to consider the amount of delay that is introduced by a coding algorithm. For real-time systems such as DCME, it is essential that it be kept to a minimum (see Sect. 9.2).

Group 3 coding has a delay of a single scan-line, since it is one-dimensional and only processes one line at a time. CAFC, on the other hand, needs several scan-lines in order to perform Symbol Matching and Substitution. Fortunately, an upper bound on the coding delay is inherent to CAFC because of the restriction on the height of a detected symbol. For CAFC to be able perform encoding and decoding by processing a fixed number of scan-lines at a time, the symbol isolator must be able to detect a symbol contained within this buffer. This buffer must be at least one pixel taller than the maximum allowed symbol height.

The maximum symbol height is a completely adjustable parameter. The larger it is, the more symbols can be isolated, giving CAFC the potential of achieving higher compression ratios for typed text. The cost is a higher coding delay. However, since the majority of text is not very large (probably no more than 14pt), there is a point of diminishing returns where a further increase in the maximum symbol height does not buy much additional compression. This seems to be in the neighborhood of 20 pixels.

Thus, although the coding delay introduced by CAFC is significantly higher than that of Group 3, it is bounded and can be adjusted to meet the needs of the system on which it is to be implemented.

Chapter 9

DCME Implementation Issues

In order to use Content-Adaptive Facsimile Coding as a secondary compression stage for facsimile communications over Digital Circuit Multiplication Equipment (DCME) [3], a number of implementation details must first be worked out.

9.1 Variable Bandwidth Output

One potential difficulty with CAFC is that its compression ratio is not fixed, but can vary significantly depending upon the nature of the source document. In fact, it can even be expected to change drastically throughout the transmission of a single page. This is because the Symbol Matching and Substitution encoder has to build up its library before it can achieve any compression, which cannot occur until a number of symbols are encoded with the less efficient Optimized 2D Run-Length coder. The situation appears even worse when one considers that CAFC is used as secondary compression. That is, the facsimile input channels to DCME are CCITT Recommendation T.4 Group 3-encoded images that must first be uncompressed before CAFC is applied. Group 3 is in general less efficient than CAFC, but the actual disparity between the two techniques varies considerably over time, especially when the source image contains a lot of typed text. So while the external facsimile terminal equipment transmits modulated Group 3-encoded data at a fixed rate, the two communicating DCMEs must exchange CAFC-compressed baseband data at an unpredictable rate.

Fortunately, a DCME configuration is an ideal environment for overcoming these sorts of problems. Because it multiplexes hundreds of channels together into a single high-speed link, it can allow for the bandwidths of each channel to vary over time, as long as the total bandwidth remains below the absolute maximum. Under typical conditions, a large number of facsimile pages are transmitted simultaneously, each sending a different portion of the page at a given time. The mechanisms in DCME for allowing variable bandwidth channels are not very straightforward and require a fairly sophisticated controller. However, unlike most other communications systems, DCME does possess this feature.

Of course, it is impossible to guarantee that the total bandwidth required by all of the DCME channels will always fall below the capacity of the high-speed link without placing severe restrictions on the total number of channels. Occasionally the channels are heavily loaded, and there is simply too much data to transmit in too short of a time span. Conventional DCME systems get around this problem on speech channels by using special coders that can compress the speech by an additional amount when necessary by sacrificing speech quality. When the system gets overloaded, a controller selects one or more voice channels to temporarily produce fewer output bits by increasing the compression, alleviating the problem. The associated increase in distortion is hardly noticeable because these periods of simultaneous high channel activity are short and infrequent.

The ability to make a tradeoff between quality and compression gain is thus a valuable feature to have in a source coder used in DCME. In most equipment today, however, facsimile and data channels are not equipped with this capability. Instead, they are simply given priority over voice channels so that only speech signals are allowed to be corrupted during overload. In the case of data channels, this is necessary to ensure an error-free transmission. But for facsimile, it is done only because there is no simple way to reduce the number of bits in a Group 3-encoded document without causing significant distortion to the page. If a facsimile compression algorithm with a selectable compression threshold could be developed, the same technique could be applied. CAFC does not provide this feature, nor does it lend itself to an easy modification so that it can. However, a number of lossy decimation and interpolation techniques have been developed which reduce the amount of redundancy in an image so that additional compression may be achieved [2] [15]. It is possible that the facsimile page can be preprocessed with such an algorithm before it is encoded with CAFC to reduce the overall bandwidth when the overload condition occurs.

9.2 Coding Delay

Another important factor in the implementation of a facsimile source coding technique is the type of system into which it will be incorporated. In a real-time facsimile communications system, both the transmitting and receiving facsimile terminal equipment are on-line and in direct communication with each other throughout the duration of the transfer; the transmitter does not disconnect until the entire document has been received. This differs from store-and-forward systems, where the document is first obtained from the transmitter, temporarily stored, transferred at a convenient time, and finally sent to the receiver. Because neither of the two facsimile terminals is tied up when the document is sent over the main communications link, store-and-forward systems can tolerate an arbitrary amount of propagation delay. Secondary facsimile compression can be easily incorporated into these systems because all processing can be performed while the terminals are off-line. In fact, since the entire page is available in storage, it is possible to use highly sophisticated compression algorithms that utilize all of the image information. Even processing time is not a major concern, because the transmission is already delayed by a period of time that is much longer than it takes to process the document. In contrast, real-time systems cannot withstand large delays between the transmitter and receiver because the CCITT Recommendation T.30 facsimile protocols do not account for them. At the beginning and end of the transfer, when two-way handshaking is performed, it is possible for some of the timeout thresholds to be exceeded, resulting in synchronization problems.

Despite these difficulties, DCME facsimile channels are always real-time systems. There are a number of specific reasons for this which are beyond the scope of this thesis. However, secondary facsimile compression can still be incorporated into real-time systems as long as certain restrictions are placed on the nature of the algorithm. First, the facsimile images must be transmitted serially from top to bottom as it is with Group 3. This requires both the encoder and decoder to be causal systems. Naturally, when generating output, they can only make use of information from the portion of the page which has been received so far. Second, the delays inherently introduced from the coding and decoding algorithms should be minimized to prevent timeouts in the protocol. Finally, the compression and decompression procedures should not demand too much processing power; this could introduce further delay into the system or make it too expensive or infeasible to implement.

Content-Adaptive Facsimile Coding is designed to meet all of the above restrictions. The serial input is processed from top to bottom, and the multiplexing stage ensures that the individual image components remain in this order in the encoded output. Of the three content coders, only Symbol Matching and Substitution has the potential for introducing significant delay into the system. This occurs only when large symbols are encoded, since many scan lines from the input have to be analyzed before a match can be detected. However, an upper bound is placed on the delay by imposing a limit on the height of a symbol that can be detected by the isolator. This restriction does degrade the compression ratio somewhat because fewer symbols can be detected and encoded with Symbol Matching and Substitution. Such a tradeoff between delay and loss of compression ratio is a property common to all source coding techniques, and a judicious choice must be made when establishing the thresholds so that the desired performance is obtained. Finally, the most compute intensive stages of CAFC are the symbol isolation and feature/template matching, and they should not present too much of a challenge for tomorrow's hardware.

9.3 Forward Error Correction

To prevent severe image distortion due to bit errors, Forward Error Correction is applied to all DCME facsimile channels. The use of FEC drastically reduces the bit error rate (BER) of a channel, allowing facsimile messages to be reliably transmitted over DCME. It does so by intentionally introducing redundancy into the data stream so that the receiver can detect and correct most errors. Despite its effectiveness, it is theoretically impossible for FEC to eliminate all bit errors in a channel; at best, there will be an occasional corrupted bit in the message. When this occurs, distortion is introduced into the received facsimile document. The effect of such an error on the reconstructed page is highly dependent upon the properties of the source coding scheme that is used. It is usually the case that image coders that achieve high compression gains are less tolerant to bit errors than those that do not. As it turns out, CAFC is extremely sensitive to corrupted data and completely breaks down when even a single bit error is introduced. This is because the arithmetic decoder that is used to demultiplex the different contents can no longer correctly decode the remainder of the message. From the point in time when an error occurs until the end of the transmission, the reconstructed image is severely distorted in an unpredictable manner.

Group 3 coding, which uses Huffman coding rather than arithmetic coding, suffers from this difficulty as well. However, the problem is mitigated through the use of special synchronization codes inserted into the data stream at the end of each scan line. When an error occurs, the entire scan line is corrupted, but the decoder can at least "resync" at the end of the line and continue decoding normally beginning with the next scan line. Usually, when a single scan line is omitted from an image, it is difficult or impossible to notice anyway. This approach works very well, and methods are being investigated to apply similar techniques to arithmetic coding. Unfortunately, even with the successful incorporation of synchronization codes into CAFC, the coding scheme still suffers from a high bit-error sensitivity. The Symbol Matching and Substitution content coder relies heavily on the equivalence of the symbol libraries at both ends of the transmission. When a scan line becomes corrupted, it is possible that the CAFC decoder might not correctly detect a symbol and update its library. From that point on, all symbols on the remainder of the page are incorrectly decoded, producing a significant amount of distortion. Even if the library remains intact, a corrupted scan line could result in the absence of a library index number and therefore a missing symbol.

One possible solution to this problem is to insert library synchronization information into the data stream to help prevent the occurrence of dangerous inconsistencies between the libraries. Although some symbols would still be corrupted, the majority would remain intact. Another idea is to selectively use an additional degree of forward error correction on the most critical portions of the page. This would include areas with new symbols and areas with a lot of repeated symbols. The vast majority of errors that occur in such regions would be corrected, decreasing the incidence of image distortion. Both of these possibilities need to be further investigated. Of course, if all else fails, it is always possible to employ a high degree of forward error correction to the entire transmission, lowering the effective bit-error-rate to some negligible amount. Increased reliability would be obtained at the expense of additional channel bandwidth.

Chapter 10

Conclusion and Recommendations

10.1 Summary and Conclusion

This thesis describes the conception, development, and optimization of a novel approach to bi-level image (facsimile) compression. The objective was to develop a page model that is more sophisticated than those used in existing compression algorithms. The idea was to separate the page into its different contents, encode them separately using the coding technique best-suited for the properties of each content, and then multiplex the compressed data into a single output data stream.

A model was selected, consisting of three classes of contents: typed text, handwriting and graphics, and dithered bitmaps. Three different coding techniques were developed to encode them: Symbol Matching and Substitution, Optimized 2D Run-Length Coding, and Direct Coding. Particular emphasis was placed on optimizing the performance of Symbol Matching and Substitution because it appeared to have the greatest potential for compression gains. Arithmetic coding was selected as the mechanism for both multiplexing the three streams and performing entropy-coding.

Procedures were developed to optimize the various components of the algorithm, and then extensive simulations were performed. Preliminary results show that CAFC outperforms CCITT Recommendation T.4 Group 3 Run-Length Coding by roughly a factor of 2:1 for most documents and almost 3:1 for typed text. With some modifications, it is believed that it could do as well as or better than JBIG for all documents. Although the Symbol Matching and Substitution component of the algorithm is lossy, the distortion that is introduced is difficult or impossible to perceive.

Finally, the initial target application, Digital Channel Multiplication Equipment, was explained and the implementation issues were discussed. CAFC has the potential to be used in such equipment to effectively double the number of facsimile channels that can be active simultaneously without any increase in the bandwidth of the high-speed channel. The cost of such a system would be modest compared to many of the components in existing DCME systems.

10.2 Improvements to Algorithm

The results of the simulations in Chap. 8 indicate that CAFC has the potential for an even higher degree of compression and image quality. Based upon these observations, the following suggestions are made for future work that could lead to significant improvements:

- Set of Contents/Coding Techniques: The page model described in Chap. 3 divides the page into typed text, handwriting, graphics, and dithered bitmaps. While this may seem like a logical classification of contents, it is certainly possible that the page could be decomposed into a different set of contents that lends itself to a more efficient set of coding schemes. An objective for future work would be to refine the CAFC model so that the set of contents better represents the page and each content is most efficiently coded while still maintaining a high degree of reliability and practicality.
- Symbol Matching: An area that should definitely be targeted for improvement is the symbol matching process, particularly template matching. In order to calibrate the Symbol Matching and Substitution encoder so that it would not incorrectly match differing symbols, it was necessary to set the rejection threshold r_t fairly high. It was shown in Sect. 7.3 that even a slight decrease in this parameter would yield a significant increase in coding performance. Based upon the number of matching symbols that are falsely rejected by the template matcher, it is believed that a much

more robust algorithm could replace it. One possible approach would be to expand feature matching to use a much larger number of features and to use a multidimensional decision region that takes into account the statistical correlations between the features.

- Residue Coding: The focus of this research was on the development of a high performance Symbol Matching and Substitution algorithm for the efficient coding of typed text. Because of this, the techniques that were developed to encode the residue, Optimized 2D Run-Length Coding and Direct Coding, cannot compete with some of the more sophisticated lossless standards such as JBIG, as was shown in Sect. 8.1. Optimized Run-Length Coding could be replaced by any number of superior two-dimensional coding techniques, both lossless [7] [8] [9] and lossy [2] [15]. Direct Coding, which does not perform any compression at all, could be replaced with a technique designed specifically for dithered images [10] (even JBIG has provisions for this). By using these approaches to encode graphics, handwriting, and dithered bitmaps, and using Symbol Matching and Substitution for typed text, a very high degree of compression would likely be obtained.
- Arithmetic Coding Models: The three arithmetic coding models that are used in CAFC, described in Sect. 5.4, were designed based upon a number of assumptions and intuitions about the contents of facsimile documents. A more systematic approach would be to analyze a large number of training set images and determine the relative probabilities of runs, new symbols, matched symbols, and bitmap fragments and the orders in which they occur. Perhaps an improved model could be developed using this information that could further reduce the number of output bits generated by the arithmetic encoder.
- Symbol Placement: It was pointed out in Sect. 8.2 that the placement of symbols in the reconstructed images is sometimes slightly off-center, resulting in lines of typed text that tend to "weave" up and down by a small amount. A proposed solution to this problem is to compute the horizontal and vertical moments of each symbol and to then align each matching symbol about this point in the reconstructed image. This would require some additional bookkeeping by the encoder, but would have no effect on the compression performance of the encoder.

With the above suggestions, as well as modifications targeted for the specific application (such as those mentioned in Chap. 9 for DCME), Content-Adaptive Facsimile Coding has the potential to be a highly reliable real-time compression system that would provide substantial cost advantage for facsimile service providers.

Bibliography

[1] CCITT Recommendation T.4, "Standardization of Group 3 facsimile apparatus for document transmissions," 1988.

[2] R. Ragland, N. Tender, and S. Dimolitsas, "Facsimile compression for Inmarsat-M," tech. rep., COMSAT Laboratories, December 1992.

[3] INTELSAT Earth Station Standard (IESS-501), "Digital circuit multiplication equipment specification," December 1992.

[4] W. K. Pratt, P. J. Capitant, W. Chen, E. Hamilton, and R. Wallis, "Combined symbol matching facsimile data compression system," *Proceedings of the IEEE*, vol. 68, pp. 786–796, July 1980.

[5] O. Johnsen, J. Segen, and G. L. Cash, "Coding of two-level pictures by pattern matching and substitution," *The Bell System Technical Journal*, pp. 2513–2545, October 1983.

[6] D. M. Silver and D. A. H. Johnson, "Facsimile coding using symbol-matching techniques," *IEEE Proceedings*, vol. 131, pp. 125–129, April 1984.

[7] F. W. Mounts, E. Bowen, and A. N. Netravali, "An ordering scheme for facsimile coding," *The Bell System Technical Journal*, vol. 58, pp. 2113–2129, November 1979.

[8] J. W. Woods, "Two-dimensional delta-mod facsimile coding," *IEEE Transactions on Communications*, vol. COM-26, pp. 936–939, June 1978.

[9] J. L. Mitchell and G. Goertzel, "Two-dimensional facsimile coding scheme," tech. rep., IBM Thomas J. Watson Research Center, 1979.

[10] T. Usubuchi, T. Omachi, and K. Iinuma, "Adaptive predictive coding for newspaper facsimile," *Proceedings of the IEEE*, vol. 68, pp. 807–813, July 1980.

[11] R. Aravind, G. Cash, D. Duttweiler, H. Hang, B. Haskell, and A. Puri, "Image and video coding standards," *AT&T Technical Journal*, pp. 67–72, January/February 1993.

[12] F. Tzeng, "Content-based super-compressive facsimile coding," tech. rep., COMSAT Laboratories, September 1990.

[13] M. Nelson, *The Data Compression Book.* San Mateo, CA: M&T Publishing, 1992.

[14] I. Witten, R. Neal, and J. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, pp. 520–540, June 1987.

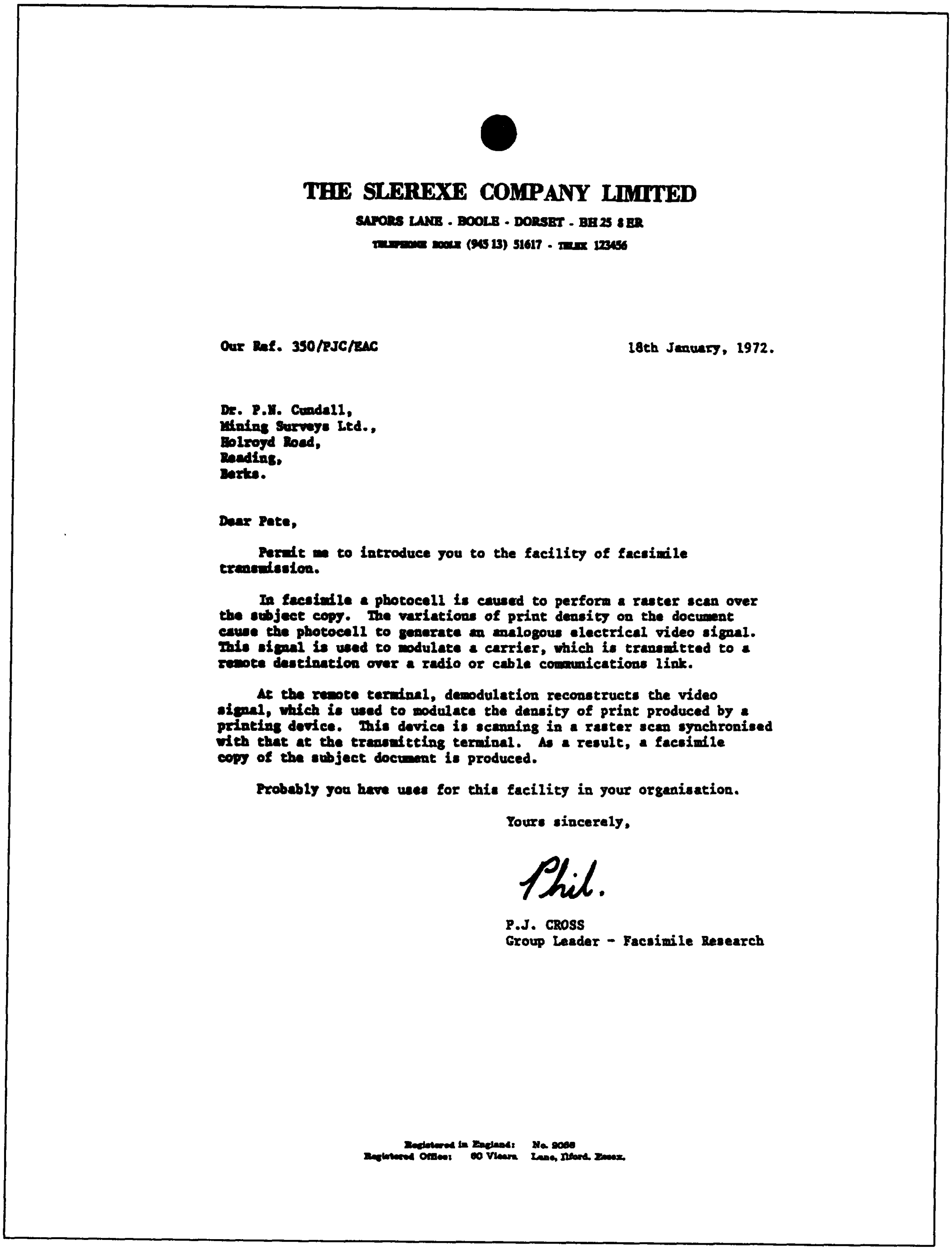
[15] M. Ismail and R. Clarke, "Adaptive block/location coding of facsimile signals using subsampling and interpolation for pre- and postprocessing," *IEEE Transactions on Communications*, vol. COM-29, pp. 1925–1933, December 1981.

Appendix A

CCITT Test Images

The following eight facsimile images are the standard set of CCITT test images. They are intended to be representative sample the types of documents that are transmitted by facsimile. They are useful for comparing the performance of different facsimile terminal equipment and coding techniques.

Each page was individually scanned into a facsimile machine, transmitted to a PC-based fax card, and then saved to a file on the host computer system. All of the images are in fine mode (200 pixels/inch) but are reduced by 30% along each axis on the following pages.



THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH 25 8 ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

Registered in England: No. 2038
Registered Office: 60 Vicars Lane, Ilford. Essex.

Figure A-1: CCITT test image #1

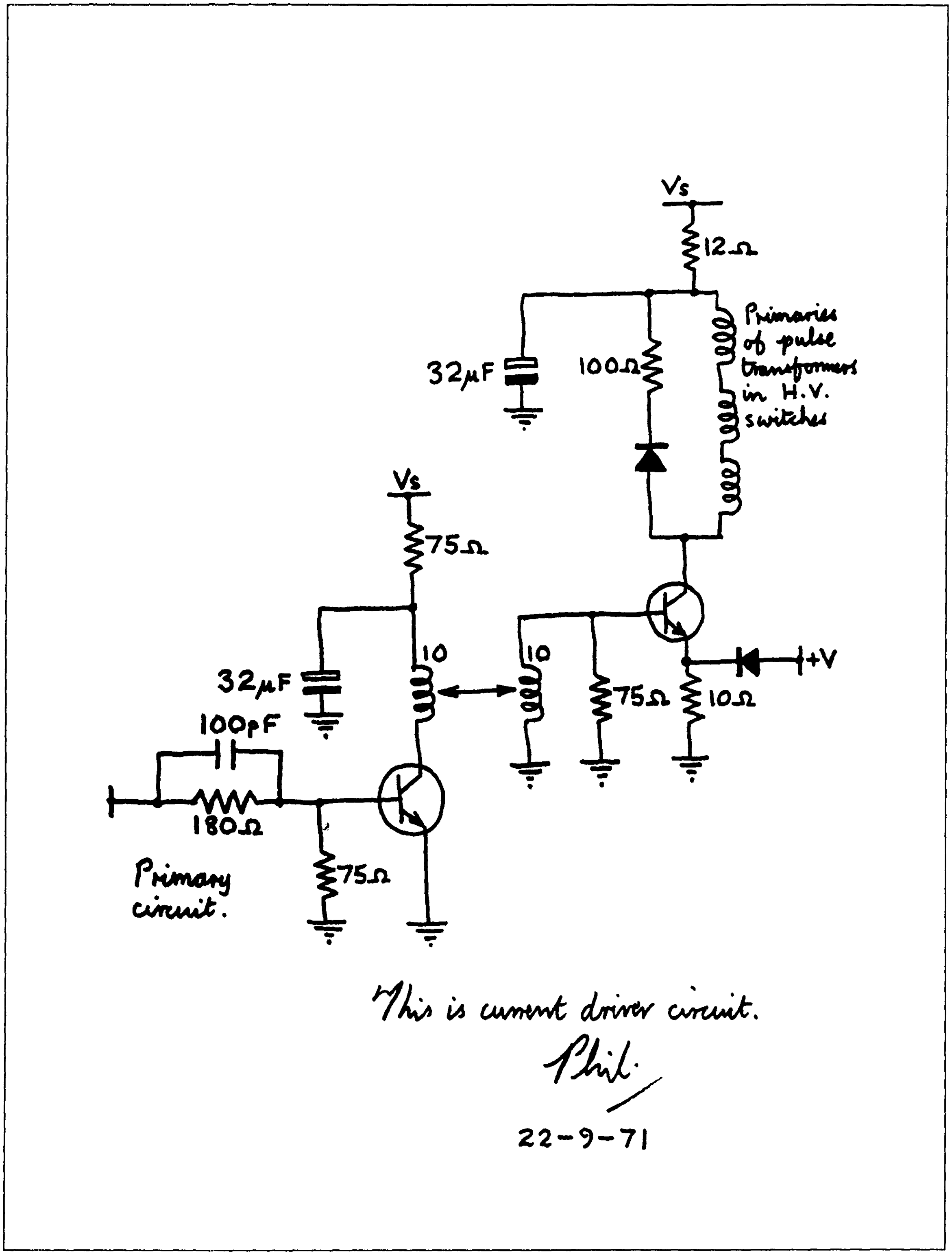


Figure A-2: CCITT test image #2

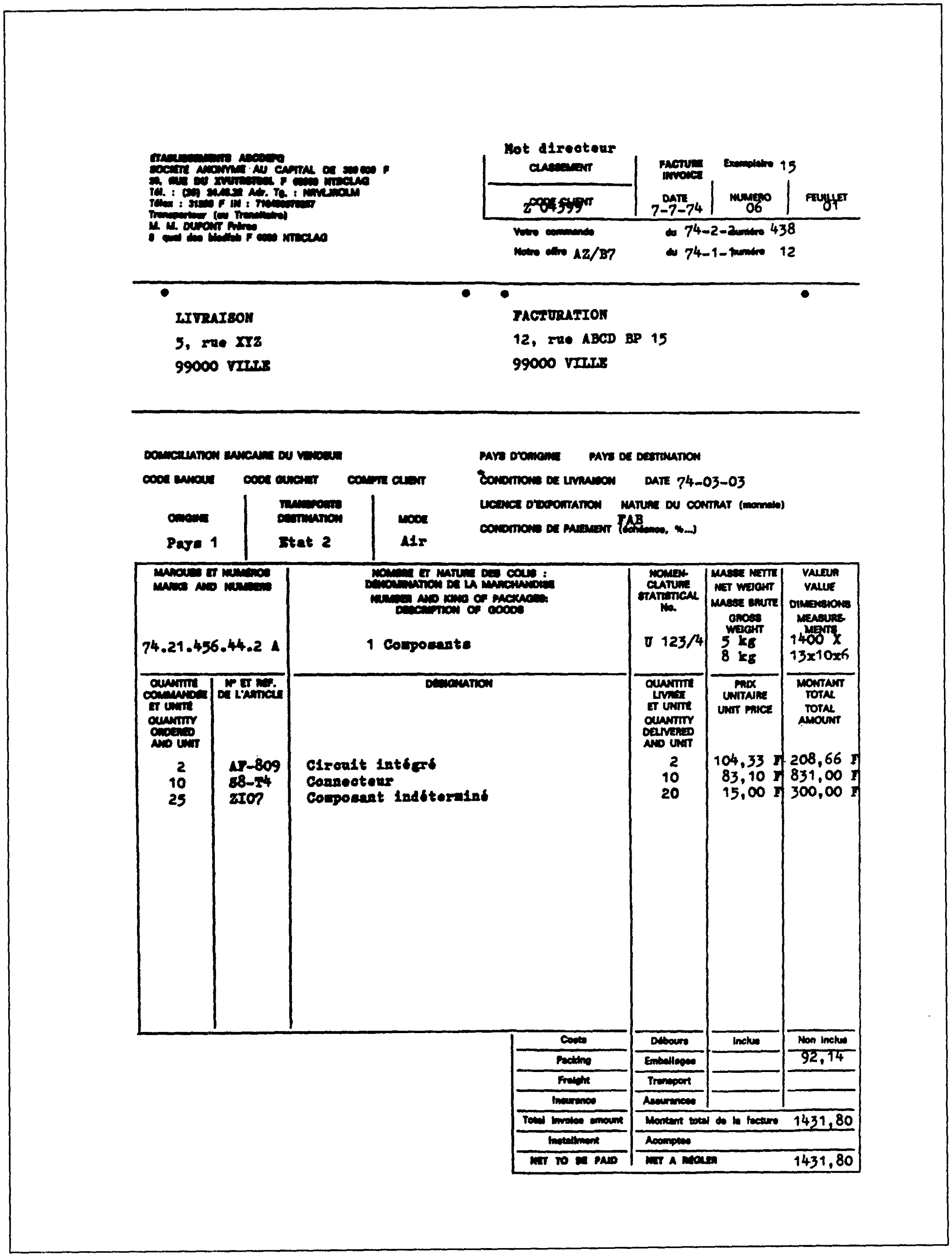
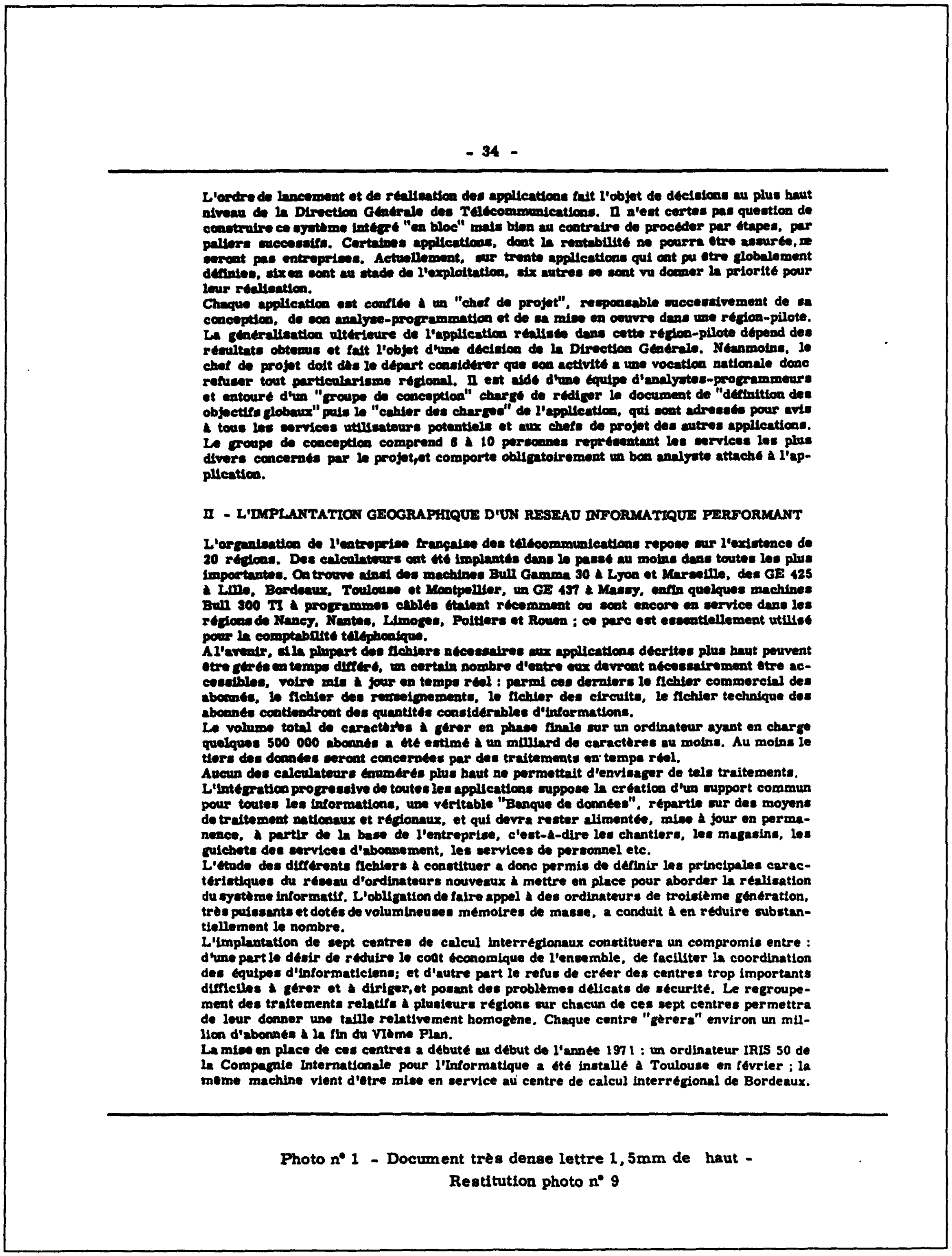


Figure A-3: CCITT test image #3



\- 34 -

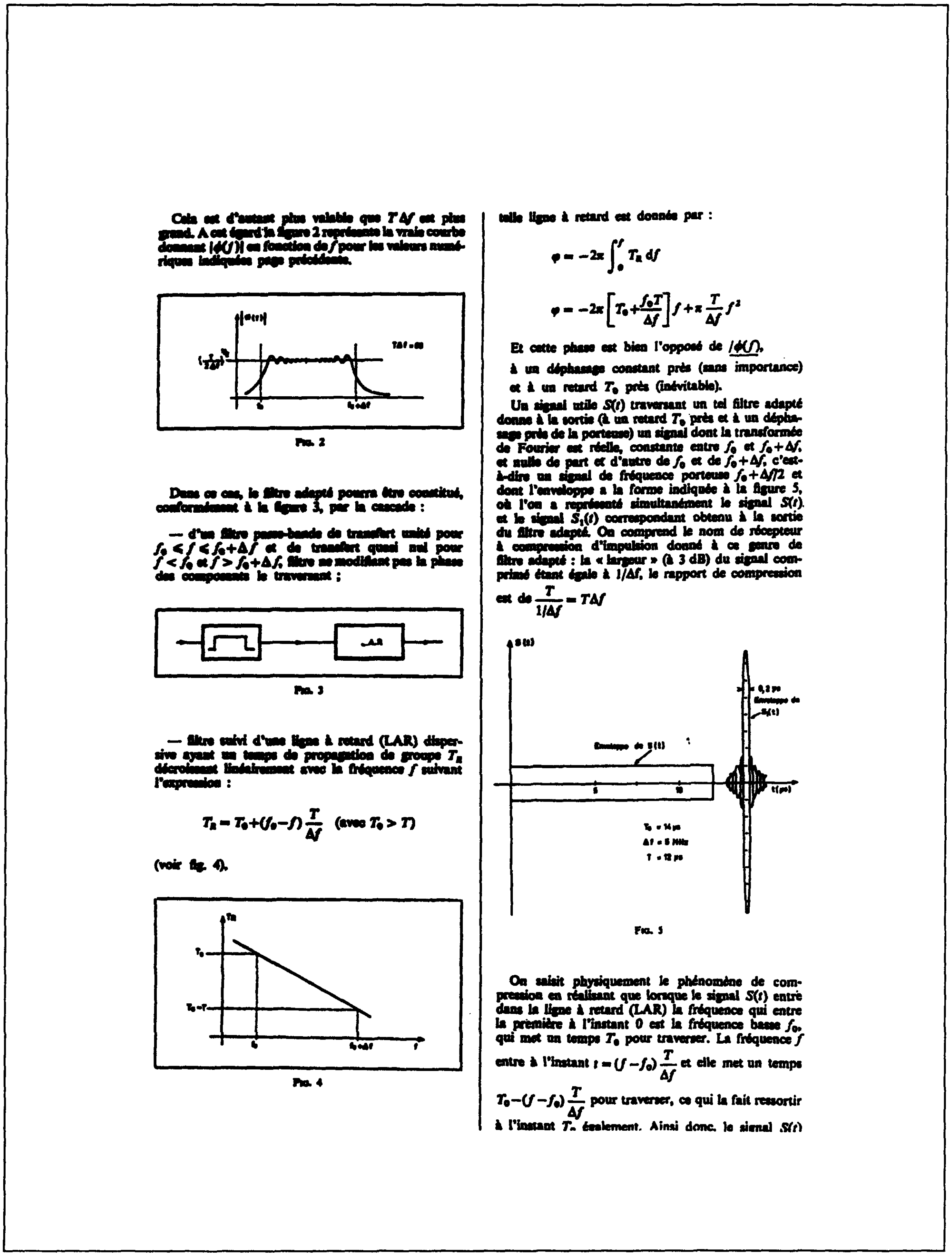
L'ordre de lancement et de réalisation des applications fait l'objet de décisions au plus haut niveau de la Direction Générale des Télécommunications. Il n'est certes pas question de construire ce système intégré "en bloc" mais bien au contraire de procéder par étapes, par paliers successifs. Certaines applications, dont la rentabilité ne pourra être assurée, ne seront pas entreprises. Actuellement, sur trente applications qui ont pu être globalement définies, six en sont au stade de l'exploitation, six autres se sont vu donner la priorité pour leur réalisation.
Chaque application est confiée à un "chef de projet", responsable successivement de sa conception, de son analyse-programmation et de sa mise en oeuvre dans une région-pilote. La généralisation ultérieure de l'application réalisée dans cette région-pilote dépend des résultats obtenus et fait l'objet d'une décision de la Direction Générale. Néanmoins, le chef de projet doit dès le départ considérer que son activité a une vocation nationale donc refuser tout particularisme régional. Il est aidé d'une équipe d'analystes-programmeurs et entouré d'un "groupe de conception" chargé de rédiger le document de "définition des objectifs globaux" puis le "cahier des charges" de l'application, qui sont adressés pour avis à tous les services utilisateurs potentiels et aux chefs de projet des autres applications. Le groupe de conception comprend 6 à 10 personnes représentant les services les plus divers concernés par le projet, et comporte obligatoirement un bon analyste attaché à l'application.

II - L'IMPLANTATION GEOGRAPHIQUE D'UN RESEAU INFORMATIQUE PERFORMANT

L'organisation de l'entreprise française des télécommunications repose sur l'existence de 20 régions. Des calculateurs ont été implantés dans le passé au moins dans toutes les plus importantes. On trouve ainsi des machines Bull Gamma 30 à Lyon et Marseille, des GE 425 à Lille, Bordeaux, Toulouse et Montpellier, un GE 437 à Massy, enfin quelques machines Bull 300 TI à programmes câblés étaient récemment ou sont encore en service dans les régions de Nancy, Nantes, Limoges, Poitiers et Rouen ; ce parc est essentiellement utilisé pour la comptabilité téléphonique.
A l'avenir, si la plupart des fichiers nécessaires aux applications décrites plus haut peuvent être gérés en temps différé, un certain nombre d'entre eux devront nécessairement être accessibles, voire mis à jour en temps réel : parmi ces derniers le fichier commercial des abonnés, le fichier des renseignements, le fichier des circuits, le fichier technique des abonnés contiendront des quantités considérables d'informations.
Le volume total de caractères à gérer en phase finale sur un ordinateur ayant en charge quelques 500 000 abonnés a été estimé à un milliard de caractères au moins. Au moins le tiers des données seront concernées par des traitements en temps réel.
Aucun des calculateurs énumérés plus haut ne permettait d'envisager de tels traitements.
L'intégration progressive de toutes les applications suppose la création d'un support commun pour toutes les informations, une véritable "Banque de données", répartie sur des moyens de traitement nationaux et régionaux, et qui devra rester alimentée, mise à jour en permanence, à partir de la base de l'entreprise, c'est-à-dire les chantiers, les magasins, les guichets des services d'abonnement, les services de personnel etc.
L'étude des différents fichiers à constituer a donc permis de définir les principales caractéristiques du réseau d'ordinateurs nouveaux à mettre en place pour aborder la réalisation du système informatif. L'obligation de faire appel à des ordinateurs de troisième génération, très puissants et dotés de volumineuses mémoires de masse, a conduit à en réduire substantiellement le nombre.
L'implantation de sept centres de calcul interrégionaux constituera un compromis entre : d'une part le désir de réduire le coût économique de l'ensemble, de faciliter la coordination des équipes d'informaticiens; et d'autre part le refus de créer des centres trop importants difficiles à gérer et à diriger, et posant des problèmes délicats de sécurité. Le regroupement des traitements relatifs à plusieurs régions sur chacun de ces sept centres permettra de leur donner une taille relativement homogène. Chaque centre "gèrera" environ un million d'abonnés à la fin du VIème Plan.
La mise en place de ces centres a débuté au début de l'année 1971 : un ordinateur IRIS 50 de la Compagnie Internationale pour l'Informatique a été installé à Toulouse en février ; la même machine vient d'être mise en service au centre de calcul interrégional de Bordeaux.

Photo n° 1 - Document très dense lettre 1,5mm de haut -
Restitution photo n° 9

Figure A-4: CCITT test image #4



Cela est d'autant plus valable que $T\Delta f$ est plus grand. A cet égard la figure 2 représente la vraie courbe donnant $|\phi(f)|$ en fonction de f pour les valeurs numériques indiquées page précédente.

FIG. 2

Dans ce cas, le filtre adapté pourra être constitué, conformément à la figure 3, par la cascade :

— d'un filtre passe-bande de transfert unité pour $f_0 \leq f \leq f_0+\Delta f$ et de transfert quasi nul pour $f < f_0$ et $f > f_0+\Delta f$, filtre ne modifiant pas la phase des composants le traversant ;

FIG. 3

— filtre suivi d'une ligne à retard (LAR) dispersive ayant un temps de propagation de groupe T_R décroissant linéairement avec la fréquence f suivant l'expression :

$$T_R = T_0+(f_0-f)\frac{T}{\Delta f} \quad (\text{avec } T_0 > T)$$

(voir fig. 4).

FIG. 4

telle ligne à retard est donnée par :

$$\varphi = -2\pi \int_0^f T_R \, df$$

$$\varphi = -2\pi \left[T_0+\frac{f_0 T}{\Delta f}\right] f + \pi \frac{T}{\Delta f} f^2$$

Et cette phase est bien l'opposé de $/\phi(f)$,

à un déphasage constant près (sans importance)

et à un retard T_0 près (inévitable).

Un signal utile $S(t)$ traversant un tel filtre adapté donne à la sortie (à un retard T_0 près et à un déphasage près de la porteuse) un signal dont la transformée de Fourier est réelle, constante entre f_0 et $f_0+\Delta f$, et nulle de part et d'autre de f_0 et de $f_0+\Delta f$, c'est-à-dire un signal de fréquence porteuse $f_0+\Delta f/2$ et dont l'enveloppe a la forme indiquée à la figure 5, où l'on a représenté simultanément le signal $S(t)$ et le signal $S_1(t)$ correspondant obtenu à la sortie du filtre adapté. On comprend le nom de récepteur à compression d'impulsion donné à ce genre de filtre adapté : la « largeur » (à 3 dB) du signal comprimé étant égale à $1/\Delta f$, le rapport de compression est de $\frac{T}{1/\Delta f} = T\Delta f$

FIG. 5

On saisit physiquement le phénomène de compression en réalisant que lorsque le signal $S(t)$ entre dans la ligne à retard (LAR) la fréquence qui entre la première à l'instant 0 est la fréquence basse f_0, qui met un temps T_0 pour traverser. La fréquence f entre à l'instant $t = (f-f_0)\frac{T}{\Delta f}$ et elle met un temps $T_0-(f-f_0)\frac{T}{\Delta f}$ pour traverser, ce qui la fait ressortir à l'instant T_0 également. Ainsi donc, le signal $S(t)$

Figure A-5: CCITT test image #5

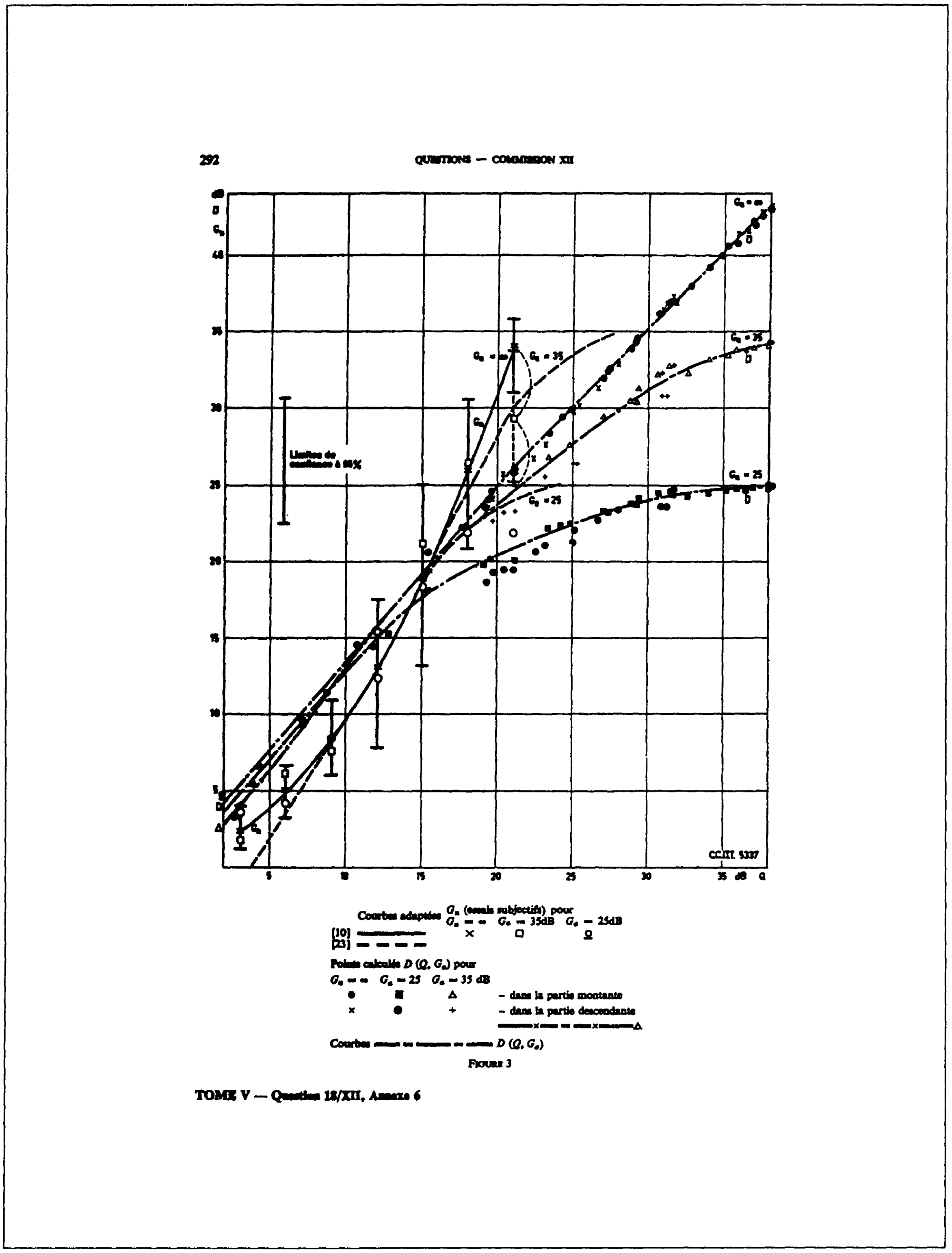
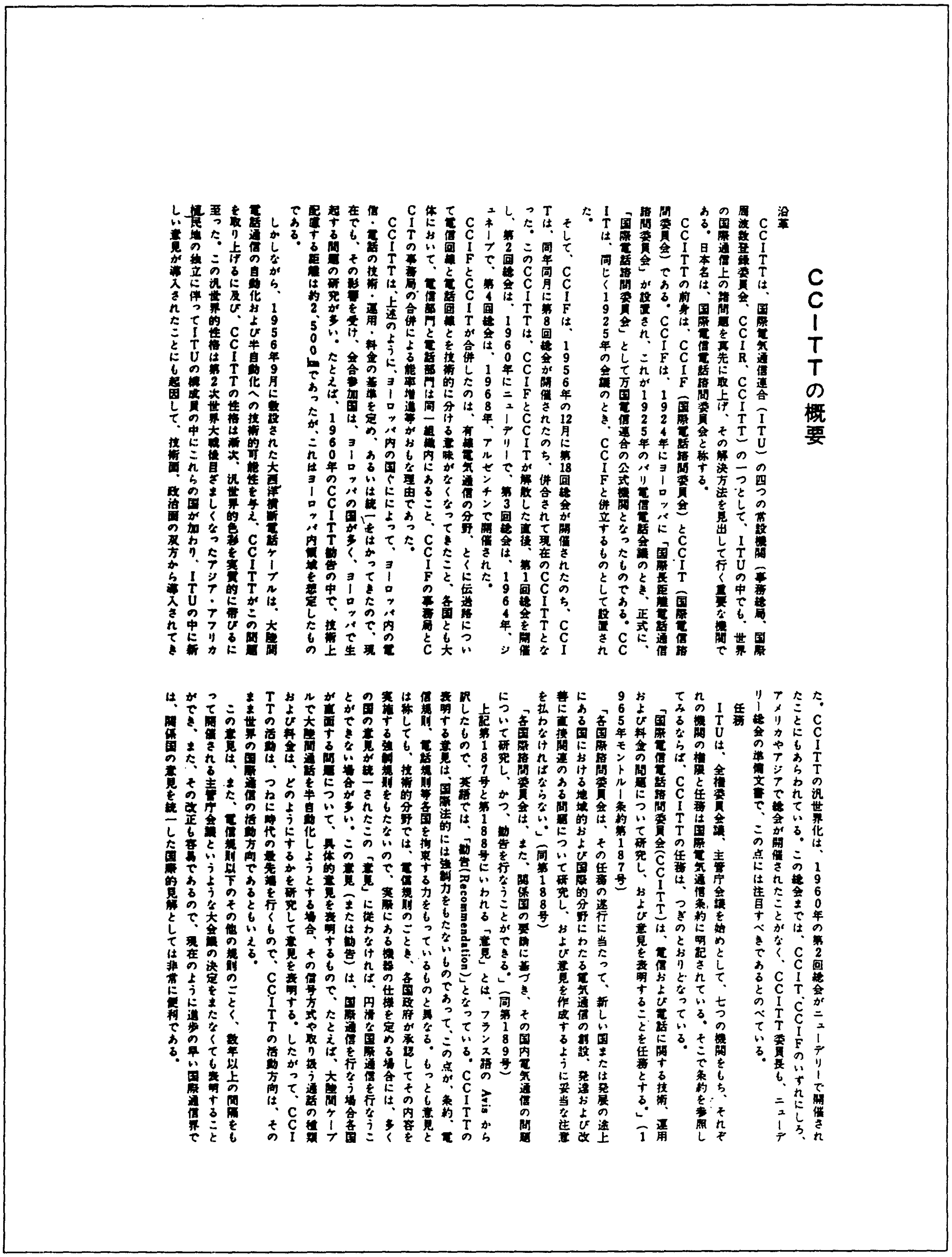


Figure A-6: CCITT test image #6



CCITTの概要

沿革

　CCITTは、国際電気通信連合（ITU）の四つの常設機関（事務総局、国際周波数登録委員会、CCIR、CCITT）の一つとして、ITUの中でも、世界の国際通信上の諸問題を真先に取上げ、その解決方法を見出して行く重要な機関である。日本名は、国際電信電話諮問委員会と称する。

　CCITTの前身は、CCIF（国際電話諮問委員会）とCCIT（国際電信諮問委員会）である。CCIFは、1924年にヨーロッパに「国際長距離電話通信諮問委員会」が設置され、これが1925年のパリ電信電話会議のとき、正式に「国際電話諮問委員会」として万国電信連合の公式機関となったものである。CCITは、同じく1925年の会議のとき、CCIFと併立するものとして設置された。

　そして、CCIFは、1956年の12月に第18回総会が開催されたのち、CCITは、同年同月に第8回総会が開催されたのち、併合されて現在のCCITTとなった。このCCITTは、CCIFとCCITが解散した直後、第1回総会を開催し、第2回総会は、1960年にニューデリーで、第3回総会は、1964年、ジュネーブで、第4回総会は、1968年、アルゼンチンで開催された。

　CCIFとCCITが合併したのは、有線電気通信の分野、とくに伝送路について電信回線と電話回線とを技術的に分ける意味がなくなってきたこと、各国とも大体において、電信部門と電話部門は同一組織内にあること、CCIFの事務局とCCITの事務局の合併による能率増進等がおもな理由であった。

　CCITTは、上述のように、ヨーロッパ内の国ぐにによって、ヨーロッパ内の電信・電話の技術・運用・料金の基準を定め、あるいは統一をはかってきたので、現在でも、その影響を受け、全会参加国は、ヨーロッパの国が多く、ヨーロッパで生起する問題の研究が多い。たとえば、1960年のCCITT勧告の中で、技術上配慮する距離は約2,500kmであったが、これはヨーロッパ内領域を想定したものである。

　しかしながら、1956年9月に敷設された大西洋横断電話ケーブルは、大陸間電話通信の自動化および半自動化への技術的可能性を与え、CCITTがこの問題を取り上げるに及び、CCITTの性格は漸次、汎世界的色彩を実質的に帯びるに至った。この汎世界的性格は第2次世界大戦後目ざましくなったアジア・アフリカ植民地の独立に伴ってITUの構成員の中にこれらの国が加わり、ITUの中に新しい意見が導入されたことにも起因して、技術面、政治面の双方から導入されてきた。CCITTの汎世界化は、1960年の第2回総会がニューデリーで開催されたことにもあらわれている。この総会までは、CCIT、CCIFのいずれにしろ、アメリカやアジアで総会が開催されたことがなく、CCITT委員長も、ニューデリー総会の準備文書で、この点には注目すべきであるとのべている。

任務

　ITUは、全権委員会議、主管庁会議を始めとして、七つの機関をもち、それぞれの機関の権限と任務は国際電気通信条約に明記されている。そこで条約を参照してみるならば、CCITTの任務は、つぎのとおりとなっている。

　「国際電信電話諮問委員会(CCITT)は、電信および電話に関する技術、運用および料金の問題について研究し、および意見を表明することを任務とする。」（1965年モントルー条約第187号）

　「各国際諮問委員会は、その任務の遂行に当たって、新しい国または発展の途上にある国における地域的および国際的分野にわたる電気通信の創設、発達および改善に直接関連のある問題について研究し、および意見を作成するように妥当な注意を払わなければならない。」（同第188号）

　「各国際諮問委員会は、また、関係国の要請に基づき、その国内電気通信の問題について研究し、かつ、勧告を行なうことができる。」（同第189号）

　上記第187号と第188号にいわれる「意見」とは、フランス語の Avis から訳したもので、英語では、「勧告(Recommendation)」となっている。CCITTの表明する意見は、国際法的には強制力をもたないものであって、この点が、条約、電信規則、電話規則等各国を拘束する力をもっているものと異なる。もっとも意見とは称しても、技術的分野では、電信規則のごとき、各国政府が承認してその内容を実施する強制規則をもたないので、実際にある機器の仕様を定める場合には、多くの国の意見が統一されたこの「意見」に従わなければ、円滑な国際通信を行なうことができない場合が多い。この意見（または勧告）は、国際通信を行なう場合各国が直面する問題について、具体的意見を表明するもので、たとえば、大陸間ケーブルで大陸間通話を半自動化しようとする場合、その信号方式や取り扱う通話の種類および料金は、どのようにするかを研究して意見を表明する。したがって、CCITTの活動は、つねに時代の最先端を行くもので、CCITTの活動方向は、そのまま世界の国際通信の活動方向であるともいえる。

　この意見は、また、電信規則以下のその他の規則のごとく、数年以上の間隔をもって開催される主管庁会議というような大会議の決定をまたなくても表明することができ、また、その改正も容易であるので、現在のように進歩の早い国際通信界では、関係国の意見を統一した国際的見解としては非常に便利である。

Figure A-7: CCITT test image #7

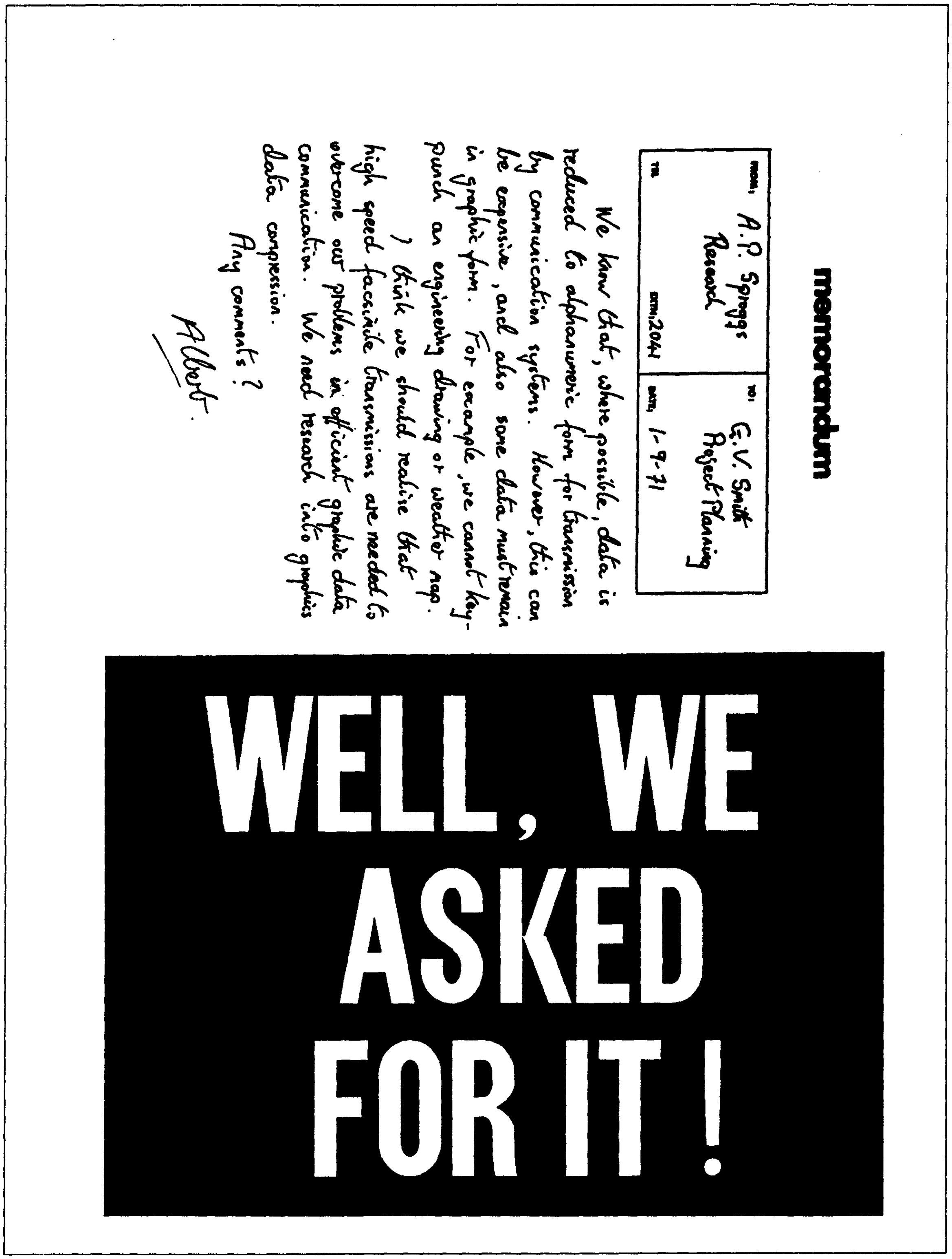


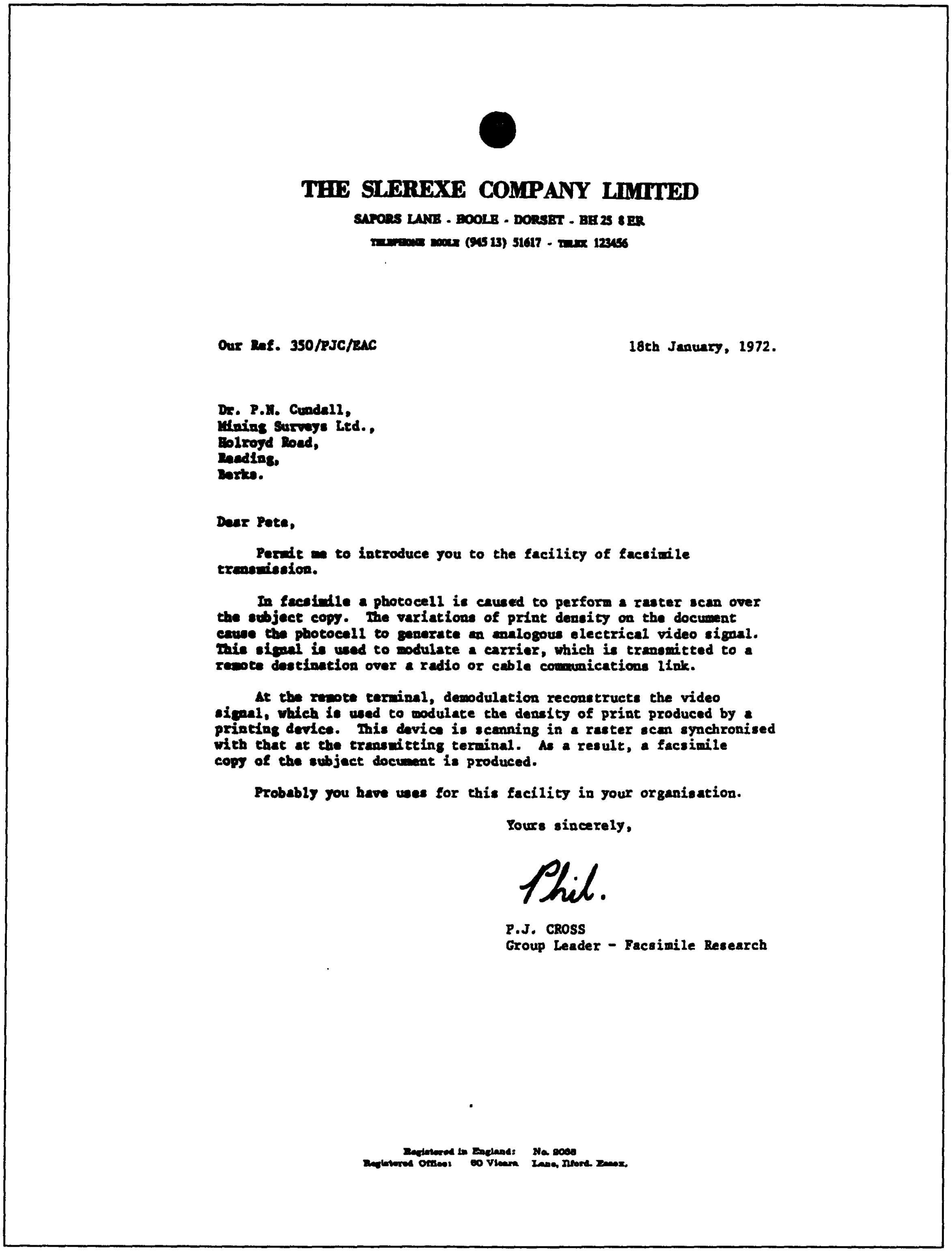
Figure A-8: CCITT test image #8

Appendix B

CAFC-Processed CCITT Test Images

The eight standard CCITT test documents were encoded with Content-Adaptive Facsimile Coding. The compressed images were then passed through the CAFC decoder to produce the following eight reconstructed images. The CAFC parameters that were used are the ones listed in the file `CAFC.h` in Appendix E.

All of the images are in fine mode (200 pixels/inch) but are reduced by 30% along each axis on the following pages.



THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH 25 8 ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC

18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

Registered in England: No. 2038
Registered Office: 60 Vicars Lane, Ilford. Essex.

Figure B-1: CCITT test image #1

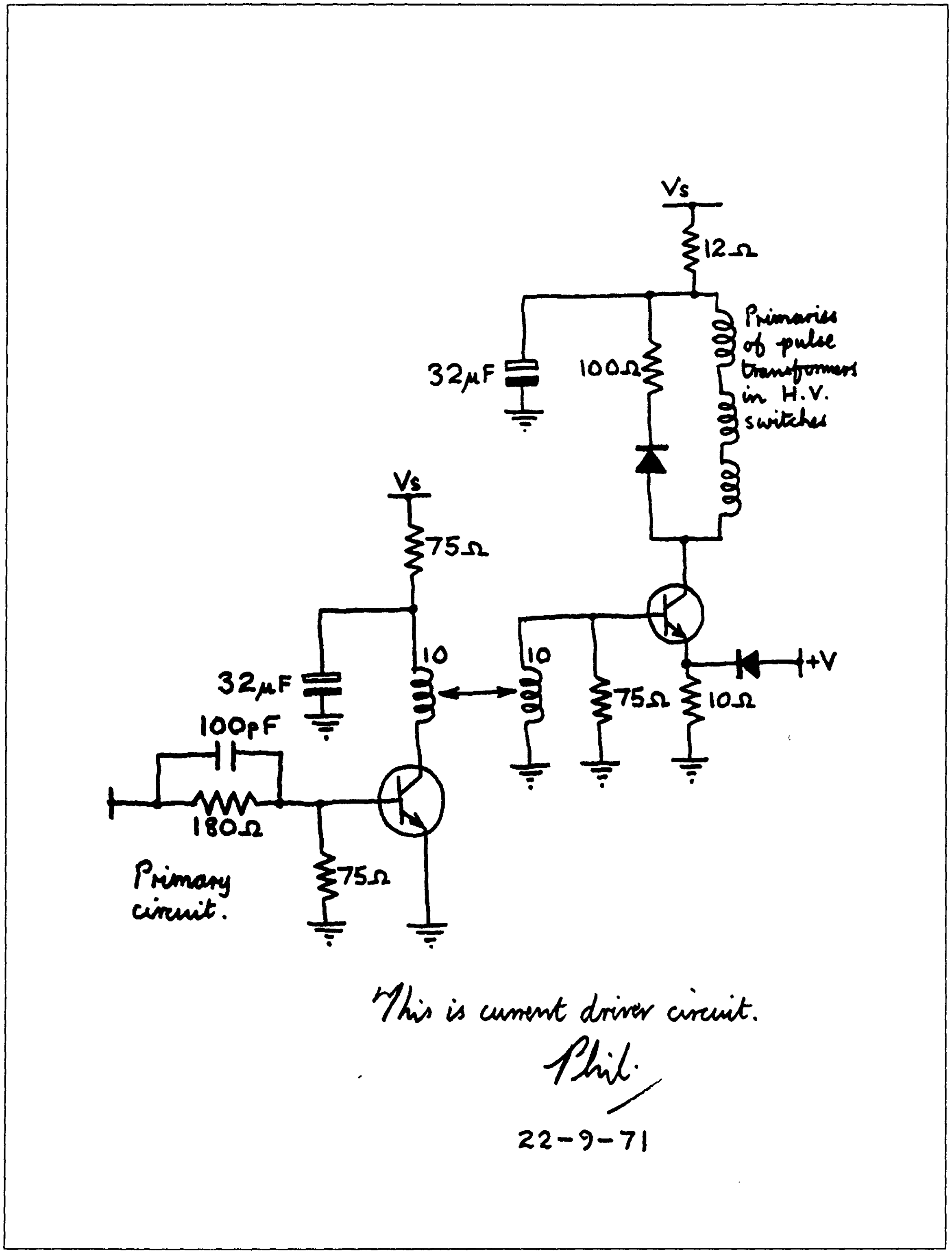
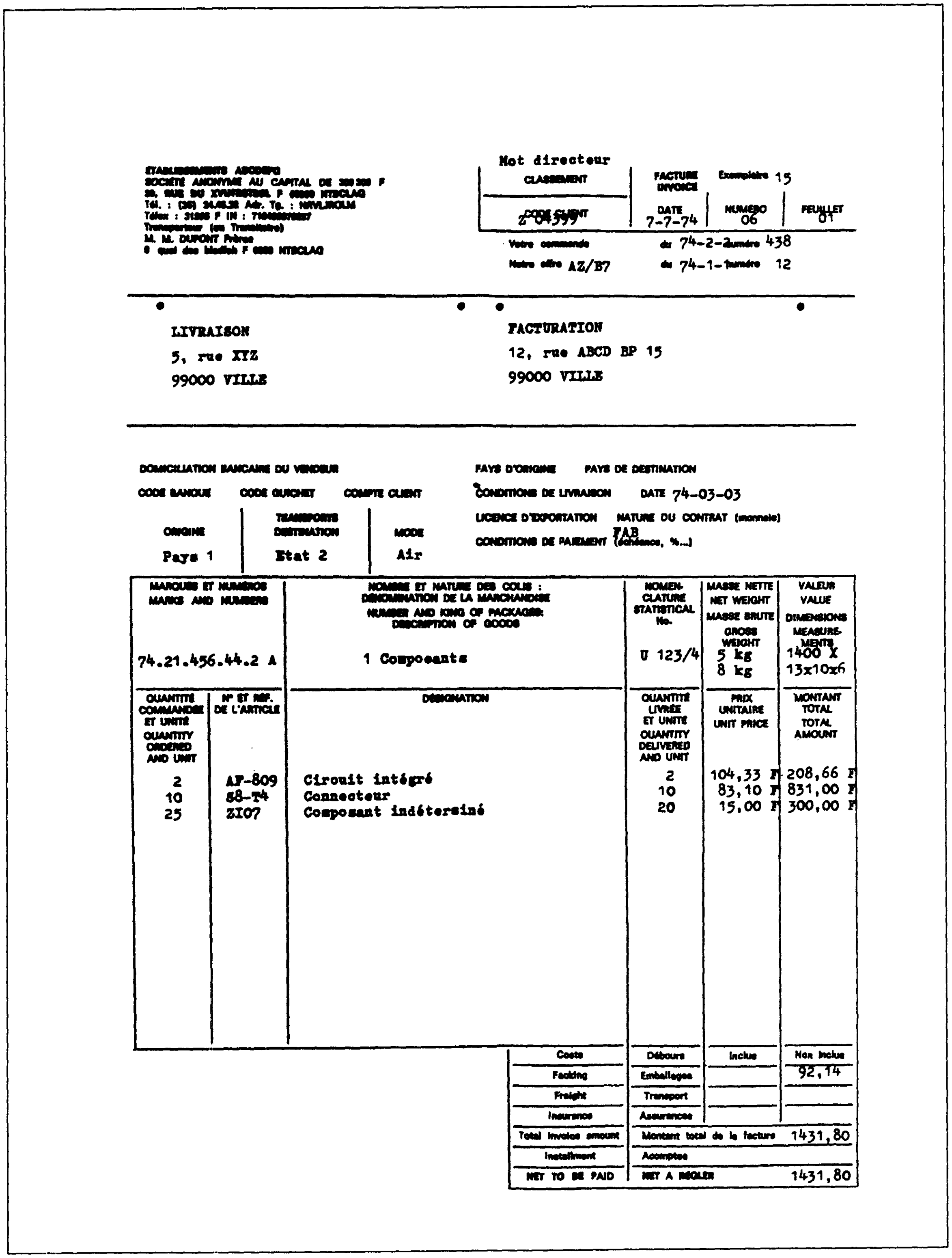


Figure B-2: CCITT test image #2



Mot directeur

CLASSEMENT	FACTURE INVOICE	Exemplaire 15	
CODE CLIENT Z 04599	DATE 7-7-74	NUMERO 06	FEUILLET 01

Votre commande du 74-2-2 numéro 438

Notre offre AZ/B7 du 74-1-1 numéro 12

LIVRAISON
5, rue XYZ
99000 VILLE

FACTURATION
12, rue ABCD BP 15
99000 VILLE

DOMICILIATION BANCAIRE DU VENDEUR

CODE BANQUE — CODE GUICHET — COMPTE CLIENT

PAYS D'ORIGINE — PAYS DE DESTINATION

CONDITIONS DE LIVRAISON — DATE 74-03-03

LICENCE D'EXPORTATION — NATURE DU CONTRAT (monnaie)

CONDITIONS DE PAIEMENT (échéance, %...) FAB

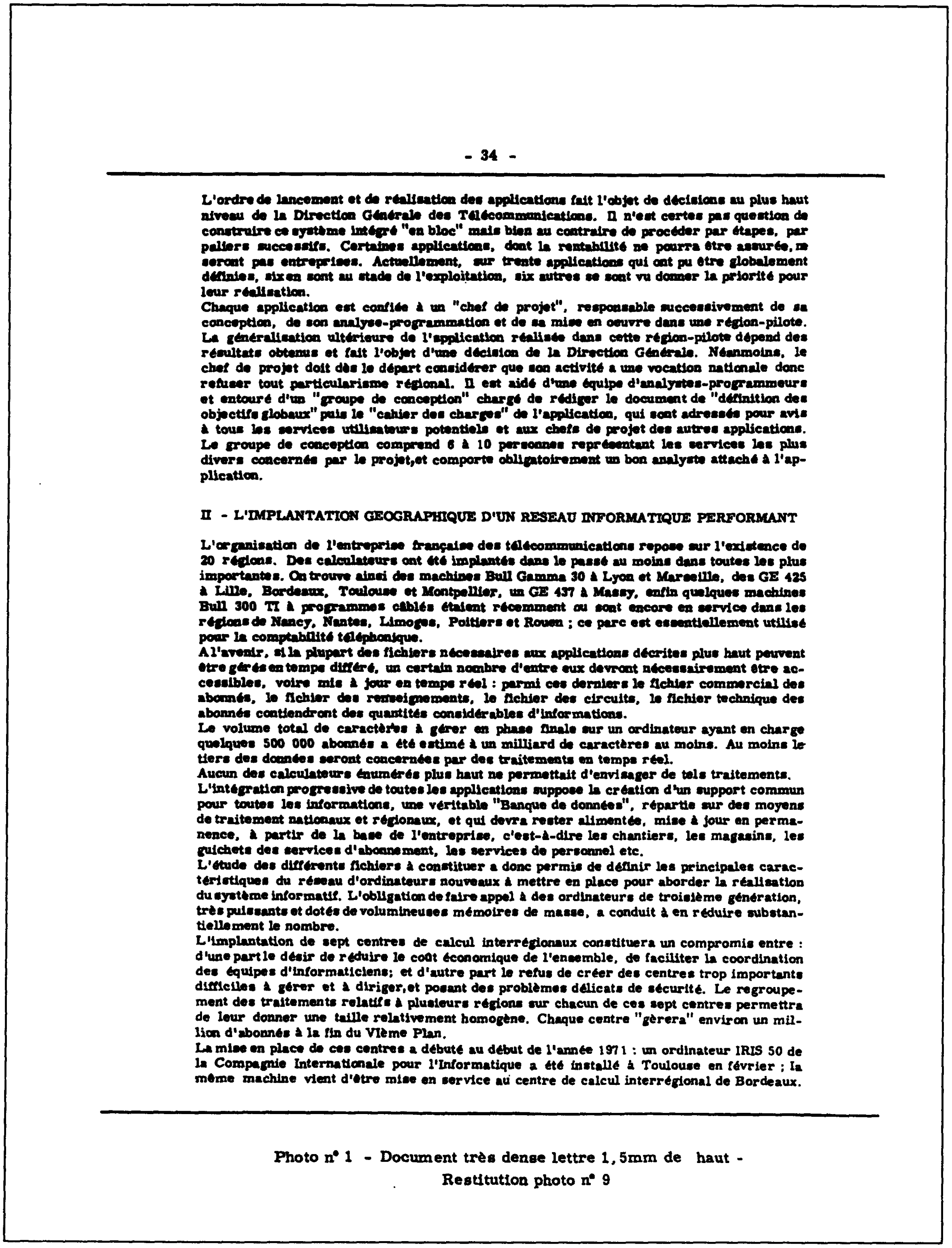
ORIGINE	TRANSPORTS DESTINATION	MODE
Pays 1	Etat 2	Air

MARQUES ET NUMÉROS MARKS AND NUMBERS	NOMBRE ET NATURE DES COLIS : DÉNOMINATION DE LA MARCHANDISE NUMBER AND KIND OF PACKAGES: DESCRIPTION OF GOODS	NOMENCLATURE STATISTICAL No.	MASSE NETTE NET WEIGHT MASSE BRUTE GROSS WEIGHT	VALEUR VALUE DIMENSIONS MEASUREMENTS
74.21.456.44.2 A	1 Composants	U 123/4	5 kg 8 kg	1400 X 13x10x6

QUANTITÉ COMMANDÉE ET UNITÉ QUANTITY ORDERED AND UNIT	N° ET RÉF. DE L'ARTICLE	DESIGNATION	QUANTITÉ LIVRÉE ET UNITÉ QUANTITY DELIVERED AND UNIT	PRIX UNITAIRE UNIT PRICE	MONTANT TOTAL TOTAL AMOUNT
2	AF-809	Circuit intégré	2	104,33 F	208,66 F
10	88-T4	Connecteur	10	83,10 F	831,00 F
25	ZI07	Composant indéterminé	20	15,00 F	300,00 F

Costs	Débours	Inclus	Non inclus
Packing	Emballages		92,14
Freight	Transport		
Insurance	Assurances		
Total invoice amount	Montant total de la facture		1431,80
Instalment	Acomptes		
NET TO BE PAID	NET A RÉGLER		1431,80

Figure B-3: CCITT test image #3



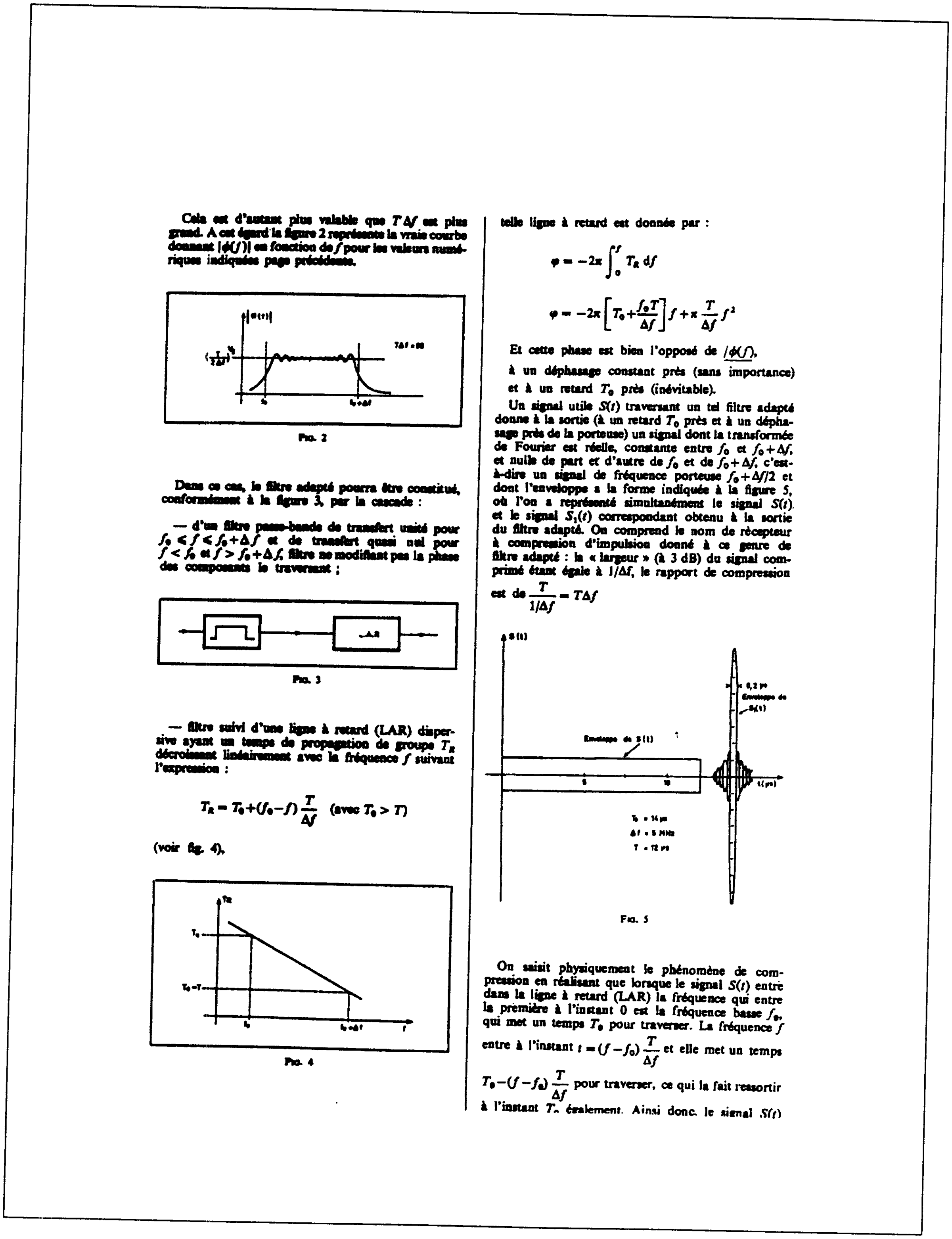
L'ordre de lancement et de réalisation des applications fait l'objet de décisions au plus haut niveau de la Direction Générale des Télécommunications. Il n'est certes pas question de construire ce système intégré "en bloc" mais bien au contraire de procéder par étapes, par paliers successifs. Certaines applications, dont la rentabilité ne pourra être assurée, ne seront pas entreprises. Actuellement, sur trente applications qui ont pu être globalement définies, six en sont au stade de l'exploitation, six autres se sont vu donner la priorité pour leur réalisation.
Chaque application est confiée à un "chef de projet", responsable successivement de sa conception, de son analyse-programmation et de sa mise en oeuvre dans une région-pilote. La généralisation ultérieure de l'application réalisée dans cette région-pilote dépend des résultats obtenus et fait l'objet d'une décision de la Direction Générale. Néanmoins, le chef de projet doit dès le départ considérer que son activité a une vocation nationale donc refuser tout particularisme régional. Il est aidé d'une équipe d'analystes-programmeurs et entouré d'un "groupe de conception" chargé de rédiger le document de "définition des objectifs globaux" puis le "cahier des charges" de l'application, qui sont adressés pour avis à tous les services utilisateurs potentiels et aux chefs de projet des autres applications. Le groupe de conception comprend 6 à 10 personnes représentant les services les plus divers concernés par le projet, et comporte obligatoirement un bon analyste attaché à l'application.

II - L'IMPLANTATION GEOGRAPHIQUE D'UN RESEAU INFORMATIQUE PERFORMANT

L'organisation de l'entreprise française des télécommunications repose sur l'existence de 20 régions. Des calculateurs ont été implantés dans le passé au moins dans toutes les plus importantes. On trouve ainsi des machines Bull Gamma 30 à Lyon et Marseille, des GE 425 à Lille, Bordeaux, Toulouse et Montpellier, un GE 437 à Massy, enfin quelques machines Bull 300 TI à programmes câblés étaient récemment ou sont encore en service dans les régions de Nancy, Nantes, Limoges, Poitiers et Rouen ; ce parc est essentiellement utilisé pour la comptabilité téléphonique.
A l'avenir, si la plupart des fichiers nécessaires aux applications décrites plus haut peuvent être gérés en temps différé, un certain nombre d'entre eux devront nécessairement être accessibles, voire mis à jour en temps réel : parmi ces derniers le fichier commercial des abonnés, le fichier des renseignements, le fichier des circuits, le fichier technique des abonnés contiendront des quantités considérables d'informations.
Le volume total de caractères à gérer en phase finale sur un ordinateur ayant en charge quelques 500 000 abonnés a été estimé à un milliard de caractères au moins. Au moins le tiers des données seront concernées par des traitements en temps réel.
Aucun des calculateurs énumérés plus haut ne permettait d'envisager de tels traitements.
L'intégration progressive de toutes les applications suppose la création d'un support commun pour toutes les informations, une véritable "Banque de données", répartie sur des moyens de traitement nationaux et régionaux, et qui devra rester alimentée, mise à jour en permanence, à partir de la base de l'entreprise, c'est-à-dire les chantiers, les magasins, les guichets des services d'abonnement, les services de personnel etc.
L'étude des différents fichiers à constituer a donc permis de définir les principales caractéristiques du réseau d'ordinateurs nouveaux à mettre en place pour aborder la réalisation du système informatif. L'obligation de faire appel à des ordinateurs de troisième génération, très puissants et dotés de volumineuses mémoires de masse, a conduit à en réduire substantiellement le nombre.
L'implantation de sept centres de calcul interrégionaux constituera un compromis entre : d'une part le désir de réduire le coût économique de l'ensemble, de faciliter la coordination des équipes d'informaticiens; et d'autre part le refus de créer des centres trop importants difficiles à gérer et à diriger, et posant des problèmes délicats de sécurité. Le regroupement des traitements relatifs à plusieurs régions sur chacun de ces sept centres permettra de leur donner une taille relativement homogène. Chaque centre "gèrera" environ un million d'abonnés à la fin du VIème Plan.
La mise en place de ces centres a débuté au début de l'année 1971 : un ordinateur IRIS 50 de la Compagnie Internationale pour l'Informatique a été installé à Toulouse en février ; la même machine vient d'être mise en service au centre de calcul interrégional de Bordeaux.

Photo n° 1 - Document très dense lettre 1,5mm de haut -
Restitution photo n° 9

Figure B-4: CCITT test image #4



Cela est d'autant plus valable que $T\Delta f$ est plus grand. A cet égard la figure 2 représente la vraie courbe donnant $|\phi(f)|$ en fonction de f pour les valeurs numériques indiquées page précédente.

Fig. 2

Dans ce cas, le filtre adapté pourra être constitué, conformément à la figure 3, par la cascade :

— d'un filtre passe-bande de transfert unité pour $f_0 \leq f \leq f_0+\Delta f$ et de transfert quasi nul pour $f < f_0$ et $f > f_0+\Delta f$, filtre ne modifiant pas la phase des composants le traversant ;

Fig. 3

— filtre suivi d'une ligne à retard (LAR) dispersive ayant un temps de propagation de groupe T_R décroissant linéairement avec la fréquence f suivant l'expression :

$$T_R = T_0+(f_0-f)\frac{T}{\Delta f} \quad (\text{avec } T_0 > T)$$

(voir fig. 4).

Fig. 4

telle ligne à retard est donnée par :

$$\varphi = -2\pi \int_0^f T_R \, df$$

$$\varphi = -2\pi\left[T_0+\frac{f_0 T}{\Delta f}\right] f + \pi\frac{T}{\Delta f} f^2$$

Et cette phase est bien l'opposé de $\underline{/\phi(f)}$, à un déphasage constant près (sans importance) et à un retard T_0 près (inévitable).

Un signal utile $S(t)$ traversant un tel filtre adapté donne à la sortie (à un retard T_0 près et à un déphasage près de la porteuse) un signal dont la transformée de Fourier est réelle, constante entre f_0 et $f_0+\Delta f$, et nulle de part et d'autre de f_0 et de $f_0+\Delta f$, c'est-à-dire un signal de fréquence porteuse $f_0+\Delta f/2$ et dont l'enveloppe a la forme indiquée à la figure 5, où l'on a représenté simultanément le signal $S(t)$ et le signal $S_1(t)$ correspondant obtenu à la sortie du filtre adapté. On comprend le nom de récepteur à compression d'impulsion donné à ce genre de filtre adapté : la « largeur » (à 3 dB) du signal comprimé étant égale à $1/\Delta f$, le rapport de compression est de $\frac{T}{1/\Delta f} = T\Delta f$

Fig. 5

On saisit physiquement le phénomène de compression en réalisant que lorsque le signal $S(t)$ entre dans la ligne à retard (LAR) la fréquence qui entre la première à l'instant 0 est la fréquence basse f_0, qui met un temps T_0 pour traverser. La fréquence f entre à l'instant $t = (f-f_0)\frac{T}{\Delta f}$ et elle met un temps $T_0-(f-f_0)\frac{T}{\Delta f}$ pour traverser, ce qui la fait ressortir à l'instant T_0 également. Ainsi donc le signal $S(t)$

Figure B-5: CCITT test image #5

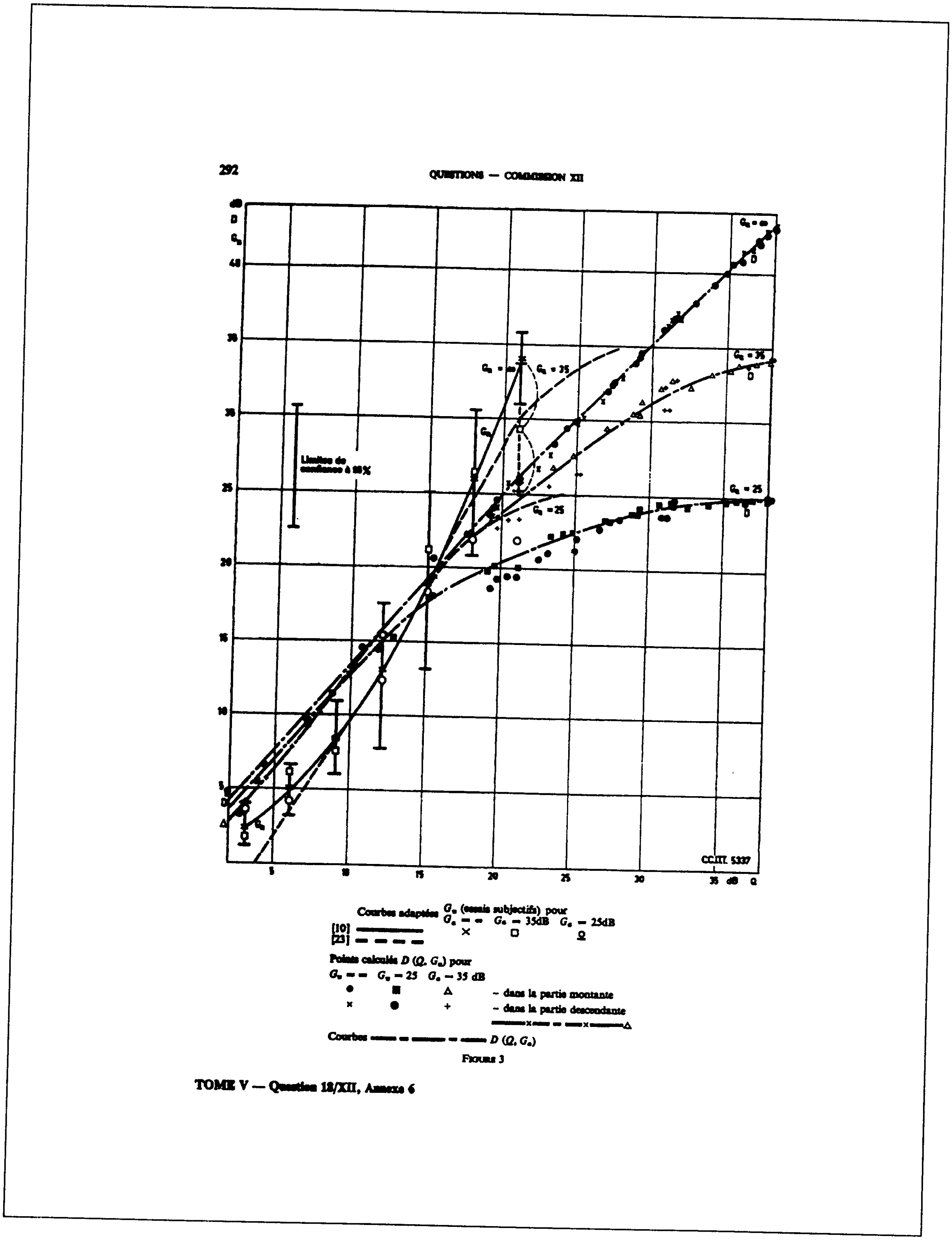


Figure B-6: CCITT test image #6

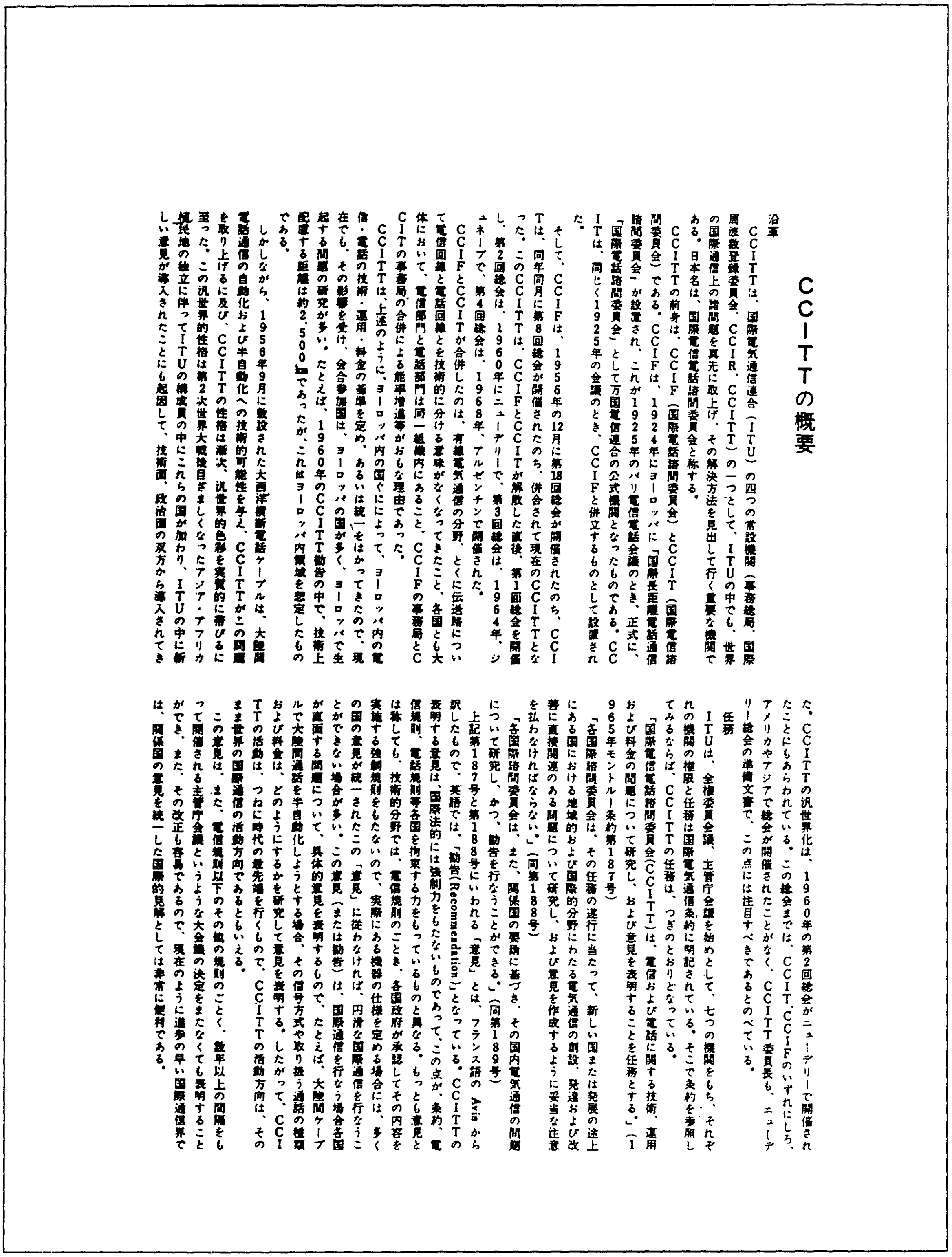


Figure B-7: CCITT test image #7

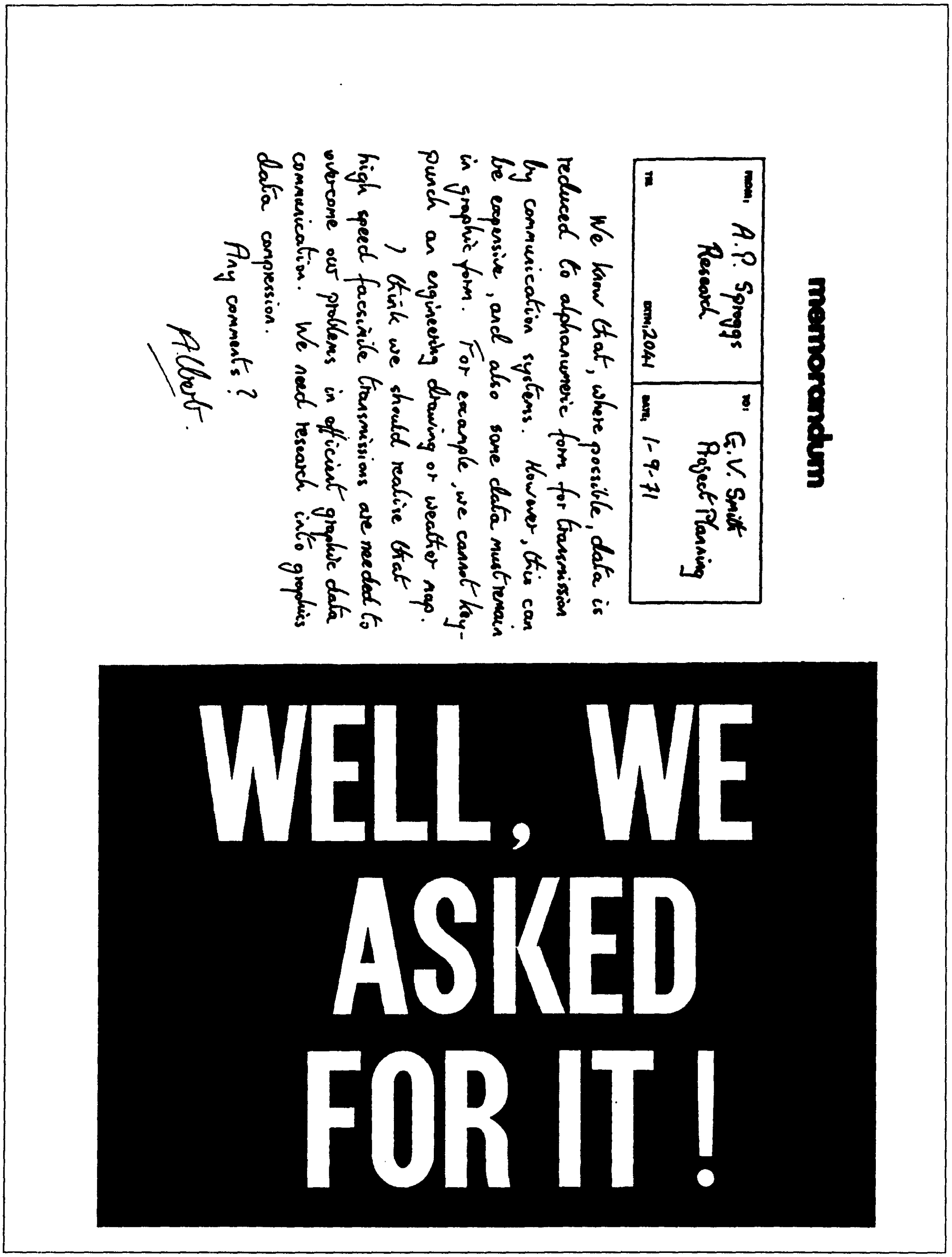


Figure B-8: CCITT test image #8

Appendix C

Typed Text Test Images

In Content-Adaptive Facsimile Coding, *symbol matching* is used to determine if an isolated symbol "matches" one that has already been detected and stored in the symbol library. The first stage of symbol matching, *feature matching*, is used to eliminate unlikely candidates early on by comparing high-level properties (or "features") of the symbols.

The following 9 test images were used to determine the effectiveness of a number of features at differentiating different instances of the same symbol. They contain 8 repetitions of 78 different characters in 3 fonts, 3 styles, and 3 sizes. Section 4.1.1 describes feature matching and Sect. 7.2 describes the procedure for selecting an optimal set of features in detail.

Each page was individually scanned into a facsimile machine, transmitted to a PC-based fax card, and then saved to a file on the host computer system. All of the images are in fine mode (200 pixels/inch) but are reduced by 60% along each axis on the following pages.

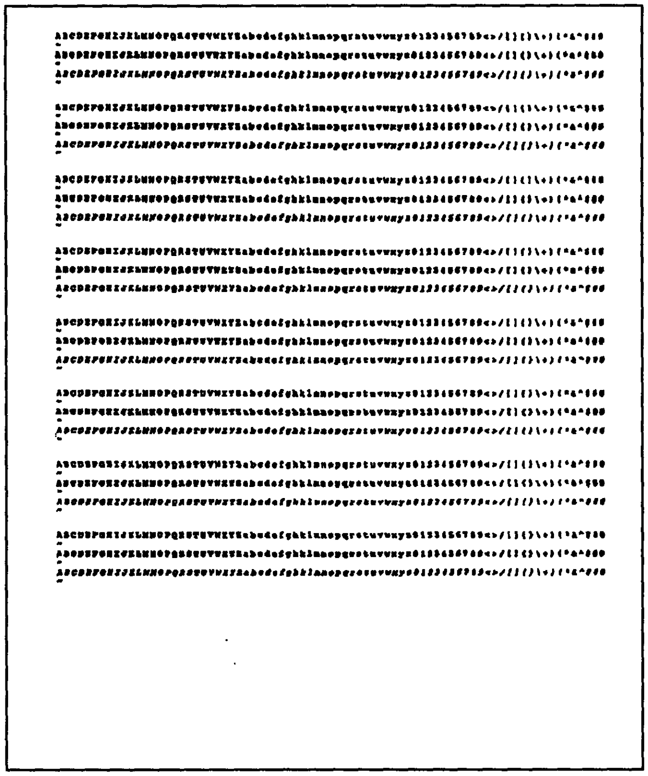


Figure C-1: Typed text image #1 – Courier 8pt

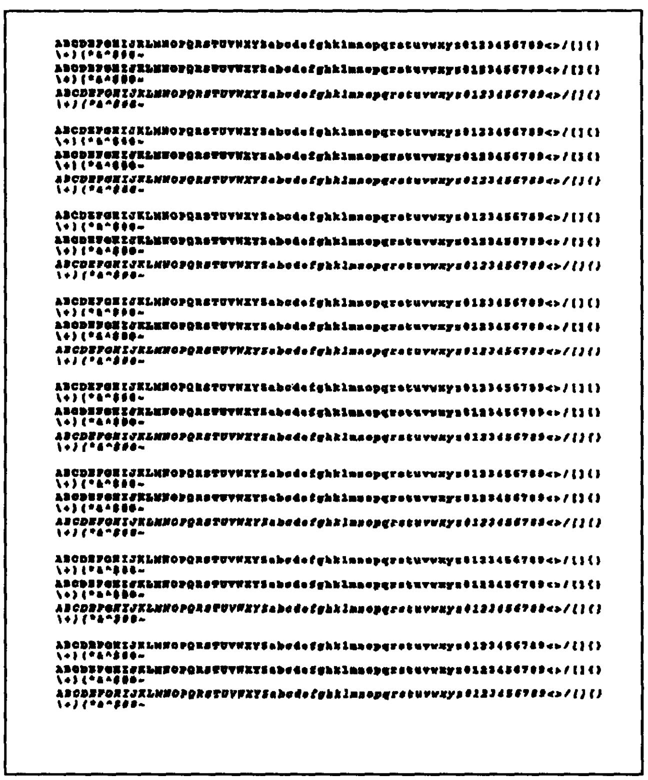


Figure C-2: Typed text image #2 – Courier 10pt

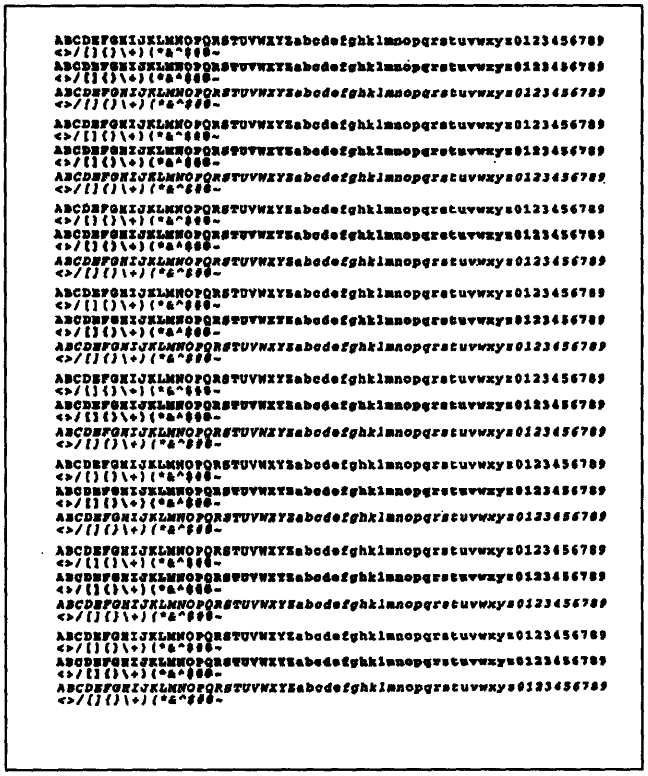


Figure C-3: Typed text image #3 – Courier 12pt

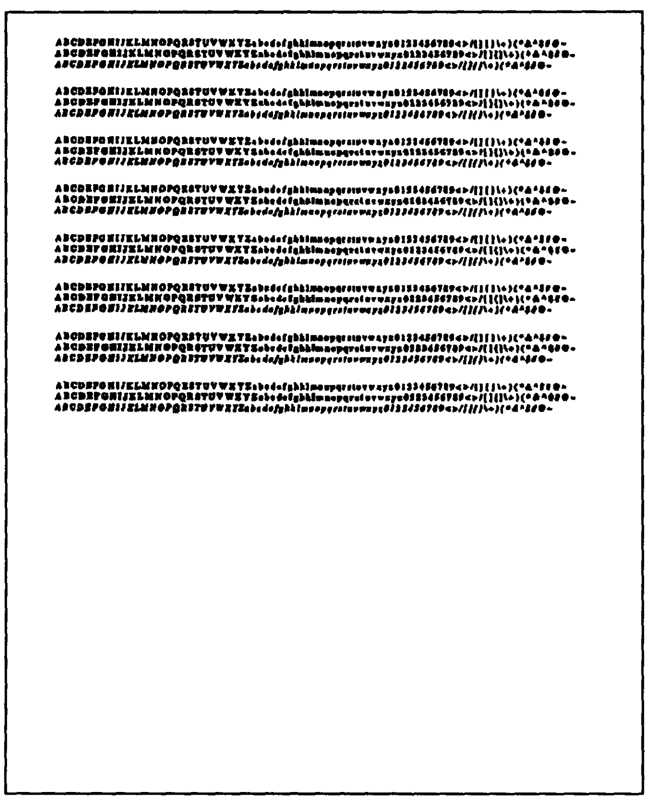


Figure C-4: Typed text image #4 – Times Roman 8pt

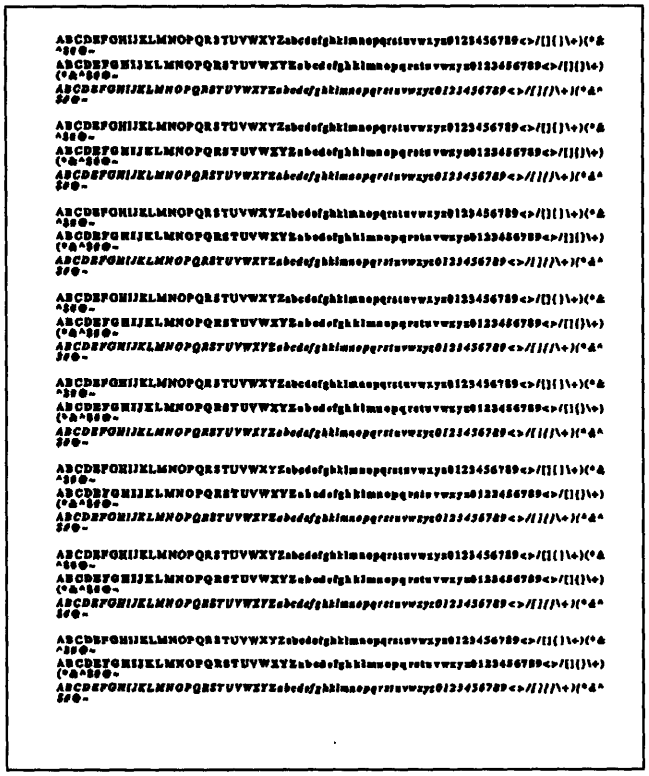


Figure C-5: Typed text image #5 – Times Roman 10pt

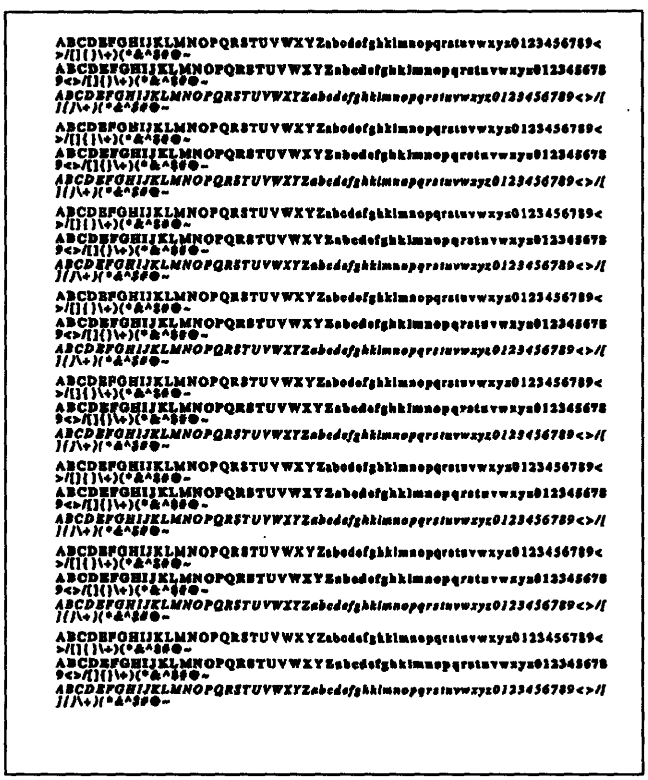


Figure C-6: Typed text image #6 – Times Roman 12pt

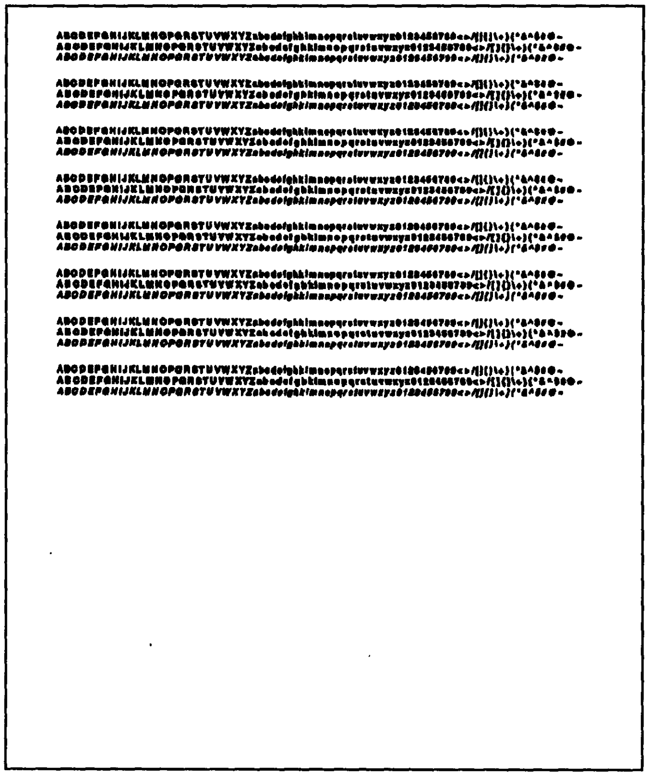


Figure C-7: Typed text image #7 – Helvetica 8pt

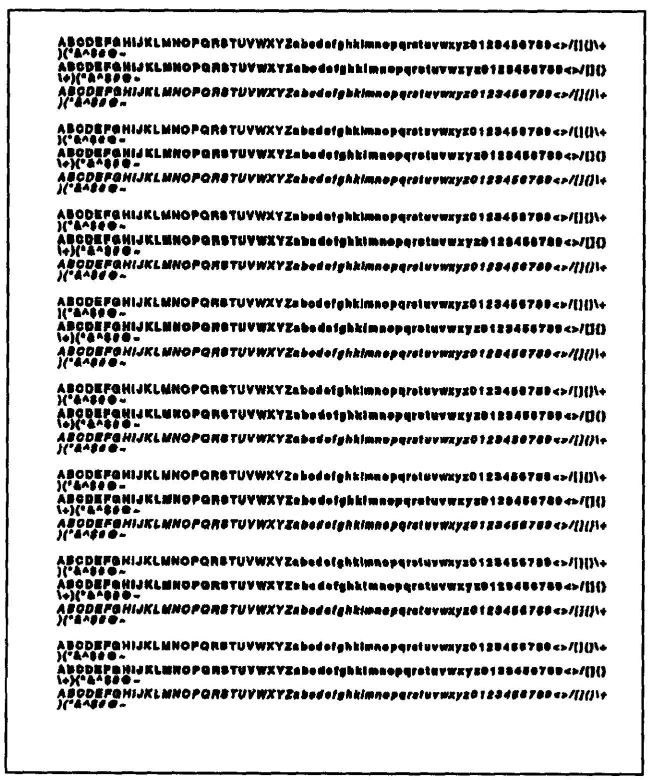


Figure C-8: Typed text image #8 – Helvetica 10pt

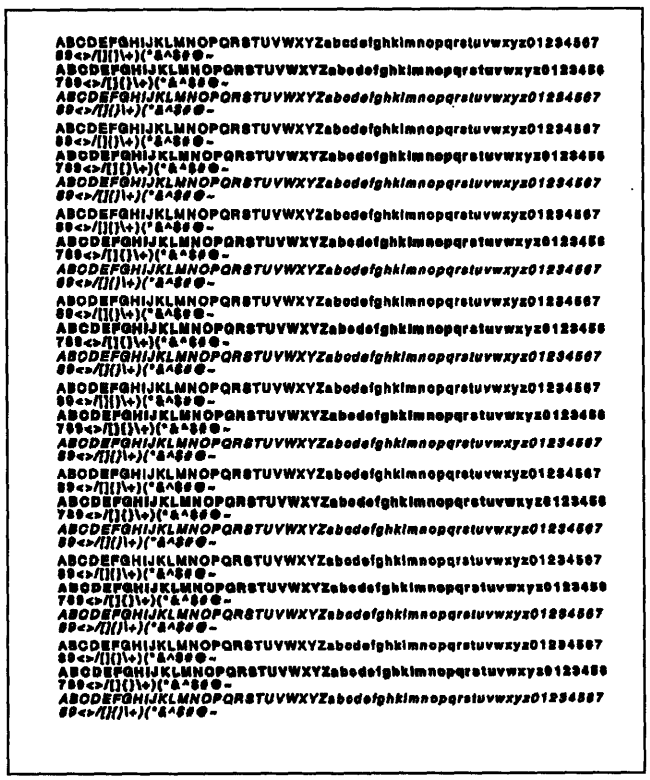


Figure C-9: Typed text image #9 – Helvetica 12pt

Appendix D

Training Set Images

The following 16 images constitute the training set that was used to calibrate the adjustable parameters of the Content-Adaptive Facsimile Coder. They are intended to be a fair representation of the types of documents that are typically transmitted via facsimile. Included in this set are pages containing typewritten text (in a variety of sizes, fonts, orientations, and styles), handwriting in English (from a number of different people), a diagram, and Chinese writing.

In particular, the training set was used to generate the set of statistics to prime the Optimized 2D Run-Length Coder. This is described in detail in Sect. 7.4.

Each page was individually scanned into a facsimile machine, transmitted to a PC-based fax card, and then saved to a file on the host computer system. All of the images are in fine mode (200 pixels/inch) but are reduced by 60% along each axis on the following pages.

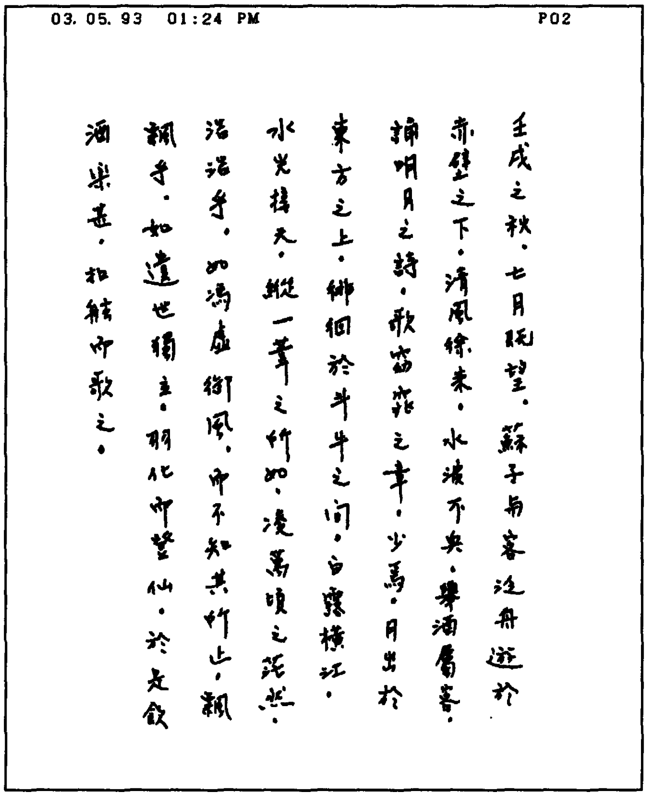
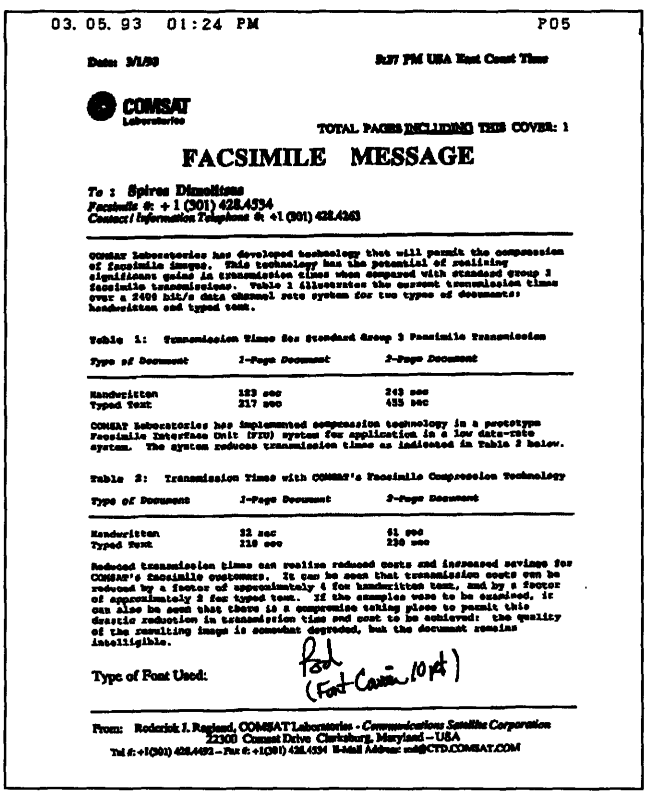


Figure D-1: Training set document #1



03. 05. 93 01:24 PM P05

Date: 3/1/93 8:37 PM USA East Coast Time

COMSAT Laboratories

TOTAL PAGES INCLUDING THIS COVER: 1

FACSIMILE MESSAGE

To : Spiros Dimolitsas
Facsimile #: +1 (301) 428.4534
Contact / Information Telephone #: +1 (301) 428.4363

COMSAT Laboratories has developed technology that will permit the compression of facsimile images. This technology has the potential of realizing significant gains in transmission times when compared with standard group 3 facsimile transmissions. Table 1 illustrates the current transmission times over a 2400 bit/s data channel rate system for two types of documents: handwritten and typed text.

Table 1: Transmission Times for Standard Group 3 Facsimile Transmission

Type of Document	*1-Page Document*	*2-Page Document*
Handwritten	123 sec	243 sec
Typed Text	217 sec	435 sec

COMSAT Laboratories has implemented compression technology in a prototype Facsimile Interface Unit (FIU) system for application in a low data-rate system. The system reduces transmission times as indicated in Table 2 below.

Table 2: Transmission Times with COMSAT's Facsimile Compression Technology

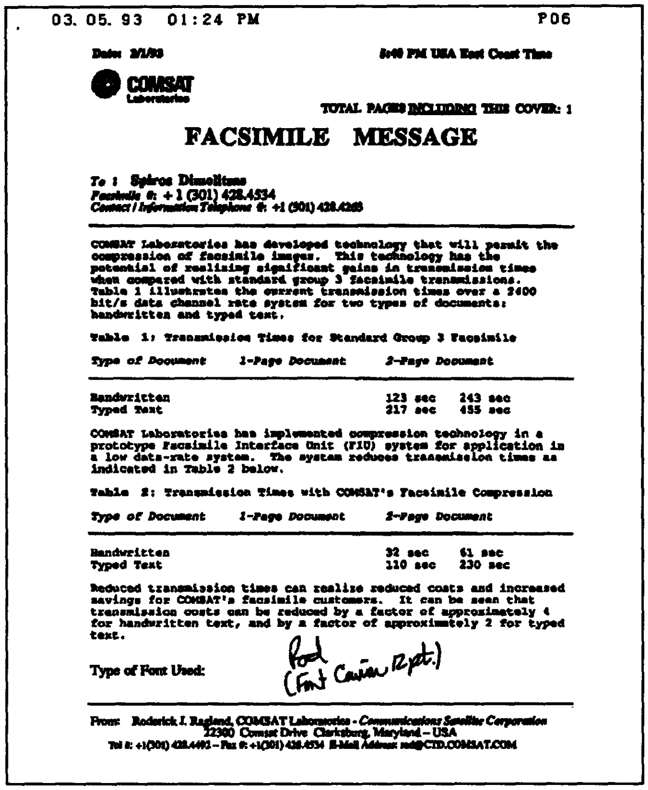
Type of Document	*1-Page Document*	*2-Page Document*
Handwritten	32 sec	61 sec
Typed Text	118 sec	230 sec

Reduced transmission times can realize reduced costs and increased savings for COMSAT's facsimile customers. It can be seen that transmission costs can be reduced by a factor of approximately 4 for handwritten text, and by a factor of approximately 2 for typed text. If the examples were to be examined, it can also be seen that there is a compromise taking place to permit this drastic reduction in transmission time and cost to be achieved: the quality of the resulting image is somewhat degraded, but the document remains intelligible.

Type of Font Used: Bold (Font Courier 10 pt)

From: Roderick J. Ragland, COMSAT Laboratories - *Communications Satellite Corporation*
22300 Comsat Drive Clarksburg, Maryland – USA
Tel #: +1(301) 428.4492 – Fax #: +1(301) 428.4534 E-Mail Address: rod@CTD.COMSAT.COM

Figure D-2: Training set document #2



03. 05. 93 01:24 PM P06

Date: 3/1/93 5:40 PM USA East Coast Time

COMSAT Laboratories

TOTAL PAGES INCLUDING THIS COVER: 1

FACSIMILE MESSAGE

To : Spiros Dimolitsas
Facsimile #: + 1 (301) 428.4534
Contact / Information Telephone #: +1 (301) 428.4265

COMSAT Laboratories has developed technology that will permit the compression of facsimile images. This technology has the potential of realizing significant gains in transmission times when compared with standard group 3 facsimile transmissions. Table 1 illustrates the current transmission times over a 2400 bit/s data channel rate system for two types of documents: handwritten and typed text.

Table 1: Transmission Times for Standard Group 3 Facsimile

Type of Document	*1-Page Document*	*2-Page Document*
Handwritten	123 sec	243 sec
Typed Text	217 sec	455 sec

COMSAT Laboratories has implemented compression technology in a prototype Facsimile Interface Unit (FIU) system for application in a low data-rate system. The system reduces transmission times as indicated in Table 2 below.

Table 2: Transmission Times with COMSAT's Facsimile Compression

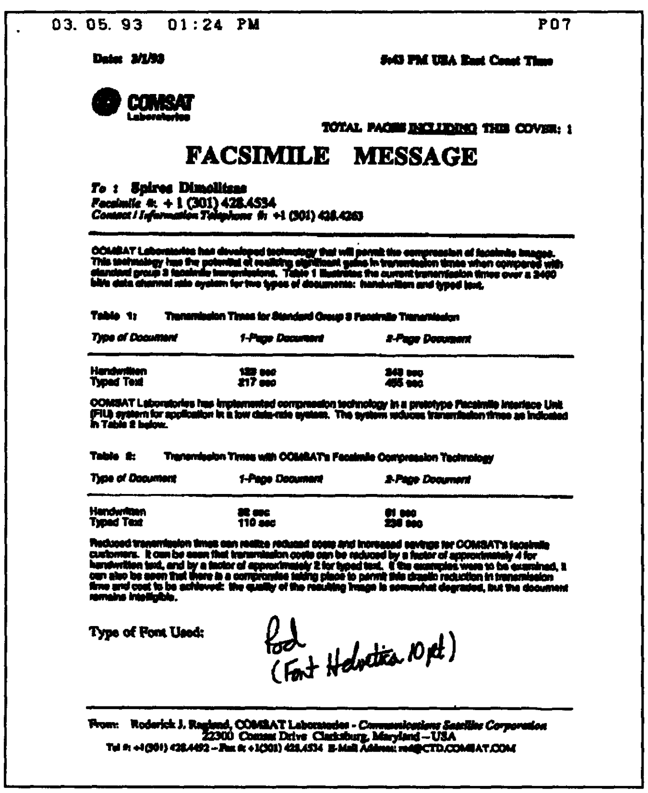
Type of Document	*1-Page Document*	*2-Page Document*
Handwritten	32 sec	61 sec
Typed Text	110 sec	230 sec

Reduced transmission times can realize reduced costs and increased savings for COMSAT's facsimile customers. It can be seen that transmission costs can be reduced by a factor of approximately 4 for handwritten text, and by a factor of approximately 2 for typed text.

Type of Font Used: Pod (Font Courier 12pt.)

From: Roderick J. Ragland, COMSAT Laboratories - *Communications Satellite Corporation*
22300 Comsat Drive Clarksburg, Maryland – USA
Tel #: +1(301) 428.4492 – Fax #: +1(301) 428.4534 E-Mail Address: rod@CTD.COMSAT.COM

Figure D-3: Training set document #3



03. 05. 93 01:24 PM P07

Date: 3/1/93 5:43 PM USA East Coast Time

COMSAT Laboratories

TOTAL PAGES INCLUDING THIS COVER: 1

FACSIMILE MESSAGE

To : Spiros Dimolitsas
Facsimile #: + 1 (301) 428.4534
Contact / Information Telephone #: +1 (301) 428.4263

COMSAT Laboratories has developed technology that will permit the compression of facsimile images. This technology has the potential of realizing significant gains in transmission times when compared with standard group 3 facsimile transmissions. Table 1 illustrates the current transmission times over a 2400 bit/s data channel rate system for two types of documents: handwritten and typed text.

Table 1: Transmission Times for Standard Group 3 Facsimile Transmission

Type of Document	*1-Page Document*	*2-Page Document*
Handwritten	123 sec	243 sec
Typed Text	217 sec	455 sec

COMSAT Laboratories has implemented compression technology in a prototype Facsimile Interface Unit (FIU) system for application in a low data-rate system. The system reduces transmission times as indicated in Table 2 below.

Table 2: Transmission Times with COMSAT's Facsimile Compression Technology

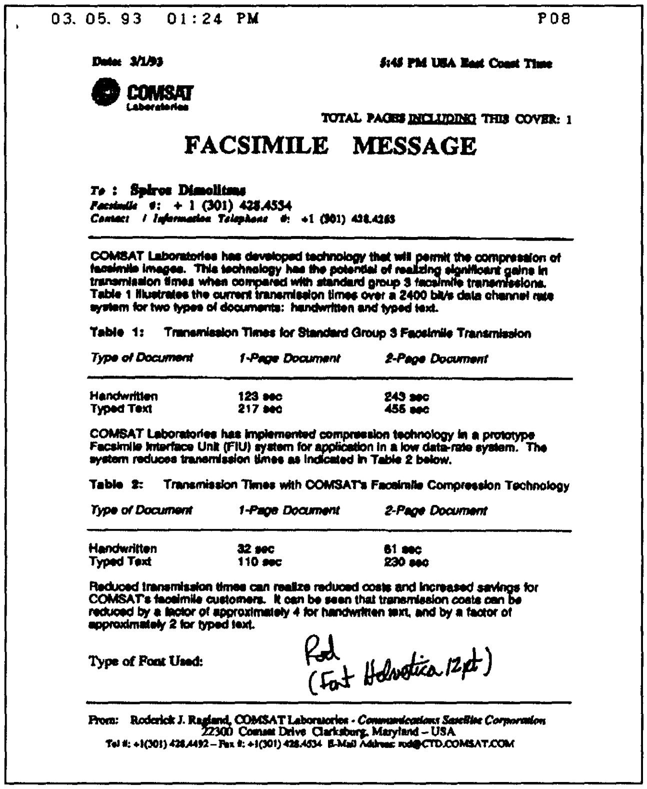
Type of Document	*1-Page Document*	*2-Page Document*
Handwritten	32 sec	61 sec
Typed Text	110 sec	230 sec

Reduced transmission times can realize reduced costs and increased savings for COMSAT's facsimile customers. It can be seen that transmission costs can be reduced by a factor of approximately 4 for handwritten text, and by a factor of approximately 2 for typed text. If the examples were to be examined, it can also be seen that there is a compromise taking place to permit this drastic reduction in transmission time and cost to be achieved: the quality of the resulting image is somewhat degraded, but the document remains intelligible.

Type of Font Used: Pod (Font Helvetica 10pt)

From: Roderick J. Ragland, COMSAT Laboratories - *Communications Satellite Corporation*
22300 Comsat Drive Clarksburg, Maryland – USA
Tel #: +1(301) 428.4492 – Fax #: +1(301) 428.4534 E-Mail Address: rod@CTD.COMSAT.COM

Figure D-4: Training set document #4



03.05.93 01:24 PM P08

Date: 3/1/93 5:45 PM USA East Coast Time

COMSAT Laboratories

TOTAL PAGES INCLUDING THIS COVER: 1

FACSIMILE MESSAGE

To : Spiros Dimolitsas
Facsimile #: + 1 (301) 428.4554
Contact / Information Telephone #: +1 (301) 428.4263

COMSAT Laboratories has developed technology that will permit the compression of facsimile images. This technology has the potential of realizing significant gains in transmission times when compared with standard group 3 facsimile transmissions. Table 1 illustrates the current transmission times over a 2400 bit/s data channel rate system for two types of documents: handwritten and typed text.

Table 1: Transmission Times for Standard Group 3 Facsimile Transmission

Type of Document	*1-Page Document*	*2-Page Document*
Handwritten	123 sec	243 sec
Typed Text	217 sec	455 sec

COMSAT Laboratories has implemented compression technology in a prototype Facsimile Interface Unit (FIU) system for application in a low data-rate system. The system reduces transmission times as indicated in Table 2 below.

Table 2: Transmission Times with COMSAT's Facsimile Compression Technology

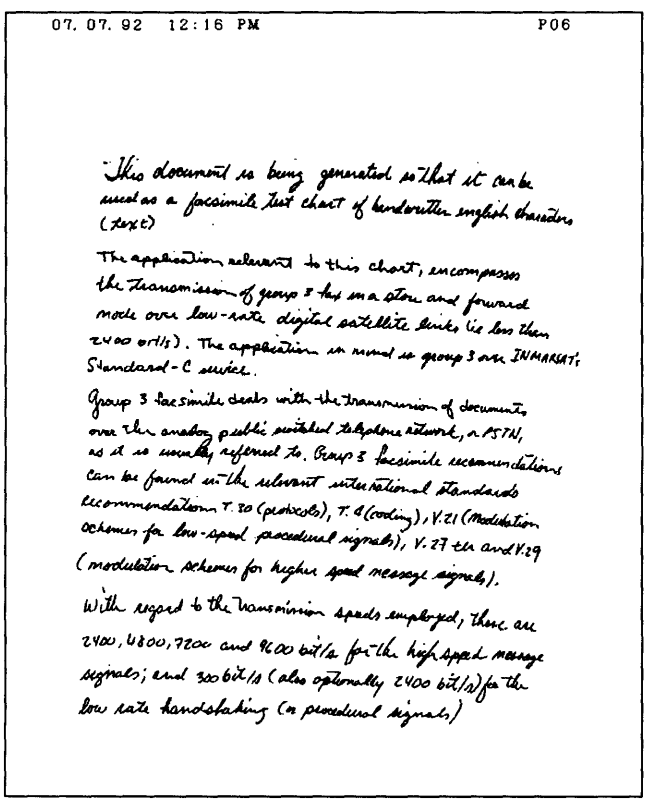
Type of Document	*1-Page Document*	*2-Page Document*
Handwritten	32 sec	61 sec
Typed Text	110 sec	230 sec

Reduced transmission times can realize reduced costs and increased savings for COMSAT's facsimile customers. It can be seen that transmission costs can be reduced by a factor of approximately 4 for handwritten text, and by a factor of approximately 2 for typed text.

Type of Font Used: Rod (Font Helvetica 12pt)

From: Roderick J. Ragland, COMSAT Laboratories - *Communications Satellite Corporation*
22300 Comsat Drive Clarksburg, Maryland – USA
Tel #: +1(301) 428.4492 – Fax #: +1(301) 428.4534 E-Mail Address: rod@CTD.COMSAT.COM

Figure D-5: Training set document #5



07.07.92 12:16 PM P06

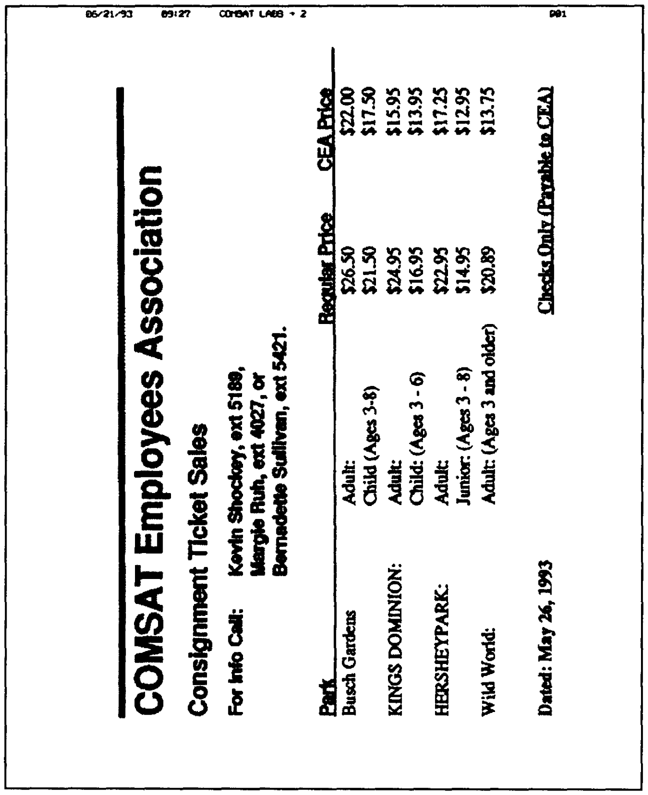
This document is being generated so that it can be used as a facsimile test chart of handwritten english characters (text)

The application relevant to this chart, encompasses the transmission of group 3 fax in a store and forward mode over low-rate digital satellite links (ie less than 2400 bit/s). The application in mind is group 3 over INMARSAT's Standard-C service.

Group 3 facsimile deals with the transmission of documents over the analog public switched telephone network, or PSTN, as it is usually referred to. Group 3 facsimile recommendations can be found in the relevant international standards recommendations T.30 (protocols), T.4 (coding), V.21 (Modulation schemes for low-speed procedural signals), V.27 ter and V.29 (modulation schemes for higher speed message signals).

With regard to the transmission speeds employed, there are 2400, 4800, 7200 and 9600 bit/s for the high speed message signals; and 300 bit/s (also optionally 2400 bit/s) for the low rate handshaking (or procedural signals)

Figure D-6: Training set document #6


05/21/93 09:27 COMSAT LABS → 2 001

COMSAT Employees Association

Consignment Ticket Sales

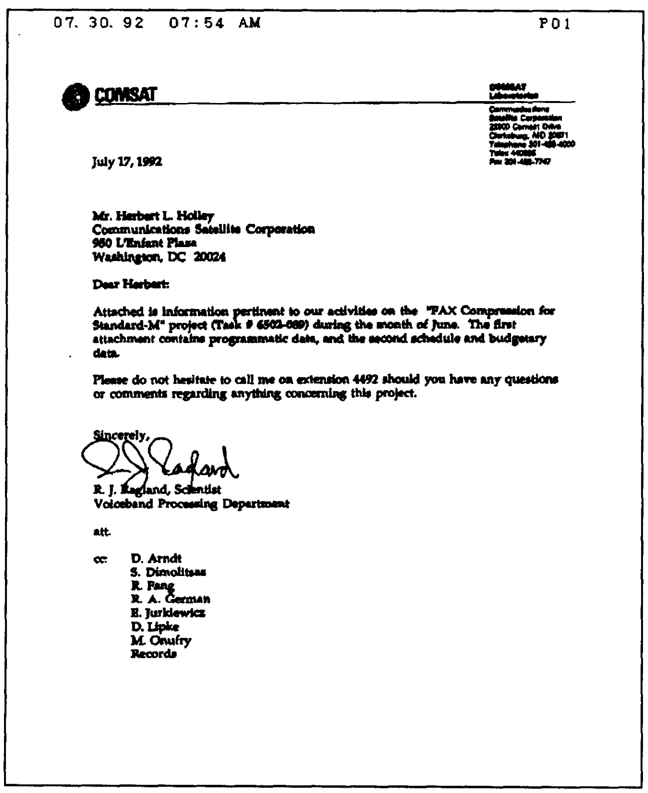
For Info Call: Kevin Shockey, ext 5189, Margie Ruth, ext 4027, or Bernadette Sullivan, ext 5421.

Park		Regular Price	CEA Price
Busch Gardens	Adult:	$26.50	$22.00
	Child (Ages 3-8)	$21.50	$17.50
KINGS DOMINION:	Adult:	$24.95	$15.95
	Child: (Ages 3 - 6)	$16.95	$13.95
HERSHEYPARK:	Adult:	$22.95	$17.25
	Junior: (Ages 3 - 8)	$14.95	$12.95
Wild World:	Adult: (Ages 3 and older)	$20.89	$13.75

Dated: May 26, 1993

Checks Only (Payable to CEA)

Figure D-7: Training set document #7


07. 30. 92 07:54 AM P01

COMSAT

July 17, 1992

Mr. Herbert L. Holley
Communications Satellite Corporation
950 L'Enfant Plaza
Washington, DC 20024

Dear Herbert:

Attached is information pertinent to our activities on the "FAX Compression for Standard-M" project (Task # 6502-089) during the month of June. The first attachment contains programmatic data, and the second schedule and budgetary data.

Please do not hesitate to call me on extension 4492 should you have any questions or comments regarding anything concerning this project.

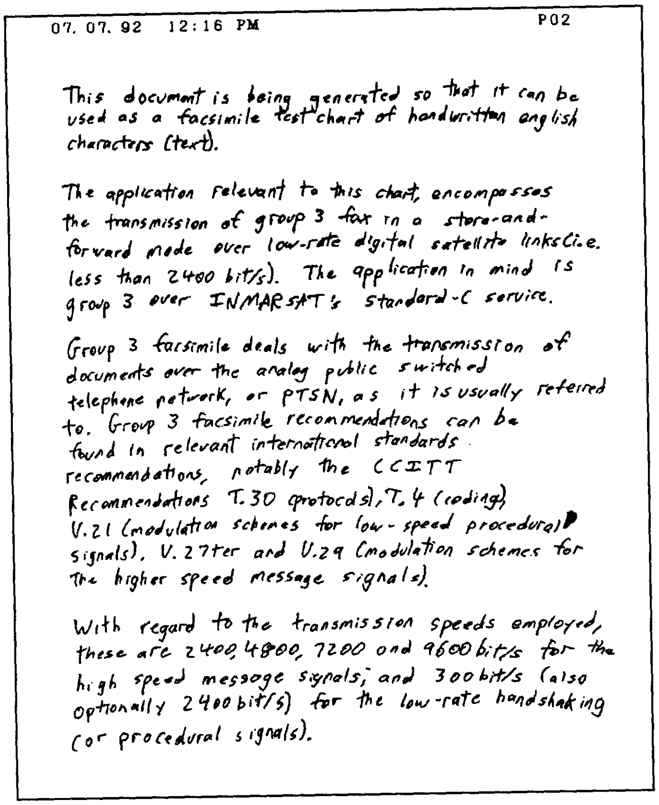
Sincerely,

R. J. Ragland, Scientist
Voiceband Processing Department

att.

cc: D. Arndt
S. Dimolitsas
R. Fang
R. A. German
E. Jurkiewicz
D. Lipke
M. Onufry
Records

Figure D-8: Training set document #8



07. 07. 92 12:16 PM P02

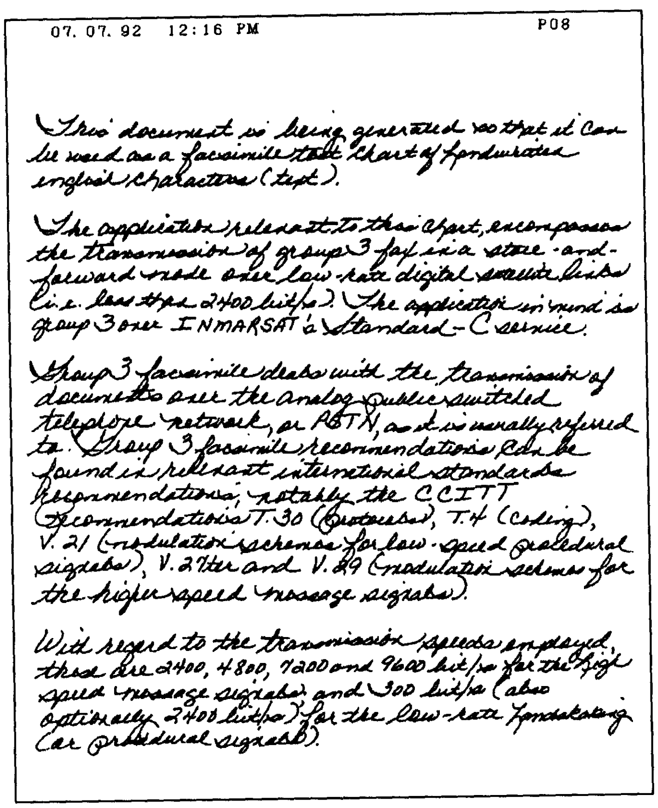
This document is being generated so that it can be used as a facsimile test chart of handwritten english characters (text).

The application relevant to this chart, encompasses the transmission of group 3 fax in a store-and-forward mode over low-rate digital satellite links (i.e. less than 2400 bit/s). The application in mind is group 3 over INMARSAT's Standard-C service.

Group 3 facsimile deals with the transmission of documents over the analog public switched telephone network, or PTSN, as it is usually referred to. Group 3 facsimile recommendations can be found in relevant international standards. recommendations, notably the CCITT Recommendations T.30 (protocols), T.4 (coding), V.21 (modulation schemes for low-speed procedural signals), V.27ter and V.29 (modulation schemes for the higher speed message signals).

With regard to the transmission speeds employed, these are 2400, 4800, 7200 and 9600 bit/s for the high speed message signals, and 300 bit/s (also optionally 2400 bit/s) for the low-rate handshaking (or procedural signals).

Figure D-9: Training set document #9



07. 07. 92 12:16 PM P08

This document is being generated so that it can be used as a facsimile test chart of handwritten english characters (text).

The application relevant to this chart, encompasses the transmission of group 3 fax in a store-and-forward mode over low-rate digital satellite links (i.e. less than 2400 bit/s). The application in mind is group 3 over INMARSAT's Standard-C service.

Group 3 facsimile deals with the transmission of documents over the analog public switched telephone network, or PSTN, as it is usually referred to. Group 3 facsimile recommendations can be found in relevant international standards recommendations, notably the CCITT Recommendations T.30 (protocols), T.4 (coding), V.21 (modulation schemes for low-speed procedural signals), V.27ter and V.29 (modulation schemes for the higher speed message signals).

With regard to the transmission speeds employed, these are 2400, 4800, 7200 and 9600 bit/s for the high speed message signals and 300 bit/s (also optionally 2400 bit/s) for the low-rate handshaking (or procedural signals).

Figure D-10: Training set document #10

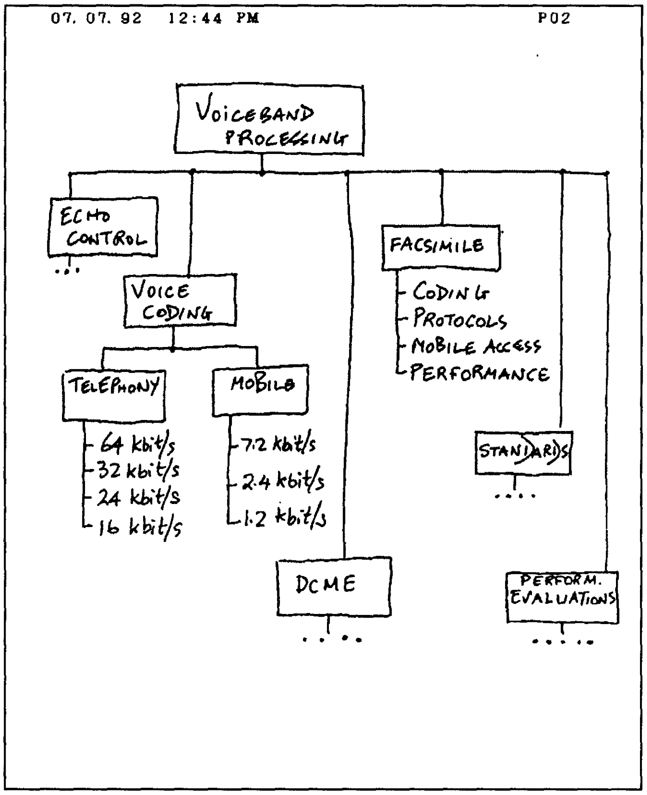
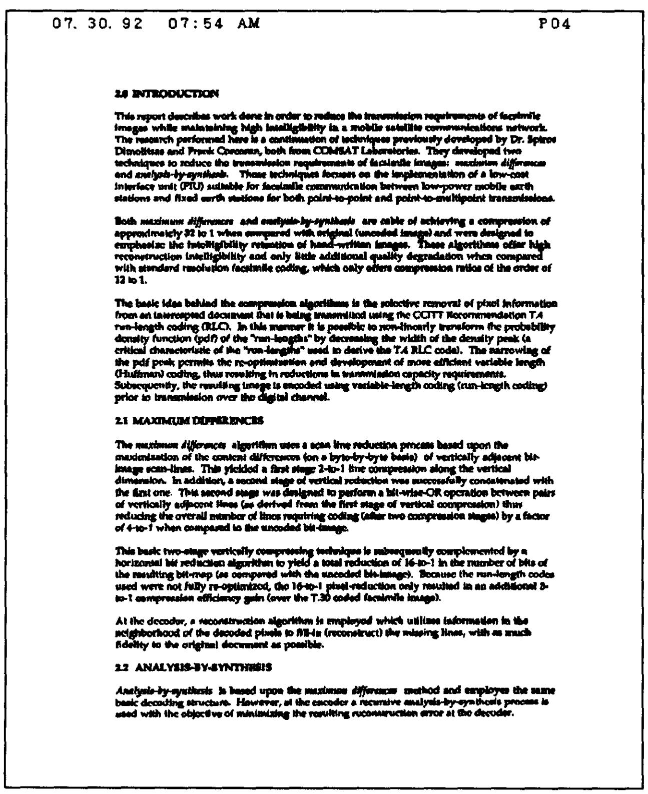


Figure D-11: Training set document #11



07. 30. 92 07:54 AM P04

2.0 INTRODUCTION

This report describes work done in order to reduce the transmission requirements of facsimile images while maintaining high intelligibility in a mobile satellite communications network. The research performed here is a continuation of techniques previously developed by Dr. Spiros Dimolitsas and Frank Corcoran, both from COMSAT Laboratories. They developed two techniques to reduce the transmission requirements of facsimile images: *maximum differences* and *analysis-by-synthesis*. These techniques focuses on the implementation of a low-cost interface unit (FIU) suitable for facsimile communication between low-power mobile earth stations and fixed earth stations for both point-to-point and point-to-multipoint transmissions.

Both *maximum differences* and *analysis-by-synthesis* are cable of achieving a compression of approximately 32 to 1 when compared with original (uncoded image) and were designed to emphasize the intelligibility retention of hand-written images. These algorithms offer high reconstruction intelligibility and only little additional quality degradation when compared with standard resolution facsimile coding, which only offers compression ratios of the order of 12 to 1.

The basic idea behind the compression algorithms is the selective removal of pixel information from an intercepted document that is being transmitted using the CCITT Recommendation T.4 run-length coding (RLC). In this manner it is possible to non-linearly transform the probability density function (pdf) of the "run-lengths" by decreasing the width of the density peak (a critical characteristic of the "run-lengths" used to derive the T.4 RLC code). The narrowing of the pdf peak permits the re-optimization and development of more efficient variable length (Huffman) coding, thus resulting in reductions in transmission capacity requirements. Subsequently, the resulting image is encoded using variable-length coding (run-length coding) prior to transmission over the digital channel.

2.1 MAXIMUM DIFFERENCES

The *maximum differences* algorithm uses a scan line reduction process based upon the maximization of the content differences (on a byte-by-byte basis) of vertically adjacent bit-image scan-lines. This yielded a first stage 2-to-1 line compression along the vertical dimension. In addition, a second stage of vertical reduction was successfully concatenated with the first one. This second stage was designed to perform a bit-wise-OR operation between pairs of vertically adjacent lines (as derived from the first stage of vertical compression) thus reducing the overall number of lines requiring coding (after two compression stages) by a factor of 4-to-1 when compared to the uncoded bit-image.

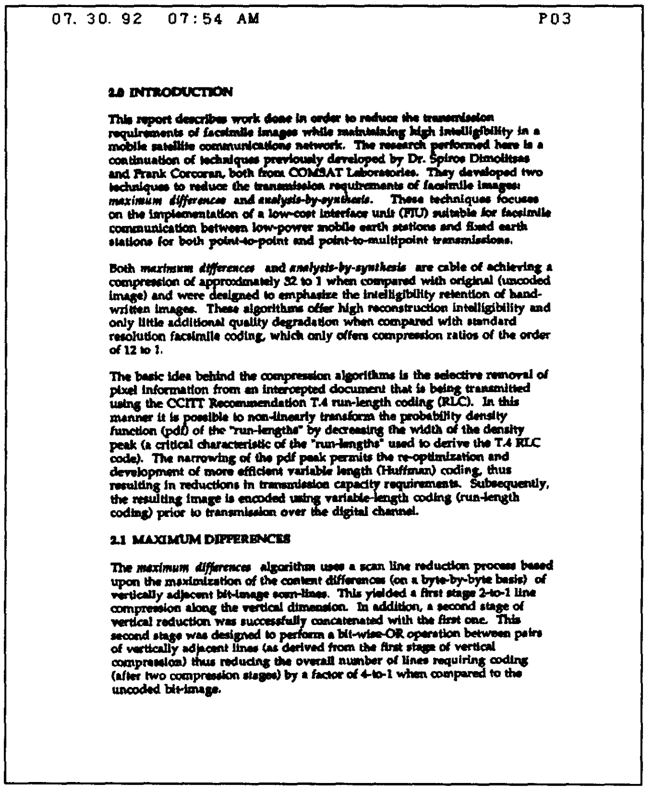
This basic two-stage vertically compressing technique is subsequently complemented by a horizontal bit reduction algorithm to yield a total reduction of 16-to-1 in the number of bits of the resulting bit-map (as compared with the uncoded bit-image). Because the run-length codes used were not fully re-optimized, the 16-to-1 pixel-reduction only resulted in an additional 3-to-1 compression efficiency gain (over the T.30 coded facsimile image).

At the decoder, a reconstruction algorithm is employed which utilizes information in the neighborhood of the decoded pixels to fill-in (reconstruct) the missing lines, with as much fidelity to the original document as possible.

2.2 ANALYSIS-BY-SYNTHESIS

Analysis-by-synthesis is based upon the *maximum differences* method and employes the same basic decoding structure. However, at the encoder a recursive analysis-by-synthesis process is used with the objective of minimizing the resulting reconstruction error at the decoder.

Figure D-12: Training set document #12



07. 30. 92 07:54 AM P03

2.0 INTRODUCTION

This report describes work done in order to reduce the transmission requirements of facsimile images while maintaining high intelligibility in a mobile satellite communications network. The research performed here is a continuation of techniques previously developed by Dr. Spiros Dimolitsas and Frank Corcoran, both from COMSAT Laboratories. They developed two techniques to reduce the transmission requirements of facsimile images: *maximum differences* and *analysis-by-synthesis*. These techniques focuses on the implementation of a low-cost interface unit (FIU) suitable for facsimile communication between low-power mobile earth stations and fixed earth stations for both point-to-point and point-to-multipoint transmissions.

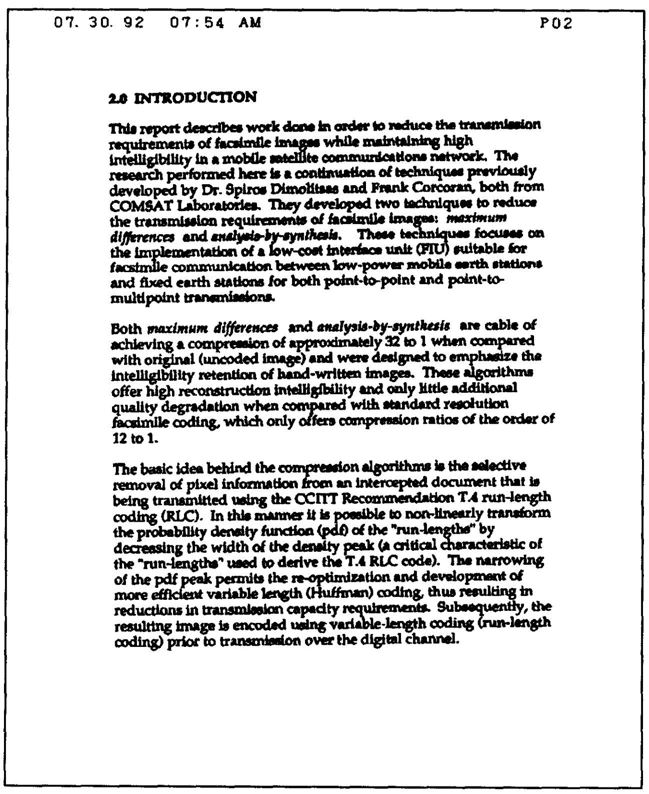
Both *maximum differences* and *analysis-by-synthesis* are cable of achieving a compression of approximately 32 to 1 when compared with original (uncoded image) and were designed to emphasize the intelligibility retention of hand-written images. These algorithms offer high reconstruction intelligibility and only little additional quality degradation when compared with standard resolution facsimile coding, which only offers compression ratios of the order of 12 to 1.

The basic idea behind the compression algorithms is the selective removal of pixel information from an intercepted document that is being transmitted using the CCITT Recommendation T.4 run-length coding (RLC). In this manner it is possible to non-linearly transform the probability density function (pdf) of the "run-lengths" by decreasing the width of the density peak (a critical characteristic of the "run-lengths" used to derive the T.4 RLC code). The narrowing of the pdf peak permits the re-optimization and development of more efficient variable length (Huffman) coding, thus resulting in reductions in transmission capacity requirements. Subsequently, the resulting image is encoded using variable-length coding (run-length coding) prior to transmission over the digital channel.

2.1 MAXIMUM DIFFERENCES

The *maximum differences* algorithm uses a scan line reduction process based upon the maximization of the content differences (on a byte-by-byte basis) of vertically adjacent bit-image scan-lines. This yielded a first stage 2-to-1 line compression along the vertical dimension. In addition, a second stage of vertical reduction was successfully concatenated with the first one. This second stage was designed to perform a bit-wise-OR operation between pairs of vertically adjacent lines (as derived from the first stage of vertical compression) thus reducing the overall number of lines requiring coding (after two compression stages) by a factor of 4-to-1 when compared to the uncoded bit-image.

Figure D-13: Training set document #13



07. 30. 92 07:54 AM P02

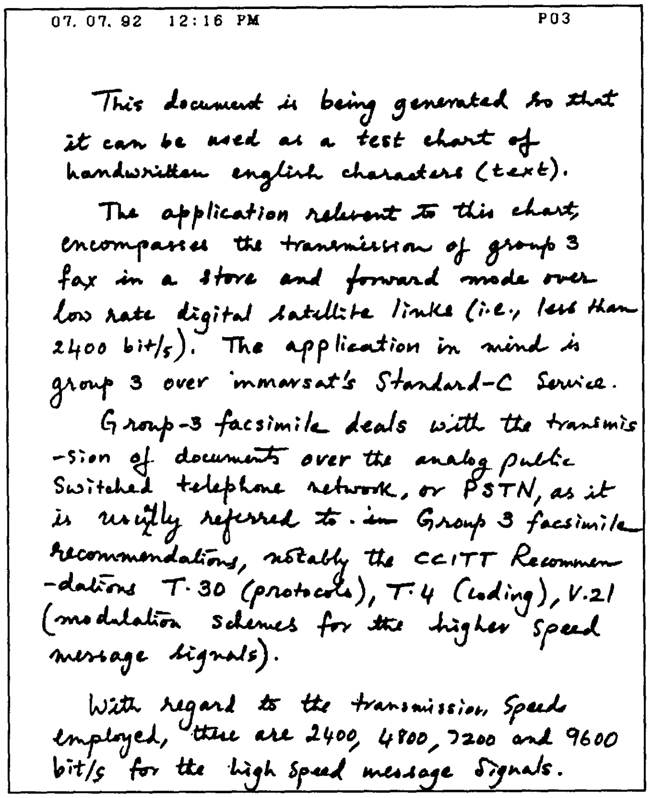
2.0 INTRODUCTION

This report describes work done in order to reduce the transmission requirements of facsimile images while maintaining high intelligibility in a mobile satellite communications network. The research performed here is a continuation of techniques previously developed by Dr. Spiros Dimolitsas and Frank Corcoran, both from COMSAT Laboratories. They developed two techniques to reduce the transmission requirements of facsimile images: *maximum differences* and *analysis-by-synthesis*. These techniques focuses on the implementation of a low-cost interface unit (FIU) suitable for facsimile communication between low-power mobile earth stations and fixed earth stations for both point-to-point and point-to-multipoint transmissions.

Both *maximum differences* and *analysis-by-synthesis* are cable of achieving a compression of approximately 32 to 1 when compared with original (uncoded image) and were designed to emphasize the intelligibility retention of hand-written images. These algorithms offer high reconstruction intelligibility and only little additional quality degradation when compared with standard resolution facsimile coding, which only offers compression ratios of the order of 12 to 1.

The basic idea behind the compression algorithms is the selective removal of pixel information from an intercepted document that is being transmitted using the CCITT Recommendation T.4 run-length coding (RLC). In this manner it is possible to non-linearly transform the probability density function (pdf) of the "run-lengths" by decreasing the width of the density peak (a critical characteristic of the "run-lengths" used to derive the T.4 RLC code). The narrowing of the pdf peak permits the re-optimization and development of more efficient variable length (Huffman) coding, thus resulting in reductions in transmission capacity requirements. Subsequently, the resulting image is encoded using variable-length coding (run-length coding) prior to transmission over the digital channel.

Figure D-14: Training set document #14



07. 07. 92 12:16 PM P03

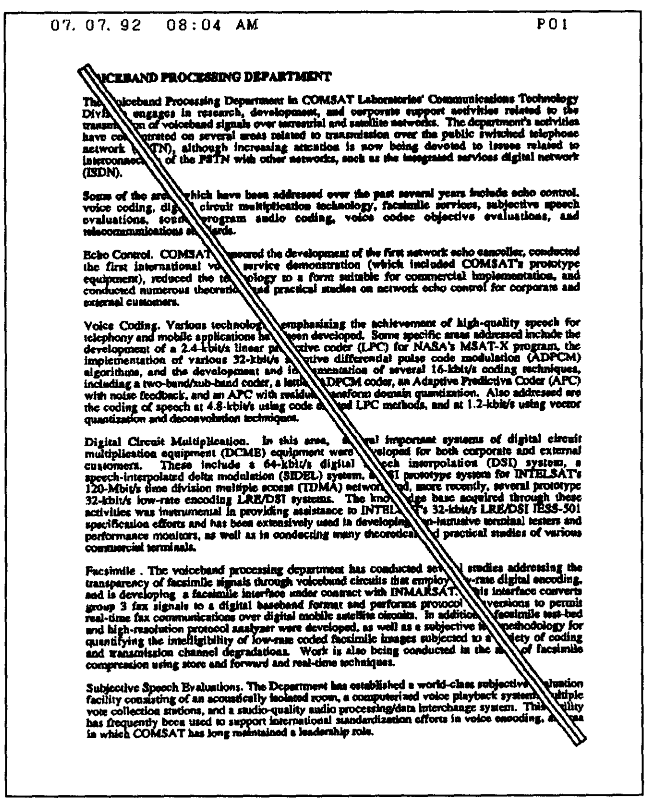
This document is being generated so that it can be used as a test chart of handwritten english characters (text).

The application relevant to this chart, encompasses the transmission of group 3 fax in a store and forward mode over low rate digital satellite links (i.e., less than 2400 bit/s). The application in mind is group 3 over inmarsat's Standard-C Service.

Group-3 facsimile deals with the transmission of documents over the analog public Switched telephone network, or PSTN, as it is usually referred to in Group 3 facsimile recommendations, notably the CCITT Recommendations T.30 (protocols), T.4 (coding), V.21 (modulation schemes for the higher speed message signals).

With regard to the transmission speeds employed, there are 2400, 4800, 7200 and 9600 bit/s for the high speed message signals.

Figure D-15: Training set document #15



07. 07. 92 08:04 AM P01

ICEBAND PROCESSING DEPARTMENT

Th oiceband Processing Department in COMSAT Laboratories' Communications Technology Divi engages in research, development, and corporate support activities related to the transmi on of voiceband signals over terrestrial and satellite networks. The department's activities have co ntrated on several areas related to transmission over the public switched telephone network TN), although increasing attention is now being devoted to issues related to interconnec of the PSTN with other networks, such as the integrated services digital network (ISDN).

Some of the are which have been addressed over the past several years include echo control, voice coding, dig circuit multiplication technology, facsimile services, subjective speech evaluations, sou rogram audio coding, voice codec objective evaluations, and telecommunications s dards.

Echo Control. COMSAT sored the development of the first network echo canceller, conducted the first international vo service demonstration (which included COMSAT's prototype equipment), reduced the te ology to a form suitable for commercial implementation, and conducted numerous theoreti and practical studies on network echo control for corporate and external customers.

Voice Coding. Various technolog emphasizing the achievement of high-quality speech for telephony and mobile applications ha een developed. Some specific areas addressed include the development of a 2.4-kbit/s linear pr ctive coder (LPC) for NASA's MSAT-X program, the implementation of various 32-kbit/s tive differential pulse code modulation (ADPCM) algorithms, and the development and i mentation of several 16-kbit/s coding techniques, including a two-band/sub-band coder, a latti ADPCM coder, an Adaptive Predictive Coder (APC) with noise feedback, and an APC with residu ansform domain quantization. Also addressed are the coding of speech at 4.8-kbit/s using code ed LPC methods, and at 1.2-kbit/s using vector quantization and deconvolution techniques.

Digital Circuit Multiplication. In this area, al important systems of digital circuit multiplication equipment (DCME) equipment were veloped for both corporate and external customers. These include a 64-kbit/s digital ech interpolation (DSI) system, a speech-interpolated delta modulation (SIDEL) system, SI prototype system for INTELSAT's 120-Mbit/s time division multiple access (TDMA) networ d, more recently, several prototype 32-kbit/s low-rate encoding LRE/DSI systems. The kno dge base acquired through these activities was instrumental in providing assistance to INTEL T's 32-kbit/s LRE/DSI IESS-501 specification efforts and has been extensively used in developing n-intrusive terminal testers and performance monitors, as well as in conducting many theoretica d practical studies of various commercial terminals.

Facsimile . The voiceband processing department has conducted se l studies addressing the transparency of facsimile signals through voiceband circuits that employ -rate digital encoding, and is developing a facsimile interface under contract with INMARSAT. his interface converts group 3 fax signals to a digital baseband format and performs protocol versions to permit real-time fax communications over digital mobile satellite circuits. In addition facsimile test-bed and high-resolution protocol analyzer were developed, as well as a subjective methodology for quantifying the intelligibility of low-rate coded facsimile images subjected to a iety of coding and transmission channel degradations. Work is also being conducted in the of facsimile compression using store and forward and real-time techniques.

Subjective Speech Evaluations. The Department has established a world-class subjective aluation facility consisting of an acoustically isolated room, a computerized voice playback system ultiple vote collection stations, and a studio-quality audio processing/data interchange system. Thi ility has frequently been used to support international standardization efforts in voice encoding, as in which COMSAT has long maintained a leadership role.

Figure D-16: Training set document #16

Appendix E

CAFC Software Implementation

This appendix contains the C source code for the software implementation of Content-Adaptive Facsimile Coding. All of the components of CAFC are supported except for Dithered Bitmap Detection and Direct Coding. The roles of each module in performing the various stages of the CAFC algorithm are described in Table E.1. The programs that perform the encoding and decoding process images in the Intel PCX format. Routines that greatly simplify the reading and writing these files are contained in the modules described in Table E.2.

These programs were used to perform most of the parameter optimizations and to generate the final results in Chap. 8 and Appendix B. Table E.3 lists the programs that were used to process the training set images in Appendices C and D to generate statistics for feature selection and Optimized 2D Run-Length Coding.

FILES	FUNCTION	DESCRIPTION
`CAFC.h`	CAFC Parameters	This include file specifies values for all CAFC parameters (minimum/maximum symbol size, symbol isolation method, features to use, etc.)
`CAFC_encode.c`	CAFC Encoding	Encodes an image in the PCX file format using Content-Adaptive Facsimile Coding (CAFC), producing a binary output file and a residue image PCX file.
`CAFC_decode.c`	CAFC Decoding	Decodes an image that was encoded with **`CAFC_encode`** ($CAFC^{-1}$). Produces a reconstructed image PCX file and a residue image PCX file.
`match.c` `match.h`	Symbol Matching	Contains routines to determine if two symbols "match" using the feature matching and feature extraction algorithms.
`features.c` `features.h`	Feature Extraction	Contains functions to compute the features of a symbol. Maintains a global set of features and provides a mechanism to extract them from a symbol.
`library.c` `library.h`	Symbol Library Management	Manages a library of symbols, allowing updates and searches to be performed. The library also contains information necessary for matching and arithmetic coding.

`symbol_filling.c`
`symbol_tracing.c`
`symbol_windowing.c`
	Symbol Library Management	Performs symbol isolation on a buffered PCX image. Each of the three approaches is implemented separately – *symbol filling*, *symbol tracing*, and *symbol windowing.*

`symbol.c` `symbol.h`	Symbol Manipulation	Contains declarations and procedures to facilitate the manipulation of symbols. A SYMBOL structure is defined and routines to create and free symbols are provided.
`AC.c` `AC.h`	Arithmetic Coding/Decoding	This module performs general arithmetic encoding and decoding. It contains routines for creating and updating source models and for entropy coding/decoding a stream of elements (symbols) using these models.

Table E.1: Summary of CAFC software modules.

PCX_buffer.c PCX_buffer.h	PCX Image Buffering	This module provides a buffered interface to PCX image files. A multiple scan-line portion of the page is maintained at all times. This buffer is simply "scrolled" up to automatically read or write a scan-line. This interface is useful for real-time algorithms that need to access several adjacent scan-lines at a time.
PCX_util.c PCX_util.h	PCX File Format Interface	This module provides a straightforward interface for line-oriented reading and writing of PCX image files.

Table E.2: Summary of PCX file format modules.

2Drl_stats.c	2D Run-Length Statistics	This program reads a set of PCX images (a training set) and determines their 2D run-length statistics. Produces output data files that are used by CAFC_encode and CAFC_decode to actually perform the coding and decoding.
feature_stats.c	Feature Statistics	Analyzes a large set of test images containing typed text in a variety of fonts. Generates statistics on the effectiveness of each feature at correctly matching symbols. Produces output files which can then be used to select the best subset of features.

Table E.3: Summary of statistics gathering programs.

E.1 Source Code – CAFC Parameters

File `CAFC.h`:

```
/*****************************************************************************
Name:     CAFC.h
Purpose:  This include file contains constants and parameters used by
          the CAFC programs.

Last Modified on 5/4/94
*****************************************************************************/

/* facimile page resolution, standard or fine */
#define PAGE_RESOLUTION     FINE_MODE

/* restrictions on symbol size */
#define MAX_SYMBOL_HEIGHT  40  /* maximum height of a symbol in pixels */
#define MAX_SYMBOL_WIDTH   60  /* maximum width of a symbol in pixels */
#define MIN_SYMBOL_HEIGHT  2   /* minimum height of a symbol in pixels */
#define MIN_SYMBOL_WIDTH   3   /* minimum width of a symbol in pixels */

/* number of scanlines in buffer, one more than the maximum symbol height */
#define NLINES     (MAX_SYMBOL_HEIGHT + 1)

/* symbol isolation techniques */
#define SYMBOL_FILLING      1
#define SYMBOL_TRACING      2
#define SYMBOL_WINDOWING    3

#define SYMBOL_ISOLATION    SYMBOL_WINDOWING /* selected technique */

/* feature matching */
#define NFEATURES 5 /* total number of features defined */

#define FEATURES {width, black_pels, vert_run_lengths, white_pels, height}
#define FEATURE_NAMES {"width","black pels","vert.  runs","white pels","height"}
#define FEATURE_MATCH_THRESHOLD {2, 30, 4, 53, 4}
#define FEATURE_EFFECTIVENESS {0.30, 0.34, 0.38, 0.48, 0.5}

/* template matching */
#define TEMPLATE_MATCH_THRESHOLD 0.82
#define TEMPLATE_MAXIMUM_SHIFT_X 2
#define TEMPLATE_MAXIMUM_SHIFT_Y 2

/* 2D run-length coding */
#define RL_STATS_WEIGHT  0.25 /* weight of initial run-length statistics */
/*************************************************************************/

/* Name of file containing 2D run-length statistics, "" for NONE. */
#define RL_FILENAME  "/u/nht/data/rl_stats2D.dat"
```

E.2 Source Code – CAFC Encoder

File `CAFC_encode.c`:

```
/*****************************************************************************
Name:     CAFC_encode.c
Purpose:  Encodes a bi-level image using Content-Adaptive Bi-Level (Facsimile)
          Coding.  The source image is assumed to be in the PCX file format.
          The encoded image is stored in a binary file.  The residue image
          can be optionally created and stored in a PCX file.
```

```
Usage:     CAFC_encode PCXsource CAFCdest [ PCXresidue ]

           PCXsource  -> filename of the source image (in PCX format)
           CAFCdest   -> filename of the destination CAFC-encoded image
           PCXresidue -> filename of residue image (optional, PCX format)

           To perform the 2D run-length encoding portion of the algorithm,
           run-length statistics are read from the data file named in
           CAFC.h if specified.

Notes:     All CAFC parameters are specified in the file CAFC.h.

Last modified on 5/4/94
***************************************************************************/
#include<stdio.h>
#include<stdlib.h>

/* CAFC include files. */
#include"CAFC.h"
#include"PCX_util.h"
#include"PCX_buffer.h"
#include"symbol.h"
#include"library.h"
#include"features.h"
#include"match.h"
#include"AC.h"

/* Preprocessor code to select to correct symbol filling functions. */
#if SYMBOL_ISOLATION == SYMBOL_FILLING
# define isolate_symbol symbol_filling_isolate
# define remove_symbol symbol_filling_remove
# define isolate_scroll symbol_filling_scroll
#elif SYMBOL_ISOLATION == SYMBOL_TRACING
# define isolate_symbol symbol_tracing_isolate
# define remove_symbol symbol_tracing_remove
# define isolate_scroll symbol_tracing_scroll
#elif SYMBOL_ISOLATION == SYMBOL_WINDOWING
# define isolate_symbol symbol_windowing_isolate
# define remove_symbol symbol_windowing_remove
# define isolate_scroll symbol_windowing_scroll
#endif

/* external symbol-isolation routines */
SYMBOL *isolate_symbol();
void remove_symbol();
void isolate_scroll();

/* buffer containing source scan-lines */
byte **source_buffer;

/* destination compressed file (CAFC) */
FILE *CAFC_dest;

/* residue output file */
PCX_FILE *residue;

/* file containing 2D run-length statistics */
FILE *rl;

/* one line of residue */
byte *residue_line;

/* previous line of residue  */
byte *residue_prev_line;

/* one line of differences between vertically adjacent pixels.          */
byte *diff_line;

/* width of image in pixels */
int maxX;
```

```
/* the symbol library */
LIBRARY *symbol_library;

/* arithmetic coding models */
AC_MODEL coding_model0;
AC_MODEL coding_model1;

/* arithmetic encoder */
AC_ENCODER encoder;

/*****************************************************************************
  report_error_and_abort:
     Prints out the specified error message and terminates execution.
*****************************************************************************/
void report_error_and_abort(message)
char *message;
{ printf("\nCAFC_encode:      %s\n",message);
  exit(EXIT_FAILURE);
}

/*****************************************************************************
  write_CAFC:
    Writes a single bit to the destination CAFC file.
*****************************************************************************/
void write_CAFC(x)
int x;
{ static unsigned int CAFC_buffer=0;  /* internal byte buffer */
  static int CAFC_buffer_size = 0;

  /* Shift new bit into internal buffer. */
  CAFC_buffer = (CAFC_buffer << 1) + x;   CAFC_buffer_size++;

  /* If buffer is full, write byte to output file. */
  if (CAFC_buffer_size == 8)
  { fputc(CAFC_buffer, CAFC_dest);
    CAFC_buffer_size = 0;
    CAFC_buffer = 0;
  }
}

/*****************************************************************************
  RL_code:
    Perform the 2D run-length coding for a portion of the scan-line.
*****************************************************************************/
void RL_code(start_x, stop_x, start_model)
int start_x, stop_x, start_model;
{ int pos;
  int run;
  int i;

  /* Compute the difference between the present and previous scan-lines. */
  for (i=start_x; i<stop_x; i++)
    diff_line[i] = ! (((residue_prev_line[i] == WHITE) &&
                        (residue_line[i] == WHITE)) ||
                       ((residue_prev_line[i] != WHITE) &&
                        ((residue_line[i] != WHITE))));

  /* Now scan through line and perform coding. */
  pos = start_x;
  do
  { /* Detect run of 0s (vertically adjacent pixels match). */
    run = 0;
    while ((pos < stop_x) && (diff_line[pos] == 0))
    { pos++; run++; }

    if ((run > 0) || (start_model == 0))
    { /* Encode run and update model. */
      encode_element(&encoder,&coding_model0,run);
```

```
      update_model(&coding_model0,run,1);
    }

    /* Detect run of 1s (vertically adjacent pixels differ). */
    run = 0;
    while (( pos < stop_x) && (diff_line[pos] == 1))
    { pos++; run++; }

    if ((run > 0) || (start_model == 1))
    { /* Encode run and update model. */
      encode_element(&encoder,&coding_model1,run);
      update_model(&coding_model1,run,1);
    }
  }
  while (pos < stop_x);
}

/*****************************************************************************
  CAFC_encode
*****************************************************************************/
void main(argc,argv)
int argc;
char *argv[];
{ SYMBOL *detected_symbol;  /* symbol detected in image */
  LIBRARY *matched_entry;   /* symbol matched in library */
  int tot_symbols = 0, unique_symbols = 0; /* symbol counts */

  int escape, new_symbol;   /* AC elements for new symbol and end of page. */
  int stats0,stats1;             /* Run-length statistics. */

  int pos, lastpos;              /* horizontal positions on scan-line */
  int i;                         /* general counter variable */

  /* Make sure that the correct number of arguments are provided. */
  if ((argc != 3) && (argc != 4))
    report_error_and_abort("Invalid number of arguments.");

  /* Open source file, create buffer, and determine width. */
  source_buffer = open_PCX_buffered(argv[1],NLINES,PAGE_RESOLUTION);
  if (source_buffer == NULL)
    report_error_and_abort("Unable to open source PCX file.");
  maxX = buffer_maxX(source_buffer);

  /* Open destination and residue (if specified) files. */
  CAFC_dest = fopen(argv[2],"wb");
  if (CAFC_dest == NULL)
    report_error_and_abort("Unable to create destination CAFC file.");
  if (argc == 4)
  { residue = create_PCX(argv[3],maxX,PAGE_RESOLUTION);
    if (residue == NULL)
      report_error_and_abort("Unable to create residue file.");
  }

  /* Initialize AC models. */
  initialize_model(&coding_model0);
  initialize_model(&coding_model1);

  /* elements for run-lengths, ESCAPE, and NEW_SYMBOL */
  for (i=0; i<=maxX; i++)
  { add_element_to_model(&coding_model0);
    add_element_to_model(&coding_model1);
    update_model(&coding_model0,i,1);
    update_model(&coding_model1,i,1);
  }
  escape = add_element_to_model(&coding_model0);
  escape = add_element_to_model(&coding_model1);
```

```
update_model(&coding_model0, escape, 1);
update_model(&coding_model1, escape, 1);
new_symbol = add_element_to_model(&coding_model0);
new_symbol = add_element_to_model(&coding_model1);
update_model(&coding_model0, new_symbol, 1);
update_model(&coding_model1, new_symbol, 1);

/* Read run-length statistics, updating encoder models (if specified). */
if (RL_FILENAME[0] != '\0')
{
  rl = fopen(RL_FILENAME,"r");
  if (rl == NULL)
    report_error_and_abort("Unable to open run-length statistics file %s.\n",
                           RL_FILENAME);
  for (i=0; i<=maxX; i++)
  { fscanf(rl, "%d %d", &stats0, &stats1);
    update_model(&coding_model0, i, (int) (stats0 * RL_STATS_WEIGHT));
    update_model(&coding_model1, i, (int) (stats1 * RL_STATS_WEIGHT));
  }
  fclose(rl);
}

/* Initialize AC encoder. */
open_AC_encoder(&encoder, write_CAFC);

/* Create line to store differences between vertically adjacent pixels. */
diff_line = (byte *) malloc(maxX * sizeof(byte));

/* Create residue line and initialize with first scan-line. */
residue_line = (byte *) malloc(maxX * sizeof(byte));
for (i=0; i<maxX; i++)
  residue_line[i] = source_buffer[0][i];

/* Create previous residue line and initialize. */
residue_prev_line = (byte *) malloc(maxX * sizeof(byte));
for (i=0; i<maxX; i++)
  residue_prev_line[i] = WHITE;

/* Initialize symbol library. */
symbol_library = NULL;

/* Encode the image. */
while (! buffer_eof(source_buffer))
{ lastpos = 0;  /* 2D run-length coding begins at leftmost pixel. */

  /* Encode a scan-line. */
  pos = 0;  /* Start scanning from leftmost pixel. */
  while (pos < maxX)
  {
    /* Detect a white run by searching for first black pixel. */
    while ((pos < maxX) && (residue_line[pos] != BLACK))
         pos++;

    if (pos < maxX)
    { /* Attempt to isolate a symbol from the page */
         detected_symbol = isolate_symbol(source_buffer,pos);

         if (detected_symbol == NULL) /* If no symbol, skip black run. */
           while ((pos < maxX) && (residue_line[pos] != WHITE))
             pos++;
         else  /* Otherwise, try to match it with one in the library. */
           { tot_symbols++;

             matched_entry =
               lookup_symbol(symbol_library, detected_symbol, symbols_match);

             if (matched_entry ==  NULL) /* If no match, detected new symbol. */
             {
```

```
          /* Perform run-length coding up to present point. */
          RL_code(lastpos, pos, 0);
          lastpos = pos;

          /* Encode new symbol and add to symbol library. */
          encode_element(&encoder,&coding_model1,new_symbol);
          update_model(&coding_model1,new_symbol,1);
          symbol_library =
               add_symbol_to_library(symbol_library, detected_symbol);
          symbol_library->AC_element =
               add_element_to_model(&coding_model0);
          symbol_library->AC_element =
               add_element_to_model(&coding_model1);
          update_model(&coding_model0,symbol_library->AC_element,1);
          update_model(&coding_model1,symbol_library->AC_element,1);

          /* find next white pixel */
          while ((pos < maxX) && (residue_line[pos] != WHITE))
               pos++;

          /* Perform run-length coding up to present point. */
          RL_code(lastpos, pos, 1);
          lastpos = pos;

          unique_symbols++;
        }
        else /* Otherwise, encode as symbol from library. */
          {
               /* Erase the symbol from the page and from memory. */
               remove_symbol(detected_symbol, pos,
                             source_buffer, residue_line);
               free_symbol(detected_symbol);

               /* Adjust for symbol shifts. */
               pos += matched_entry->symbol->shift - detected_symbol->shift;

               /* Perform run-length coding up to present point. */
               if (pos > lastpos)
                 RL_code(lastpos, pos, 0);
               lastpos = pos;

               /* Encode element using appropriate model. */
               if (pos < 1)
                 encode_element(&encoder,&coding_model1,
                                matched_entry->AC_element);
               else if (diff_line[pos - 1] == 0)
                 encode_element(&encoder,&coding_model1,
                                matched_entry->AC_element);
               else
                 encode_element(&encoder,&coding_model0,
                                matched_entry->AC_element);

               /* Update both models. */
               update_model(&coding_model0,matched_entry->AC_element,1);
               update_model(&coding_model1,matched_entry->AC_element,1);
          }
      }
  }
}
RL_code(lastpos, maxX, 0); /* Run-length code to the end of the line. */

/* Write residue line if specified. */
if (argc == 4)
  write_line(residue,residue_line);

/* Update previous residue line. */
for (i=0; i<maxX; i++)
  residue_prev_line[i] = residue_line[i];
```

```
    isolate_scroll(source_buffer,residue_line);
  }

  /* Encode escape element to indicate end of page.  AC coding complete. */
  encode_element(&encoder,&coding_model0,escape);
  close_AC_encoder(&encoder);

  /* Flush output buffer. */
  for (i=0; i<7; i++)
    write_CAFC(0);

  /* Close files.*/
  close_buffer(source_buffer);
  if (argc == 4)
    close_PCX(residue);
  fclose(CAFC_dest);

  /* Print statistics. */
  printf("Encoding complete.\n\n");
  printf("    %d unique symbols\n",unique_symbols);
  printf("    %d total symbols\n",tot_symbols);
}
```

E.3 Source Code – CAFC Decoder

File `CAFC_decode.c`:

```
/**************************************************************************
Name:       CAFC_decode.c
Purpose:    Decodes a bi-level image using Content-Adaptive Bi-Level (Facsimile)
            Coding.  The CAFC-encoded source image is a binary file.  The
            destination reconstructed image is stored in the PCX file format.
            The residue image can be optionally created and stored in a PCX
            file.

Usage:      CAFC_decode CAFCsource PCXdest PCXresidue [ width ]

            CAFCsource -> filename of the source CAFC-encoded image
            PCXdest    -> filename of the reconstructed image (in PCX format)
            PCXresidue -> filename of residue image
            width      -> width of image in pixels; if omitted, 1728 is assumed

            To perform the 2D run-length encoding portion of the algorithm,
            run-length statistics are read from the data file named in
            CAFC.h if specified.

Notes:      All CAFC parameters are specified in the file CAFC.h.

Last modified on 1/13/94
**************************************************************************/
#include<stdio.h>
#include<stdlib.h>

/* CAFC include files. */
#include"CAFC.h"
#include"PCX_util.h"
#include"PCX_buffer.h"
#include"symbol.h"
#include"library.h"
#include"features.h"
#include"match.h"
#include"AC.h"
```

```
/* Preprocessor code to select to correct symbol filling functions. */
#if SYMBOL_ISOLATION == SYMBOL_FILLING
# define isolate_symbol symbol_filling_isolate
# define remove_symbol symbol_filling_remove
# define isolate_scroll symbol_filling_scroll
#elif SYMBOL_ISOLATION == SYMBOL_TRACING
# define isolate_symbol symbol_tracing_isolate
# define remove_symbol symbol_tracing_remove
# define isolate_scroll symbol_tracing_scroll
#elif SYMBOL_ISOLATION == SYMBOL_WINDOWING
# define isolate_symbol symbol_windowing_isolate
# define remove_symbol symbol_windowing_remove
# define isolate_scroll symbol_windowing_scroll
#endif

/* external CAFC routines */
SYMBOL *isolate_symbol();
void remove_symbol();
void isolate_scroll();

/* Define structure for storing information about decoded symbols. */
typedef struct
  { int pos, line;
    int AC_element;
    short int new;
  } SYMBOL_INFO;

/* source file (CAFC-encoded) */
FILE *CAFC_src;

/* file containing 2D run-length statistics */
FILE *rl;

/* buffers to store decoded scan-lines and residue */
byte **dest_buffer, **residue_buffer;

/* pointer to the previous scan-line in the output buffer.   */
byte *residue_prev_line;

/* width of image in pixels */
int maxX;

/* the symbol library */
LIBRARY *symbol_library;

/* arithmetic coding models */
AC_MODEL coding_model0;
AC_MODEL coding_model1;

/* arithmetic decoder */
AC_DECODER decoder;

/* list of all symbols in source buffer */
SYMBOL_INFO *symbol_list;
int symbol_list_max;
int symbol_list_start, symbol_list_size;


/*****************************************************************************
  report_error_and_abort:
      Prints out the specified error message and terminates execution.
*****************************************************************************/
void report_error_and_abort(message)
char *message;
{ printf("\nCAFC_decode:     %s\n",message);
  exit(EXIT_FAILURE);
}

/*****************************************************************************
  read_CAFC:
```

```
  Reads a single bit from the source CAFC file.
*****************************************************************************/
int read_CAFC()
{ static unsigned int CAFC_buffer;  /* internal byte buffer */
  static int CAFC_buffer_size = 0;

  /* If buffer is empty, fetch another byte from the file. */
  if (CAFC_buffer_size == 0)
  { CAFC_buffer = fgetc(CAFC_src);
    CAFC_buffer_size = 8;
  }

  /* Shift out a bit from the internal buffer. */
  CAFC_buffer_size--;
  return((CAFC_buffer >> CAFC_buffer_size) & 1);
}


/*****************************************************************************
  CAFC_decode
*****************************************************************************/
void main(argc,argv)
int argc;
char *argv[];
{ SYMBOL *detected_symbol; /* symbol detected in image */
  LIBRARY *matched_entry;  /* symbol matched in library */

  int element;               /* AC element read from CAFC file. */
  int escape, new_symbol;  /* AC elements for new symbol and end of page. */
  int stats0,stats1;         /* Run-length statistics. */

  int pos, line;            /* current horizontal and vertical position */
  int dest_buff_line;       /* current line in output buffer */
  int lines_written;        /* total number of lines written to output PCX files */
  int i, j;                 /* general counter variables */

  /* Make sure that the correct number of arguments are provided. */
  if ((argc != 4) && (argc != 5))
    report_error_and_abort("Invalid number of arguments.");

  /* Open source CAFC file. */
  CAFC_src = fopen(argv[1],"rb");
  if (CAFC_src == NULL)
    report_error_and_abort("Unable to open source CAFC file.");

  /* Determine image width. */
  if (argc == 5)
    sscanf(argv[4],"%d",&maxX);
  else
    maxX = 1728;
  if (maxX < 0)
    report_error_and_abort("Invalid image width.");

  /* Open destination and residue files. */
  dest_buffer = create_PCX_buffered(argv[2],maxX,NLINES,PAGE_RESOLUTION);
  if (dest_buffer == NULL)
    report_error_and_abort("Unable to create destination PCX file.");
  dest_buff_line = 0;
  residue_buffer = create_PCX_buffered(argv[3],maxX,NLINES,PAGE_RESOLUTION);
  if (residue_buffer == NULL)
    report_error_and_abort("Unable to create residue file.");

  /* Initialize symbol list. */
  symbol_list_max =
    (long) maxX * NLINES / (MIN_SYMBOL_HEIGHT + 2) / (MIN_SYMBOL_WIDTH + 2);
  symbol_list = (SYMBOL_INFO *) malloc(symbol_list_max * sizeof(SYMBOL_INFO));
  symbol_list_start = 0;  symbol_list_size = 0;
```

```
/* Initialize AC models. */
initialize_model(&coding_model0);
initialize_model(&coding_model1);

/* elements for run-lengths, ESCAPE, and NEW_SYMBOL */
for (i=0; i<=maxX; i++)
{ add_element_to_model(&coding_model0);
  add_element_to_model(&coding_model1);
  update_model(&coding_model0,i,1);
  update_model(&coding_model1,i,1);
}
escape = add_element_to_model(&coding_model0);
escape = add_element_to_model(&coding_model1);
update_model(&coding_model0, escape, 1);
update_model(&coding_model1, escape, 1);
new_symbol = add_element_to_model(&coding_model0);
new_symbol = add_element_to_model(&coding_model1);
update_model(&coding_model0, new_symbol, 1);
update_model(&coding_model1, new_symbol, 1);

/* Read run-length statistics, updating encoder models (if specified). */
if (RL_FILENAME[0] != '\0')
{
  rl = fopen(RL_FILENAME,"r");
  if (rl == NULL)
    report_error_and_abort("Unable to open run-length statistics file %s.\n",
                           RL_FILENAME);
  for (i=0; i<=maxX; i++)
  { fscanf(rl, "%d %d", &stats0, &stats1);
    update_model(&coding_model0, i, (int) (stats0 * RL_STATS_WEIGHT));
    update_model(&coding_model1, i, (int) (stats1 * RL_STATS_WEIGHT));
  }
  fclose(rl);
}

/* Initialize AC decoder. */
open_AC_decoder(&decoder, read_CAFC);

/* Initialize symbol library. */
symbol_library = NULL;

/* Obtain pointer to previous residue scan-line. */
residue_prev_line = buffer_prev_line(residue_buffer);

/* Decode the image. */
line = 0;              /* initialize line number */
lines_written = 0; /* Initialize # lines written. */
element = decode_element(&decoder,&coding_model0); /* Decode 1st element. */

/* Decode until last symbol is reached and all lines have been written. */
while ((element != escape) || (lines_written < line))
{
  if (element != escape)
  {
    /* Decode a scan-line. */
    pos = 0;       /* Start at leftmost pixel. */
    line++;        /* Advance to next scan-line. */

    while (pos < maxX)
    {
        /* Decode AC element. */
        if (element > maxX) /* Is this a repeated symbol? */
        {
          /* If so, add to symbol list for future substitution. */
          i = (symbol_list_start + symbol_list_size) % symbol_list_max;
          symbol_list_size++;
          symbol_list[i].AC_element = element;
```

```
  symbol_list[i].pos = pos;  symbol_list[i].line = line;
  symbol_list[i].new = 0;

  /* Update AC model and decode next AC element. */
  update_model(&coding_model0,element,1);
  update_model(&coding_model1,element,1);
  element = decode_element(&decoder,&coding_model0);
}
else
{ /* Decode a run of 0s (vertically adjacent pixels same). */
  for (i=0; i<element; i++)
    { dest_buffer[dest_buff_line][pos] = residue_prev_line[pos];
      residue_buffer[dest_buff_line][pos] = residue_prev_line[pos];
      pos++;
    }

  /* Update AC model. */
  update_model(&coding_model0,element,1);

  /* If not end of line, decode next element - symbol or run of 1s. */
  if (pos >= maxX)
    element = decode_element(&decoder,&coding_model0);
  else
  { element = decode_element(&decoder,&coding_model1);

    /* Examine decoded element. */
    if (element == new_symbol) /* new symbol? */
      {
          /* Add to symbol list for future isolation. */
          i = (symbol_list_start + symbol_list_size) % symbol_list_max;
          symbol_list_size++;
          symbol_list[i].AC_element =
            add_element_to_model(&coding_model0);
          symbol_list[i].AC_element =
            add_element_to_model(&coding_model1);
          symbol_list[i].pos = pos;  symbol_list[i].line = line;
          symbol_list[i].new = 1;

          /* Update AC model and decode next element. */
          update_model(&coding_model0,symbol_list[i].AC_element,1);
          update_model(&coding_model1,symbol_list[i].AC_element,1);
          update_model(&coding_model1,element,1);
          element = decode_element(&decoder,&coding_model1);

          /* Decode run of 1s (vertically adjacent pixels different). */
          for (i=pos; i < pos+element; i++)
            { if (residue_prev_line[i] == WHITE)
                    dest_buffer[dest_buff_line][i] = BLACK;
              else
                dest_buffer[dest_buff_line][i] = WHITE;
                residue_buffer[dest_buff_line][i] =
                    dest_buffer[dest_buff_line][i];
            }
          pos += element;
      }
    else if (element > maxX) /* Is this a repeated symbol? */
      {
          /* If so, add to symbol list for future substitution. */
          i = (symbol_list_start + symbol_list_size) % symbol_list_max;
          symbol_list_size++;
          symbol_list[i].AC_element = element;
          symbol_list[i].pos = pos;  symbol_list[i].line = line;
          symbol_list[i].new = 0;

          /* Update AC model. */
          update_model(&coding_model0,element,1);
```

```
          }
        else
          { /* Decode run of 1s (vertically adjacent pixels different) */
              for (i=pos; i < pos+element; i++)
                { if (residue_prev_line[i] == WHITE)
                    dest_buffer[dest_buff_line][i] = BLACK;
                  else
                    dest_buffer[dest_buff_line][i] = WHITE;
                  residue_buffer[dest_buff_line][i] =
                    dest_buffer[dest_buff_line][i];
                }
              pos += element;
          }
        /* Update AC model and decode next element. */
        update_model(&coding_model1,element,1);
        element = decode_element(&decoder,&coding_model0);
      }
    }
  }
}

/* Advance to the next destination scan-line. */
if (dest_buff_line < (NLINES - 1))
{ residue_prev_line = residue_buffer[dest_buff_line];
  dest_buff_line++;
}
else
  { /* Determine if symbols were marked for isolation or substitution. */
      while ((symbol_list_size > 0) &&
             (symbol_list[symbol_list_start].line == (lines_written + 1)))
        {
          if (symbol_list[symbol_list_start].new)
          { /* Isolate new symbol and add to symbol library. */
            detected_symbol =
                isolate_symbol(residue_buffer,
                               symbol_list[symbol_list_start].pos);
            if (detected_symbol == NULL)
                report_error_and_abort("Major internal decoding error.");
            symbol_library =
                add_symbol_to_library(symbol_library, detected_symbol);
            symbol_library->AC_element =
                symbol_list[symbol_list_start].AC_element;
          }
          else
          { /* Search library for original version of symbol. */
            matched_entry = symbol_library;
            while (matched_entry->AC_element !=
                     symbol_list[symbol_list_start].AC_element)
                matched_entry = matched_entry->next_entry;

            /* Copy symbol to destination buffer. */
            for (j = 0; j < matched_entry->symbol->maxY; j++)
                for (i = 0; i < matched_entry->symbol->maxX; i++)
                  if (matched_entry->symbol->bitmap[j][i] == BLACK)
                    dest_buffer[j][i + symbol_list[symbol_list_start].pos -
                                matched_entry->symbol->shift] = BLACK;
          }

          /* Remove request from symbol list. */
          symbol_list_size--;
          symbol_list_start = (symbol_list_start + 1) % symbol_list_max;
        }
      /* Generate residue, scroll output buffers... */
      for (i=0; i<maxX; i++)
        if (residue_buffer[0][i] != WHITE)
          residue_buffer[0][i] = BLACK;
```

```
        scroll_buffer(dest_buffer);
        scroll_buffer(residue_buffer);
        residue_prev_line = residue_buffer[dest_buff_line - 1];
        lines_written++;
    }
  }

  /* Close output files.  AC decoding complete. */
  close_AC_decoder(&decoder);
  close_buffer(dest_buffer);
  close_buffer(residue_buffer);
  fclose(CAFC_src);
}
```

E.4 Source Code – Symbol Matching

File `match.h`:

```
/***************************************************************************
Name:     match.h
Purpose:  This header file contains definitions used by the symbol matching
          routines (match.c).

Last Modified on 12/30/93
***************************************************************************/

/* declarations for symbol matching routines */
int symbols_match();
int features_match();
int templates_match();
```

File `match.c`:

```
/***************************************************************************
Name:       match.c
Purpose:    This file contains routines for matching symbols (feature matching
            and template matching).

Contents:
  symbols_match(s1, s2)     => Determines if two symbols match based upon
                               feature matching and template matching.
  features_match(s1, s2)  => Determines if two symbols match based upon
                               feature matching.
  templates_match(s1, s2) => Determines if two symbols match based upon
                               template matching.

Last modified on 12/30/93
***************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include"CAFC.h"
#include"PCX_util.h"
#include"symbol.h"
#include"features.h"
#include"match.h"

/* Definition for arrays containing features matching parameters. */
int feature_match_threshold[] = FEATURE_MATCH_THRESHOLD;
```

```
                                        /* thresholds to use for matching */
int feature_effectiveness[] = FEATURE_EFFECTIVENESS;
                                        /* effectiveness of features */
/******************************************************************************
  symbols_match:
      Determines if the two specified symbols are a good match using
      feature matching and template matching.  Returns 1 if they are and 0
      otherwise.
******************************************************************************/
int symbols_match(s1, s2)
SYMBOL *s1, *s2;
{
  if (features_match(s1, s2))
    if (templates_match(s1, s2))
      return(1);
  else
    return(0);
}

/******************************************************************************
  features_match:
      Using the CAFC feature matching algorithm, determines if the two
      specified symbols are likely to be a good match.          Returns 1 if they
      are and 0 otherwise.
******************************************************************************/
int features_match(s1, s2)
SYMBOL *s1, *s2;
{ int i;
  int f1, f2;
  int diff;
  int eliminated;

  eliminated = 0;
  i=0;

  /* Attempt to match the symbols using succesive features. */
  while ((i < NFEATURES) && (! eliminated))
    {
      /* Extract the features from the two symbols. */
      f1 = extract_feature(s1,i);
      f2 = extract_feature(s2,i);

      /* if the abolute difference exceeds the threshold, eliminate. */
      diff = (f1 - f2);
      if (diff < 0)
            diff = -diff;
      if (diff >= feature_match_threshold[i])
            eliminated = 1;
      i++;
    }
  return (! eliminated);
}

/******************************************************************************
  templates_match:
      Using the CAFC template matching algorithm, determines if the two
      specified symbols are a good match.  Returns 1 if they are and 0
      otherwise.
******************************************************************************/
int templates_match(s1, s2)
SYMBOL *s1, *s2;
{
  int x, y, t_x, t_y;
  int s1_moment_x,s1_moment_y,s2_moment_x,s2_moment_y;
  int s1_tot_black,s2_tot_black;
  float correlation;

  /* Compute the "center of mass" for s1 to resolution of 1/8 pixel. */
```

```
  s1_moment_x = 0; s1_moment_y = 0;
  s1_tot_black = 0;
  for (x=0; x < s1->maxX; x++)
    for (y=0; y < s1->maxY; y++)
      if (s1->bitmap[y][x] == BLACK)
            { s1_moment_x += x;
              s1_moment_y += y;
              s1_tot_black++;
            }
  s1_moment_x = 2 * s1_moment_x / s1_tot_black;
  s1_moment_y = 2 * s1_moment_y / s1_tot_black;

  /* Compute the "center of mass" for s2 to resolution of 1/8 pixel. */
  s2_moment_x = 0; s2_moment_y = 0;
  s2_tot_black = 0;
  for (x=0; x < s2->maxX; x++)
    for (y=0; y < s2->maxY; y++)
      if (s2->bitmap[y][x] == BLACK)
            { s2_moment_x += x;
              s2_moment_y += y;
              s2_tot_black++;
            }
  s2_moment_x = 2 * s2_moment_x / s2_tot_black;
  s2_moment_y = 2 * s2_moment_y / s2_tot_black;

  /* Make sure symbols are lined up well. */
  if ((abs(s1_moment_x - s2_moment_x) <= TEMPLATE_MAXIMUM_SHIFT_X) &&
      (abs(s1_moment_y - s2_moment_y) <= TEMPLATE_MAXIMUM_SHIFT_Y))
  {
    /* Now compute the cross-correlation. */
    correlation = 0.0;
    for (y=0; (y < (s1->maxY*2)); y++)
      for (x=0; (x < (s1->maxX*2)); x++)
            { t_x = (x - s1_moment_x + s2_moment_x)/2;
              t_y = (y - s1_moment_y + s2_moment_y)/2;
              if ((t_x >= 0) && (t_y >= 0) &&
                  (t_x < s2->maxX) && (t_y < s2->maxY))
                correlation += (s1->bitmap[y/2][x/2] == BLACK) *
                  (s2->bitmap[t_y][t_x] == BLACK);
            }

    /* This is actually the square of the correlation... */
    correlation = correlation * correlation /
                  (s1_tot_black * s2_tot_black * 4 * 4);

    /* If correlation is high enough, symbols are considered a match. */
    return(correlation > TEMPLATE_MATCH_THRESHOLD);

  }
  else
    return(0);
}
```

:

E.5 Source Code – Feature Extraction

File **features.h**:

```
/*****************************************************************************
Name:      features.h
```

```
Purpose:  This header file contains definitions used by the feature extraction
          routines (features.c).

Last Modified on 12/30/93
******************************************************************************/

/* declaration of features */
 int width();
int height();
int black_pels();
int white_pels();
int horiz_run_lengths();
int vert_run_lengths();
int moment_x();
int moment_y();
int average_width();

/* arrays that contain the features and feature names */
extern int (*features[]) ();
extern char *feature_names[];

/* declarations for feature extraction routine */
int extract_feature();
```

File `features.c`:

```
/******************************************************************************
Name:       features.c
Purpose:    This file contains routines to extract the symbol features
         used by Content-Adaptive Facsimile Coding (CAFC).

Contents:
  extract_feature(symbol, f_index)  => Extracts specified feature from symbol.

  definitions for all features used in CAFC

Last modified on 12/23/93
******************************************************************************/

#include<stdio.h>
#include"CAFC.h"
#include"PCX_util.h"
#include"symbol.h"
#include"features.h"

/* Definition for arrays containing features and feature names. */
int (*features[]) () = FEATURES;       /* functions to compute features */
char *feature_names[] = FEATURE_NAMES; /* corresponding features names */
/******************************************************************************
  extract_feature:
    Given a feature index number, extracts the feature from the symbol.  If
    the feature had been previously extracted, the value is obtained from
    the SYMBOL structure.  Otherwise it is computed and stored away for
    possible later use.
******************************************************************************/
int extract_feature(symbol, f_index)
SYMBOL *symbol;
int f_index;
{
 /* If feature has not already been determined for this symbol, compute it. */
 if (! symbol->f_known[f_index])
   {
    symbol->features[f_index] = (*features[f_index]) (symbol);
    symbol->f_known[f_index] = 1;
```

```
   }

  /* Return feature value. */
  return(symbol->features[f_index]);
}

/****************************************************************************/
/* FEATURES BEGIN HERE */

/* Width of the symbol in pixels. */
int width(s)
SYMBOL *s;
{ return(s->maxX);
}

/* Height of the symbol in pixels. */
int height(s)
SYMBOL *s;
{ return(s->maxY);
}

/* Total number of black pixels in the symbol. */
int black_pels(s)
SYMBOL *s;
{ int x,y;
  int tot_black_pels;

  tot_black_pels = 0;
  for (y=0; y < s->maxY; y++)
    for (x=0; x < s->maxX; x++)
      tot_black_pels += (s->bitmap[y][x] == BLACK);
  return(tot_black_pels);
}

/* Total number of white pixels in the symbol. */
int white_pels(s)
SYMBOL *s;
{ int x,y;
  int tot_white_pels;

  tot_white_pels = 0;
  for (y=0; y < s->maxY; y++)
    for (x=0; x < s->maxX; x++)
      tot_white_pels += (s->bitmap[y][x] == WHITE);
  return(tot_white_pels);
}

/* Total number of horizontal black run-lengths in the symbol. */
int horiz_run_lengths(s)
SYMBOL *s;
{ int x,y;
  int tot_run_lengths;

  tot_run_lengths = 0;
  for (y=0; y < s->maxY; y++)
    for (x=0; x < s->maxX; x++)
      if (s->bitmap[y][x] == BLACK)
            if (x == (s->maxX - 1))
              tot_run_lengths++;
            else if (s->bitmap[y][x+1] == WHITE)
              tot_run_lengths++;
  return(tot_run_lengths);
}

/* Total number of vertical black run-lengths in the symbol. */
int vert_run_lengths(s)
SYMBOL *s;
```

```
{ int x,y;
  int tot_run_lengths;

  tot_run_lengths = 0;
  for (x=0; x < s->maxX; x++)
    for (y=0; y < s->maxY; y++)
      if (s->bitmap[y][x] == BLACK)
            if (y == (s->maxY - 1))
              tot_run_lengths++;
            else if (s->bitmap[y+1][x] == WHITE)
              tot_run_lengths++;
  return(tot_run_lengths);
}

/* Horizontal moment of symbol. */
int moment_x(s)
SYMBOL *s;
{ int x,y;
  int moment, tot_black;

  moment = 0; tot_black = 0;
  for (x=0; x < s->maxX; x++)
    for (y=0; y < s->maxY; y++)
      if (s->bitmap[y][x] == BLACK)
            { moment += x;
              tot_black++;
            }

  return(moment/tot_black);
}

/* Vertical moment of symbol. */
int moment_y(s)
SYMBOL *s;
{ int x,y;
  int moment, tot_black;

  moment = 0; tot_black = 0;
  for (x=0; x < s->maxX; x++)
    for (y=0; y < s->maxY; y++)
      if (s->bitmap[y][x] == BLACK)
            { moment += y;
              tot_black++;
            }

  return(moment/tot_black);
}

/* Average width. */
int average_width(s)
SYMBOL *s;
{ int y;
  int left, right;
  int tot_width;

  tot_width = 0;

  for (y=0; y < s->maxY; y++)
  {
    left = 0;
    while ((left < s->maxX) && (s->bitmap[y][left] == WHITE))
      left++;

    right = s->maxX - 1;
    while ((right > 0) && (s->bitmap[y][right] == WHITE))
      right--;

    if (right != 0)
      tot_width += right - left + 1;
```

```
  }
  return(tot_width/s->maxY);
}
```

E.6 Source Code – Symbol Library Management

File `library.h`:

```
/*****************************************************************************
Name:     library.h
Purpose:  This header file contains definitions used by the library managment
          routines (library.c).

Last Modified on 12/29/93
*****************************************************************************/

/* definition for symbol library type */
typedef struct library_entry
  { SYMBOL *symbol;      /* the symbol itself */
    int n_occurrences;   /* number of times it has occurred in document */
    int AC_element;      /* arithmetic coding element number */
    struct library_entry *next_entry; /* pointer to the next library entry */
  } LIBRARY;

/* declarations for library managment routines */
LIBRARY  *add_symbol_to_library();
LIBRARY  *lookup_symbol();
```

File `library.c`:

```
/*****************************************************************************
Name:     library.c
Purpose:  This file contains routines for manipulating symbol libraries.

The LIBRARY structure is defined in library.h

Contents:
  add_symbol_to_library(symbol_library, symbol)          => Create new library entry
                                                            with specified symbol.
  lookup_symbol(symbol_library, symbol, compare) => Search library for
                                                            matching symbol.

Last modified on 12/29/93
*****************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include"CAFC.h"
#include"symbol.h"
#include"library.h"

/*****************************************************************************
  add_symbol_to_library:
     Adds a symbol to a symbol library.          Returns a new pointer to the library.
*****************************************************************************/
LIBRARY  *add_symbol_to_library(symbol_library, symbol)
LIBRARY *symbol_library;
SYMBOL *symbol;
```

```
{ LIBRARY *new_entry;

  /* Allocate space for the new library entry and initialize. */
  new_entry = (LIBRARY *) malloc(sizeof(LIBRARY));
  new_entry->symbol = symbol;     /* use specified symbol */
  new_entry->n_occurrences = 0; /* initially 0 occurrences */

  /* Link new entry to symbol library. */
  new_entry->next_entry = symbol_library;

  /* Return new pointer to symbol library. */
  return(new_entry);
}

/***************************************************************************
  lookup_symbol:
      Determines if a specified symbol can be matched to one in a symbol
      library.        Takes as argments pointers to the symbol library, the
      symbol, and a function to perform the comparisons.  If a successful
      match is made, a pointer to the matching library entry is returned.
      Otherwise, NULL is returned.  The library is automatically sorted
      so that the more frequently-occurring symbols are kept at the beginning.
***************************************************************************/
LIBRARY  *lookup_symbol(symbol_library, symbol, compare)
LIBRARY *symbol_library;
SYMBOL *symbol;
int (*compare) ();
{ int found;
  LIBRARY *lib_ptr; /* pointer to search the library */
  SYMBOL *temp_symbol; /* temporary variables */
  int temp_int;        /* used for swapping */

  lib_ptr = symbol_library;

  /* Search symbol library for matching symbol. */
  found = 0;
  while ((lib_ptr != NULL) && (! found))
    if ((*compare) (symbol, lib_ptr->symbol))
          found = 1;
      else
          lib_ptr = lib_ptr->next_entry;

  /* If there was a match, return pointer to that entry; otherwise, NULL. */
  if (found)
  {
    /* Increment number of occurrences. */
    lib_ptr->n_occurrences++;

    /* Move to new position in library to keep sorted. */
    found = 0;

    while (! found)
      if (symbol_library->n_occurrences <= lib_ptr->n_occurrences)
          { found = 1;

            /* swap the contents of the library entries */
            temp_symbol = lib_ptr->symbol;
            lib_ptr->symbol = symbol_library->symbol;
            symbol_library->symbol = temp_symbol;
            temp_int = lib_ptr->AC_element;
            lib_ptr->AC_element = symbol_library->AC_element;
            symbol_library->AC_element = temp_int;
            temp_int = lib_ptr->n_occurrences;
            lib_ptr->n_occurrences = symbol_library->n_occurrences;
            symbol_library->n_occurrences = temp_int;
          }
      else
          symbol_library = symbol_library->next_entry;
```

```
    return(symbol_library);
  }
  else
    return(NULL);
}
```

E.7 Source Code – Symbol Isolation

File `symbol_filling.c`:

```
/****************************************************************************
Name:     symbol_filling.c
Purpose:  Performs the symbol isolation stage of Content-Adaptive Facsimile
          Coding using the symbol filling technique.

Contents:
  symbol_filling_isolate(buffer,position,residue_line)
                          => Attempts to isolate a particular symbol given
                             a source buffer and position of reference pixel.
  symbol_filling_remove(symbol,position,buffer,residue_line)
                          => Removes a detected symbol from the source buffer.
  symbol_filling_scroll(buffer,residue_line)
                          => Scrolls the source buffer up one scan-line in
                             a manner that preserves important side information
                             used by symbol_filling_isolate and updates the
                             residue scan-line.

Last modified on 12/17/93
****************************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include"CAFC.h"
#include"PCX_util.h"
#include"PCX_buffer.h"
#include"symbol.h"

/* additional pixel representations (for black foreground) */
#define SYMBOL_PIXEL 3
#define NON_SYMBOL   4

/* global variables used during isolation */
int left, right, top, bottom; /* boundries of symbol */
byte tag;                     /* value to use when tagging pixels */
byte **buffer;                /* buffer to scan for symbols */
int maxX, nlines;             /* boundries of buffer */


/****************************************************************************
 fill:    Takes as arguments the coordinates of a black pixel in the source
          buffer.  Uses a flood-fill algorithm to tag the entire cluster of
          contiguous black pixels.  In the process, determines the maximum
          boundries of this cluster.  This function is internal to
          symbol_filling.c.

          The algorithm first tags all contiguous black pixels in the
          horizontal scan-line segment consisting of the specified pixel.
          Then, it recursively calls itself with the coordinates of all black
          pixels immediately above and below this segment.

 The following global variables are used and assumed to be preset with
 the appropriate values prior to the call to this function:

 byte **buffer       => two-dimensional array of pixels to scan
 int maxX, nlines  => horizonal and vertical dimensions of this array
```

```
  int left, right,  => outermost boundries of filled region -- assumes that
     top, bottom    =>       initially left=right=x and top=bottom=y
     byte tag           => value to tag region with
*****************************************************************************/
void fill(x, y)
int x, y;
{ int line_left,line_right;  /* boundries of current scan-line */

  /* Tag all contiguous black pixels to the left on current scan-line (y). */
  line_left = x;
  while ((line_left > 0) && (buffer[y][line_left] == BLACK))
  { buffer[y][line_left] = tag;
    line_left--;
  }

  /* Check for special case of left edge of buffer. */
  if ((line_left == 0) && (buffer[y][0] == BLACK))
    buffer[y][line_left] = tag;
  else
    line_left++;  /* line_left gets leftmost tagged pixel. */

  /* Tag all contiguous black pixels to the right on current scanline (y). */
  if (x == (maxX - 1)) /* If we are already at the rightmost edge, */
    line_right = x;    /* skip this part. */
  else
    { line_right = x + 1;
      while ((line_right < (maxX - 1)) && (buffer[y][line_right] == BLACK))
      { buffer[y][line_right] = tag;
          line_right++;
      }

      /* Check for special case of right edge of buffer. */
      if ((line_right == (maxX - 1)) && (buffer[y][maxX - 1] == BLACK))
          buffer[y][line_right] = tag;
      else
          line_right--;  /* line_right gets rightmost tagged pixel. */
    }

  /* Expand left and right boundries if necessary. */
  if (line_left < left)          left = line_left;
  if (line_right > right) right = line_right;

  /* For all black pixels above and below tagged segment, recursively call
     fill.  Expand  top and bottom boundries when necessary. */
  for (x = (line_left - 1); x <= (line_right + 1); x++)
    if ((x >= 0) && (x < maxX))
    { if (y > 0)
          if (buffer[y - 1][x] == BLACK)
            { if ((y - 1) < top)
                top = y - 1;
              fill(x, y - 1);
            }
      if (y < (nlines - 1))
          if (buffer[y + 1][x] == BLACK)
            { if ((y + 1) > bottom)
                bottom = y + 1;
              fill(x, y + 1);
            }
    }
}

/*****************************************************************************
  symbol_filling_isolate:
     Given a source buffer and location of a reference black pixel,
     attempts to isolate a cluster of contiguous black pixels to form
     a symbol.      If a cluster can be isolated that fits within the allowed
     size constraints, it is returned in the form of a symbol structure.
     Otherwise, NULL is returned.
```

```
    Important side information is stored in the buffer so that large
    clusters are handled correctly.  The caller should not directly access
    the buffer.  Instead, a seperate one-dimensional array of pixels,
    residue_line, should be maintained that contains only the first
    scan-line of the buffer.          It is automatically updated with the
    symbol_filling_scroll function.  Detected symbols can be properly
    removed from the source image with the symbol_filling_remove function.

    The following constants must be defined:
        MAX_SYMBOL_HEIGHT, MAX_SYMBOL_WIDTH => maximum allowed symbol size
        MIN_SYMBOL_HEIGHT, MIN_SYMBOL_WIDTH => minimum allowed symbol size
******************************************************************************/
SYMBOL *symbol_filling_isolate(source_buffer, position)
byte **source_buffer;
int position;
{ int x,y;
  SYMBOL *detected_symbol;

  /* Make sure that pixel in specified position is a possible candidate. */
  if (source_buffer[0][position] == BLACK)
  {
    /* Assign appropriate values to global variables referring to buffer. */
    buffer = source_buffer;
    maxX = buffer_maxX(buffer); nlines = buffer_nlines(buffer);

    /* Fill contiguous black region with the value SYMBOL_PIXEL. */
    left = position; right = position; top = 0; bottom = 0;
    tag = SYMBOL_PIXEL;
    fill(position, 0);

    /* If region fits within the symbol size constraints, it is a symbol. */
    if (((bottom + 1) <= MAX_SYMBOL_HEIGHT) &&
          ((right - left + 1) <= MAX_SYMBOL_WIDTH) &&
          ((bottom + 1) >= MIN_SYMBOL_HEIGHT) &&
          ((right - left + 1) >= MIN_SYMBOL_WIDTH))
    {
      /* Create a new symbol structure for this symbol. */
      detected_symbol = create_symbol(right-left+1, bottom+1, position-left);

      /* Copy the symbol to the bitmap field. */
      for (y = 0; y <= bottom; y++)
          for (x=left; x <= right; x++)
            detected_symbol->bitmap[y][x - left] =
              (source_buffer[y][x] == SYMBOL_PIXEL) ? BLACK : WHITE;
    }
    else /* otherwise, not a symbol */
      detected_symbol = NULL;

    /* Retag all black pixels as NON-SYMBOL. */
    for (y=0; y <= bottom; y++)
      for (x=left; x <= right; x++)
          if (source_buffer[y][x] == SYMBOL_PIXEL)
            source_buffer[y][x] = NON_SYMBOL;
  }
  else
    detected_symbol = NULL;

  return(detected_symbol);
}

/******************************************************************************
  symbol_filling_remove:
    Given a detected symbol, its location, the source buffer, and a
    seperate residue scan-line array, erases the symbol from the source
    buffer and residue line.          This prevents the symbol from being detected
    again or from appearing in subsequent residue lines.
******************************************************************************/
```

```
void symbol_filling_remove(detected_symbol, position,
                           source_buffer, residue_line)
SYMBOL *detected_symbol;
int position;
byte **source_buffer;
byte *residue_line;
{ int x,y;

  /* Erase symbol from source buffer. */
  for (y = 0; y < detected_symbol->maxY; y++)
    for (x = 0; x < detected_symbol->maxX; x++)
      if (detected_symbol->bitmap[y][x] == BLACK)
          source_buffer[y][x + position - detected_symbol->shift] = WHITE;

  /* Remove symbol from residue_line. */
  for (x = 0; x < detected_symbol->maxX; x++)
    if (detected_symbol->bitmap[0][x] == BLACK)
      residue_line[x + position - detected_symbol->shift] = WHITE;
}

/***************************************************************************
  symbol_filling_scroll:
     Scrolls a buffer up by one scan-line.  In the process, propogates
     down any pixels marked as NON-SYMBOL so that they will not later
     be incorrectly isolated.          In addition, generates a new residue_line.
***************************************************************************/
void symbol_filling_scroll(source_buffer,residue_line)
byte **source_buffer;
byte *residue_line;
{ int x;

  /* Scroll up buffer and scan next line. */
  scroll_buffer(source_buffer);

  /* Determine dimensions of buffer. */
  maxX = buffer_maxX(source_buffer);
  nlines = buffer_nlines(source_buffer);

  /* Propogate down any non-symbols that could not fit in buffer. */
  tag = NON_SYMBOL;
  buffer = source_buffer;
  for (x=0; x<maxX; x++)
    if (source_buffer[nlines - 2][x] == NON_SYMBOL)
    { if (source_buffer[nlines - 1][x] == BLACK)
          fill(x, nlines - 1);
      if (x > 0)
          if (source_buffer[nlines - 1][x - 1] == BLACK)
            fill(x - 1, nlines - 1);
      if (x < (maxX - 1))
          if (source_buffer[nlines - 1][x + 1] == BLACK)
            fill(x + 1, nlines - 1);
    }

  /* Generate new residue_line. */
  for (x=0; x<maxX; x++)
    residue_line[x] = (source_buffer[0][x] == WHITE) ? WHITE : BLACK;
}
```

File `symbol_tracing.c`:

```
/***************************************************************************
Name:      symbol_tracing.c
Purpose:   Performs the symbol isolation stage of Content-Adaptive Facsimile
           Coding using the symbol tracing technique.
```

```
Contents:
  symbol_tracing_isolate(buffer,position,residue_line)
                          => Attempts to isolate a particular symbol given
                               a source buffer and position of reference pixel.
  symbol_tracing_remove(symbol,position,buffer,residue_line)
                          => Removes a detected symbol from the source buffer.
  symbol_tracing_scroll(buffer,residue_line)
                          => Scrolls the source buffer up one scan-line,
                               updating the residue scan-line.

Last modified on 12/20/93
***************************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include"CAFC.h"
#include"PCX_util.h"
#include"PCX_buffer.h"
#include"symbol.h"

/* additional pixel representations (for black foreground) */
#define  BOUNDRY          5
#define  DOUBLE_BOUNDRY    6
#define  NON_SYMBOL_BOUNDRY 7

/* global variables used during isolation */
int left, right, top, bottom; /* boundries of symbol */
byte **buffer;                /* buffer to scan for symbols */
byte *prev_line;              /* scan-line immediately preceeding buffer */
int maxX, nlines;             /* boundries of buffer */

/***************************************************************************
 trace:   Takes as arguments the coordinates of a black pixel in the source
          buffer.  Uses a contour tracing algorithm to tag the outline of
          a black cluster in the image.        In the process, determines the maximum
          boundries of this cluster.  Returns 1 if the trace ends on the same
          pixel that it started.          This function is internal to symbol_tracing.c.

          The specified pixel is used as a starting point for the trace and
          should be on the right or upper boundry of the black object.  The
          trace is performed in the clockwise direction and ends when the
          original pixel is retraced in the same direction or when the upper
          or lower ends of the buffer are exceeded.  The right and left ends of
          each horizontal segment in the object are tagged with the value
          provided in the argument boundry_tag or, when this segment is just
          one pixel wide, double_boundry_tag.

  The following global variables are used and assumed to be preset with
  the appropriate values prior to the call to this function:

  byte **buffer        => two-dimensional array of pixels to scan
  byte *prev_line   => scan-line immediately preceeding buffer
  int maxX, nlines  => horizonal and vertical dimensions of this array
  int left, right,  => outermost boundries of traced region provided here
      top, bottom
***************************************************************************/
int trace(start_x, start_y, boundry_tag, double_boundry_tag)
int start_x, start_y, boundry_tag, double_boundry_tag;
{ int dx, dy, temp; /* direction of trace */
  int x, y;  /* position of current pixel in trace */
  pixel p;

  /* Initialize starting pixel, outermost traced boundries, and direction. */
  x = start_x;      y = start_y;                    /* starting pixel */
  left = x; right = x; top = y; bottom = y;  /* outer boundries */
  dx = 1; dy = 0;                                   /* initial direction */

  /* Trace until starting point is reached in the same direction or
     the top (including prev_line) or bottom of the buffer is exceeded. */
  while ((! ((dx == 1) && (dy == -1) && (x == start_x) && (y == start_y))) &&
          ((y + dy) >= -1) && ((y + dy) < nlines))
```

```
  {
   /* Determine color of pixel in current direction, WHITE if beyond edge. */
   if (((x + dx) >= 0) && ((x + dx) < maxX))
     if ((y + dy) == -1)
          p = prev_line[x+dx];
     else
          p = buffer[y+dy][x+dx];
   else
     p = WHITE;

   /* If this pixel is white, tag if necessary and rotate clockwise. */
   if (p == WHITE)
     {
          /* Tag the current pixel the direction passes through horizontal. */
          if ((dy == 0) && (y >= 0))
            if (buffer[y][x] == boundry_tag)
              buffer[y][x] = double_boundry_tag; /* If already tagged, use */
            else                                 /* double_boundry_tag. */
              buffer[y][x] = boundry_tag;

          /* Rotate clockwise 45 degrees. */
          temp = dx - dy; dy = dx + dy; dx = temp;
          dx = (dx > 0) - (dx < 0);
          dy = (dy > 0) - (dy < 0);
     }
   else /* Otherwise, we found the next pixel in the trace. */
     {
          /* Move in this direction, expanding outer boundries if necessary. */
          x += dx; y += dy;
          if (x > right) right = x;
          if (y > bottom) bottom = y;
          if (x < left) left = x;
          if (y < top) top = y;

          /* Rotate counter-clockwise by 135 degrees to search for next pixel. */
          temp = dy - dx; dy = -dx - dy; dx = temp;
          dx = (dx > 0) - (dx < 0);
          dy = (dy > 0) - (dy < 0);
     }
  }
  /* Return a 1 if ending pixel is the same as starting pixel, otherwise 0. */
  return((x == start_x) && (y == start_y));
}

/***************************************************************************
  symbol_tracing_isolate:
     Given a source buffer and location of a reference black pixel,
     attempts to isolate a cluster of black pixels through contour tracing
     to form a symbol. If a cluster can be isolated that fits within the
     allowed size constraints, it is returned in the form of a symbol
     structure.  Otherwise, NULL is returned.

     Important side information is stored in the buffer so that large
     objects are handled correctly.  The caller should not directly access
     the buffer.  Instead, a seperate one-dimensional array of pixels,
     residue_line, should be maintained that contains only the first
     scan-line of the buffer.          It is automatically updated with the
     symbol_tracing_scroll function.  Detected symbols can be properly
     removed from the source image with the symbol_tracing_remove function.

     The following constants must be defined:
          MAX_SYMBOL_HEIGHT, MAX_SYMBOL_WIDTH => maximum allowed symbol size
          MIN_SYMBOL_HEIGHT, MIN_SYMBOL_WIDTH => minimum allowed symbol size
***************************************************************************/
SYMBOL *symbol_tracing_isolate(source_buffer, position)
byte **source_buffer;
int position;
{ SYMBOL *detected_symbol;
```

```
  int x,y,i;
  int in_symbol, valid_symbol, valid_pixel;

  /* Assign appropriate values to global variables referring to buffer. */
  buffer = source_buffer;
  maxX = buffer_maxX(buffer); nlines = buffer_nlines(buffer);
  prev_line = buffer_prev_line(buffer);

  /* Determine position of the pixel farthest to the right in black segment. */
  i = position;
  while ((i < (maxX - 1)) && (buffer[0][i] != WHITE))
    i++;
  if (buffer[0][i] == WHITE)
    i--;

  /* Determine if this is a valid starting pixel.  It must be BLACK (not tagged
     from a previous trace) and its upper right neighbor must be WHITE. */
  valid_pixel = (buffer[0][i] == BLACK);
  if (i < (maxX - 1))
    if (prev_line[i + 1] != WHITE)
      valid_pixel = 0;

  /* Proceed only if this is a valid starting pixel. */
  if (valid_pixel)
  {
    /* Trace.  Symbol is only valid if trace ends where it started. */
    valid_symbol = trace(i,0,BOUNDRY,DOUBLE_BOUNDRY);

    /* Determine if traced region is within the size contraints of a symbol. */
    valid_symbol &= ((bottom + 1) <= MAX_SYMBOL_HEIGHT) &&
                    ((right - left + 1) <= MAX_SYMBOL_WIDTH) &&
                    ((bottom + 1) >= MIN_SYMBOL_HEIGHT) &&
                    ((right - left + 1) >= MIN_SYMBOL_WIDTH);

    /* If region is a valid symbol, proceed. */
    if (valid_symbol)
    {
      /* Create a new symbol structure for this symbol. */
      detected_symbol = create_symbol(right-left+1, bottom+1, position-left);

      /* Copy the symbol to the bitmap field. */
      for (y = 0; y < detected_symbol->maxY; y++)
           for (x=left, in_symbol = 0; x <= right; x++)
             if (source_buffer[y][x] == BOUNDRY)
               { in_symbol = ! in_symbol;
                 detected_symbol->bitmap[y][x - left] = BLACK;
               }
             else if (source_buffer[y][x] == DOUBLE_BOUNDRY)
               detected_symbol->bitmap[y][x - left] = BLACK;
             else
               detected_symbol->bitmap[y][x - left] =
                 (in_symbol) ? source_buffer[y][x] : WHITE;
    }
    else /* otherwise, no detected symbol */
      detected_symbol = NULL;

    /* Retag all boundry pixels as NON_SYMBOL_BOUNDRY. */
    trace(i,0,NON_SYMBOL_BOUNDRY,NON_SYMBOL_BOUNDRY);
  }
  else
    detected_symbol = NULL;

  return(detected_symbol);
}

/*****************************************************************************
  symbol_tracing_remove:
      Given a detected symbol, its location, the source buffer, and a
```

```
    seperate residue scan-line array, erases the symbol from the source
    buffer and residue line.          This prevents the symbol from being detected
    again or from appearing in subsequent residue lines.
******************************************************************************/
void symbol_tracing_remove(detected_symbol, position,
                                source_buffer, residue_line)
SYMBOL *detected_symbol;
int position;
byte **source_buffer;
byte *residue_line;
{ int x,y;

  /* Erase symbol from source buffer. */
  for (y = 0; y < detected_symbol->maxY; y++)
    for (x = 0; x < detected_symbol->maxX; x++)
      if (detected_symbol->bitmap[y][x] == BLACK)
          source_buffer[y][x + position - detected_symbol->shift] = WHITE;

  /* Remove symbol from residue_line. */
  for (x = 0; x < detected_symbol->maxX; x++)
    if (detected_symbol->bitmap[0][x] == BLACK)
      residue_line[x + position - detected_symbol->shift] = WHITE;
}

/******************************************************************************
  symbol_tracing_scroll:
    Scrolls a buffer up by one scan-line and generates a new residue_line.
******************************************************************************/
void symbol_tracing_scroll(source_buffer,residue_line)
byte **source_buffer;
byte *residue_line;
{ int x;

  /* Determine buffer width and height. */
  maxX = buffer_maxX(source_buffer);
  nlines = buffer_nlines(source_buffer);

  /* Scroll buffer up one line. */
  scroll_buffer(source_buffer);

  /* Generate new residue_line. */
  for (x=0; x<maxX; x++)
    residue_line[x] = (source_buffer[0][x] == WHITE) ? WHITE : BLACK;
}
```

File `symbol_windowing.c`:

```
/******************************************************************************
Name:      symbol_windowing.c
Purpose:   Performs the symbol isolation stage of Content-Adaptive Facsimile
           Coding using the symbol windowing technique.

Contents:
  symbol_windowing_isolate(buffer,position,residue_line)
                          => Attempts to isolate a particular symbol given
                               a source buffer and position of reference pixel.
  symbol_windowing_remove(symbol,position,buffer,residue_line)
                          => Removes a detected symbol from the source buffer.
  symbol_windowing_scroll(buffer,residue_line)
                          => Scrolls the source buffer up one scan-line,
                               updating the residue scan-line.

Last modified on 12/20/93
******************************************************************************/
#include<stdio.h>
#include<stdlib.h>
```

```
#include"CAFC.h"
#include"PCX_util.h"
#include"PCX_buffer.h"
#include"symbol.h"

/***********************************************************************
  symbol_windowing_isolate:
     Given a source buffer and location of a reference black pixel,
     attempts to isolate a cluster of black pixels by systematically
     expanding a rectangular window until its border contains only
     white pixels. If a cluster can be isolated that fits within the
     allowed size constraints, it is returned in the form of a symbol
     structure.  Otherwise, NULL is returned.

     The caller should not directly access the buffer.          Instead, a
     seperate one-dimensional array of pixels, residue_line, should
     be maintained that contains only the first scan-line of the buffer.
     It is automatically updated with the symbol_tracing_scroll function.
     Detected symbols can be properly removed from the source image with
     the symbol_windowing_remove function.

     The following constants must be defined:
          MAX_SYMBOL_HEIGHT, MAX_SYMBOL_WIDTH => maximum allowed symbol size
          MIN_SYMBOL_HEIGHT, MIN_SYMBOL_WIDTH => minimum allowed symbol size
***********************************************************************/
SYMBOL *symbol_windowing_isolate(source_buffer, position)
byte **source_buffer;
int position;
{ SYMBOL *detected_symbol;
 byte *prev_line;
 int left, right, bottom;
 int left_clear, right_clear, top_clear, bottom_clear;
 int x, y;
 int maxX, nlines;

 /* Assign appropriate values to variables referring to buffer. */
 maxX = buffer_maxX(source_buffer); nlines = buffer_nlines(source_buffer);
 prev_line = buffer_prev_line(source_buffer);

 /* Initialize 3 edges of window (4th is the top, in prev_line). */
 left = position; right = position; bottom = 1;

 /* Initialize flags which indicate status of each border. */
 top_clear =          /* Top edge clear if pixel above is WHITE. */
   (prev_line[position] == WHITE);
 left_clear = 0;  /* The left edge is initially not clear. */
 right_clear = 0;  /* The right edge is initially not clear. */
 bottom_clear = 0; /* The bottom edge is initially not clear. */

 /* Iterate until edges are clear, top is unclear, or window is too big. */
 while ((! (left_clear && right_clear && top_clear && bottom_clear)) &&
         top_clear && ((right-left-1) < MAX_SYMBOL_WIDTH) &&
         (bottom < MAX_SYMBOL_HEIGHT))
 {
  /* Expand to the left until left is clear, top is not clear,
     or size limit is reached. */
  while ((top_clear) && (! left_clear) && (left >= 0) &&
          ((right-left-1) < MAX_SYMBOL_WIDTH))
  {
   /* Expand one pixel to the left. */
   left--;

   /* Determine if new left border is clear. */
   left_clear = 1;
   if (left > 0)
   {
       /* Left is not clear if any pixel in border is not WHITE. */
       for (y = 0; y <= bottom; y++)
```

```
          if (source_buffer[y][left] != WHITE)
            left_clear = 0;

        /* Check new pixel on top and bottom border and update status. */
        top_clear &= (prev_line[left] == WHITE);
        bottom_clear &= (source_buffer[bottom][left] == WHITE);
    }
  }

  /* Expand to the right until right is clear, top is not clear,
     or size limit is reached. */
  while ((top_clear) && (! right_clear) && (right < maxX) &&
        ((right-left-1) < MAX_SYMBOL_WIDTH))
  {
    /* Expand one pixel to the right. */
    right++;

    /* Determine if new right border is clear. */
    right_clear = 1;
    if (right < (maxX - 1))
    { for (y = 0; y <= bottom; y++)
          if (source_buffer[y][right] != WHITE)
            right_clear = 0;

        /* Check new pixel on top and bottom border and update status. */
        top_clear &= (prev_line[right] == WHITE);
        bottom_clear &= (source_buffer[bottom][right] == WHITE);
    }
  }

  /* Expand down until bottom is clear, sides are not clear,
     or size limit is reached. */
  if (top_clear)
    while ((! bottom_clear) && (bottom < (nlines - 1)) &&
           (bottom < MAX_SYMBOL_HEIGHT))
    { /* Expand one pixel down. */
        bottom++;

    /* Determine if new bottom, left, and right borders are clear. */
        bottom_clear = 1;
        for (x=left + 1; x < right; x++)
          bottom_clear &= (source_buffer[bottom][x] == WHITE);
        if (left >= 0)
          left_clear &= (source_buffer[bottom][left] == WHITE);
        if (right < maxX)
          right_clear &= (source_buffer[bottom][right] == WHITE);
    }
 }

 /* If all borders are clear and the window is big enough, make a symbol. */
 if (left_clear && right_clear && bottom_clear && top_clear &&
    ((right-left-1) >= MIN_SYMBOL_WIDTH) && (bottom >= MIN_SYMBOL_HEIGHT))
  {
    /* Create a new symbol structure for this symbol. */
    detected_symbol = create_symbol(right-left-1, bottom, position-left-1);

    /* Copy the symbol to the bitmap field. */
    for (y = 0; y < detected_symbol->maxY; y++)
        for (x=left+1; x < right; x++)
          detected_symbol->bitmap[y][x - left - 1] = source_buffer[y][x];
  }
 else /* otherwise, no detected symbol */
  detected_symbol = NULL;

 return(detected_symbol);
}

/****************************************************************************
```

```
  symbol_windowing_remove:
     Given a detected symbol, its location, the source buffer, and a
     seperate residue scan-line array, erases the symbol from the source
     buffer and residue line.        This prevents the symbol from being detected
     again or from appearing in subsequent residue lines.
*****************************************************************************/
void symbol_windowing_remove(detected_symbol, position,
                             source_buffer, residue_line)
SYMBOL *detected_symbol;
int position;
byte **source_buffer;
byte *residue_line;
{ int x,y;

  /* Erase symbol from source buffer. */
  for (y = 0; y < detected_symbol->maxY; y++)
    for (x = 0; x < detected_symbol->maxX; x++)
      source_buffer[y][x + position - detected_symbol->shift] = WHITE;

  /* Remove symbol from residue_line. */
  for (x = 0; x < detected_symbol->maxX; x++)
    residue_line[x + position - detected_symbol->shift] = WHITE;
}

/*****************************************************************************
  symbol_windowing_scroll:
     Scrolls a buffer up by one scan-line and generates a new residue_line.
*****************************************************************************/
void symbol_windowing_scroll(source_buffer,residue_line)
byte **source_buffer;
byte *residue_line;
{ int x, maxX;

  /* Determine buffer width. */
  maxX = buffer_maxX(source_buffer);

  /* Scroll buffer up one line. */
  scroll_buffer(source_buffer);

  /* Generate new residue_line. */
  for (x=0; x < maxX; x++)
    residue_line[x] = source_buffer[0][x];
}
```

E.8 Source Code – Symbol Manipulation

File `symbol.c`:

```
/*****************************************************************************
Name:     symbol.c
Purpose:  This file contains routines for manipulating symbol structures.

The SYMBOL structure is defined in symbol.h

Contents:
  create_symbol(maxX, maxY, shift) => Create new symbol structure.
  free_symbol(old_symbol)           => Deallocate memory used by symbol.
  display_symbol(symbol)            => Displays ASCII version of symbol.

Last modified on 12/30/93
*****************************************************************************/

#include<stdio.h>
```

```
#include<stdlib.h>
#include"CAFC.h"
#include"PCX_util.h"
#include"symbol.h"

/*****************************************************************************
  create_symbol:  Creates new symbol structure and initializes its fields
                  with the specified dimensions and horizontal shift.  Returns
                  a pointer to the symbol.
*****************************************************************************/
SYMBOL *create_symbol(maxX, maxY, shift)
int maxX, maxY, shift;
{ SYMBOL *new_symbol;
  int i;

  /* Create a new symbol structure. */
  new_symbol = (SYMBOL *) malloc(sizeof(SYMBOL));

  /* Create symbol bitmap. */
  new_symbol->bitmap = (pixel **) malloc(maxY * sizeof(pixel *));
  for (i = 0; i < maxY; i++)
    new_symbol->bitmap[i] = (pixel *) malloc(maxX * sizeof(pixel));

  /* Set dimensions. */
  new_symbol->maxX = maxX;
  new_symbol->maxY = maxY;
  new_symbol->shift = shift;

  /* Initially, no features are known. */
  for (i=0; i<NFEATURES; i++)
    new_symbol->f_known[i] = 0;

  return(new_symbol);
}

/*****************************************************************************
  free_symbol:     Deallocate memory used by symbol.
*****************************************************************************/
void free_symbol(old_symbol)
SYMBOL *old_symbol;
{ int i;

  /* Free symbol bitmap. */
  for (i=0; i < old_symbol->maxY; i++)
    free(old_symbol->bitmap[i]);
  free(old_symbol->bitmap);

  /* Free the symbol structure. */
  free(old_symbol);
}
```

File `symbol.h`:

```
/*****************************************************************************
Name:     symbol.h
Purpose:  This header file contains definitions used by the symbol managment
          routines (symbol.c).

Last Modified on 12/30/93
*****************************************************************************/
typedef unsigned char pixel;

/* definition for symbol type */
typedef struct
  { pixel **bitmap; /* bitmap containing pixel data */
    int maxX, maxY; /* dimensions of bitmap */
```

```
  int shift;          /* horizontal position of first black pixel in first row */
  int features[NFEATURES]; /* its features */
  int f_known[NFEATURES];  /* 1 for features that are known */
 } SYMBOL;

/* declarations for symbol managment routines */
SYMBOL *create_symbol();
void free_symbol();
```

E.9 Source Code – Arithmetic Coding/Decoding

File AC.c:

```
/*************************************************************************
Name:     AC.c
Purpose:  This file contains routines to perform arithmetic coding and
          decoding.

          The header file AC.h must be included in any program that uses
          these routines. The coding model, represented by an AC_MODEL
          structure type, contains the number of occurrences of each
          possible element (symbol). The model can be changed during
          the encoding process, but the same changes must be made at
          the decoder to be consistent.

          The AC_ENCODER and AC_DECODER structure types contain all of the
          necessary state variables for an encoding or decoding process.
          This way, multiple encoding and decoding processes can be managed
          seperately and simultaneously.

Contents:
  initialize_model(model)              => Initializes new coding model.
  add_element_to_model(model)          => Adds new element to model with
                                            zero occurrences.
  update_model(model,element,count)    => Increases number of occurrences of
                                            an element in model by count.
  open_AC_encoder(encoder,output_func) => Begin a new encoding process.
  open_AC_decoder(decoder,input_func)  => Begin a new decoding process.
  encode_element(encoder,model,element)=> Encode an element.
  decode_element(decoder,model,element)=> Obtain next decoded element.
  close_AC_encoder(encoder)            => End an encoding process.
  close_AC_decoder(decoder)            => End a decoding process.

Last modified on 12/21/93
*************************************************************************/

#include<stdio.h>
#include<string.h>
#include"AC.h"


/*************************************************************************
  initialize_model: Takes as an argument a pointer to a AC_MODEL structure
                    to initialize. Prepares model for use by encoder or
                    decoder, setting the total number of elements to zero.
                    Returns a pointer to the model.
*************************************************************************/
AC_MODEL *initialize_model(model)
AC_MODEL *model;
{ model->n_elements = 0;  /* Initially zero elements. */
  model->totals = NULL;   /* Initially, no counts. */
  return(model);
}
```

```
/*****************************************************************************
  add_element_to_model:
          Adds a new element to the specified AC_MODEL structure, initially
          with zero occurrences.      The count should be increased with
          update_model before the model is used again.  Returns an integer
          that should be used to refer to this element on all subsequent
          encoding and decoding operations.
*****************************************************************************/
int add_element_to_model(model)
AC_MODEL *model;
{
  /* Increment the number of elements and the size of the totals array. */
  model->n_elements++;
  model->totals = (unsigned long *)
    realloc(model->totals,model->n_elements*sizeof(unsigned long));

  /* Set the number of occurrences of this element to zero. */
  if (model->n_elements == 1)  /* If this is first element, total = 0. */
    model->totals[0] = 0;
  else    /* Otherwise, set total to same as previous element. */
    model->totals[model->n_elements-1] = model->totals[model->n_elements-2];

  return(model->n_elements - 1);
}


/*****************************************************************************
  update_model:    Increase the number of occurrences of the specified element
                   in the specified model by specified count.
*****************************************************************************/
AC_MODEL *update_model(model, element, count)
AC_MODEL *model;
int element;
int count;
{ int i, new_count, total;

  /* Increase total for specified element and all subsequent elements. */
  for (i=element; i < model->n_elements; i++)
    model->totals[i] += count;

  /* If the new total is too high, scale back all counts. */
  while (model->totals[model->n_elements - 1] >= 16384)
    {
     /* Divide all counts by 2 to reduce total. */
     total = 0;
     count = model->totals[0];
     for (i=0; i < model->n_elements; i++)
          { new_count = count / 2;
            if (new_count == 0) /* Make sure that all counts are positive. */
              new_count = 1;
            total += new_count;
            if (i < model->n_elements)
              count = model->totals[i + 1] - model->totals[i];
            model->totals[i] = total;
          }
    }

  return(model);
}


/*****************************************************************************
  open_AC_encoder:  Begin a new encoding process by initializing the state
                    variables in the specified AC_ENCODER structure.  All
                    encoded bits are individually passed to the function
                    provided when they become available.
*****************************************************************************/
```

```
void open_AC_encoder(encoder,output_func)
AC_ENCODER *encoder;
void (*output_func) ();
{
  /* Initizlize to full 16-bit range with zero underflow bits. */
  encoder->low_range = 0x0000;    /* 0b0000000000000000 */
  encoder->high_range = 0xFFFF; /* 0b1111111111111111 */
  encoder->underflow = 0;

  /* Store output function pointer. */
  encoder->output_func = output_func;
}

/****************************************************************************
  open_AC_decoder:  Begin a new decoding process by initializing the state
                    variables in the specified AC_DECODER structure.  All
                    encoded bits are obtained from the function provided when
                    they are needed.
****************************************************************************/
void open_AC_decoder(decoder,input_func)
AC_DECODER *decoder;
int (*input_func) ();
{ int i;

  /* Initizlize to full 16-bit range. */
  decoder->low_range = 0x0000;    /* 0b0000000000000000 */
  decoder->high_range = 0xFFFF; /* 0b1111111111111111 */

  /* Store input function pointer. */
  decoder->input_func = input_func;

  /* Fill encoded_bits buffer with encoded bits. */
  decoder->encoded_bits = 0;
  for (i=0; i<16; i++)
    decoder->encoded_bits = (decoder->encoded_bits << 1) + (*input_func) ();
}

/****************************************************************************
  encode_element:  Given an AC_ENCODER structure, an AC_MODEL structure, and
                   an integer referring to an element in the model, encodes
                   the element.
****************************************************************************/
void encode_element(encoder, model, element)
AC_ENCODER *encoder;
AC_MODEL *model;
int element;
{ unsigned long range, base, total_count;

  /* Determine current base and range of encoder. */
  base = encoder->low_range;
  range = encoder->high_range - encoder->low_range + 1;

  /* Compute new range for encoder based upon element and model. */
  total_count = model->totals[model->n_elements - 1];
  if (element > 0)
    encoder->low_range =
      base + range * model->totals[element - 1] / total_count;
  encoder->high_range =
    base + range * model->totals[element] / total_count - 1;

  /* Shift out any matching upper bits in low and high ends of range. */
  while ((encoder->low_range >> 15) == (encoder->high_range >> 15))
  {
    /* Pass output bit to output_func. */
    (*encoder->output_func) (encoder->high_range >> 15);

    /* Pass any underflow bits to output_func. */
    while (encoder->underflow > 0)
```

```
    { (*encoder->output_func) (1 - (encoder->high_range >> 15));
        encoder->underflow--;
    }
   /* Shift low_range and high_range to the left to remove encoded bit. */
   encoder->high_range =
     ((encoder->high_range & 0x7FFF /* 0b0111111111111111 */) << 1) + 1;
   encoder->low_range =
     ((encoder->low_range & 0x7FFF /* 0b0111111111111111 */) << 1);
 }

 /* Determine if there is underflow. */
 while (((encoder->low_range >> 14) == 1) &&
         ((encoder->high_range >> 14) == 2))
 {
   /* If so, increment underflow count. */
   encoder->underflow++;

   /* Shift out underflow bits in low and high ends of range. */
   encoder->high_range =
     ((encoder->high_range & 0x3FFF /* 0b0011111111111111 */) << 1) |
        0x8001; /* 0b1000000000000001 */
   encoder->low_range =
     ((encoder->low_range & 0x3FFF /* 0b0011111111111111 */) << 1);
 }
}

/****************************************************************************
  decode_element:  Given an AC_DECODER structure and an AC_MODEL structure,
                   returns an integer referring to the next encoded element
                   in the input stream.
****************************************************************************/
int decode_element(decoder, model)
AC_DECODER *decoder;
AC_MODEL *model;
{ unsigned long range, base;
  int count, total_count;
  int element;

  /* Determine current base and range of encoder. */
  base = decoder->low_range;
  range = decoder->high_range - decoder->low_range + 1;

  /* Scan through element ranges to determine encoded element. */
  total_count = model->totals[model->n_elements - 1];
  count =
    ((decoder->encoded_bits - base + 1) * total_count + range - 1) / range;
  element = 0;
  while (count > model->totals[element])
    element++;

  /* Compute new range for decoder base upon element and model. */
  if (element > 0)
    decoder->low_range =
      base + range * model->totals[element - 1] / total_count;
  decoder->high_range =
    base + range * model->totals[element] / total_count - 1;

  /* If upper bits match in low and high ends of range, shift them out
     and shift in new encoded bits. */
  while ((decoder->low_range >> 15) == (decoder->high_range >> 15))
  {
    /* Shift low_range and high_range to the left to remove encoded bit. */
    decoder->high_range =
      ((decoder->high_range & 0x7FFF /* 0b0111111111111111 */) << 1) + 1;
    decoder->low_range =
      ((decoder->low_range & 0x7FFF /* 0b0111111111111111 */) << 1);

    /* Shift out encoded bit in buffer and shift in new one. */
```

```
    decoder->encoded_bits =
      ((decoder->encoded_bits & 0x7FFF /* 0b0111111111111111 */) << 1) +
          (*decoder->input_func) ();
  }

  /* Determine if there is underflow. */
  while (((decoder->low_range >> 14) == 1) &&
         ((decoder->high_range >> 14) == 2))
  {
    /* If so, shift out underflow bit. */
    decoder->high_range =
      ((decoder->high_range & 0x3FFF /* 0b0011111111111111 */) << 1) |
          0x8001; /* 0b1000000000000001 */
    decoder->low_range =
      ((decoder->low_range & 0x3FFF /* 0b0011111111111111 */) << 1);

    /* Shift out underflow bit in buffer and shift in new encoded bit. */
    decoder->encoded_bits =
      (((decoder->encoded_bits & 0x3FFF /* 0b0011111111111111 */) << 1) |
       (decoder->encoded_bits & 0x8000 /* 0b1000000000000000 */)) +
          (*decoder->input_func) ();
  }
  /* Return the decoded element. */
  return(element);
}

/**************************************************************************
  close_AC_encoder: End a coding process. Flushes out remaining bits in
                    specified AC_ENCODER structure.
**************************************************************************/
void close_AC_encoder(encoder)
AC_ENCODER *encoder;
{
  /* Output high bit (bit 15) in high_range. */
  (*encoder->output_func) (encoder->high_range >> 15);

  /* Output any underflow bits */
  while (encoder->underflow > 0)
  { (*encoder->output_func) (1 - (encoder->high_range >> 15));
    encoder->underflow--;
  }
  /* Output second highest bit (bit 14) in high_range. */
  (*encoder->output_func) ((encoder->high_range >> 14) & 1);
}

/**************************************************************************
  close_AC_decoder: End a decoding process.
**************************************************************************/
void close_AC_decoder(decoder)
AC_DECODER *decoder;
{ /* Nothing to do! Routine provided for completeness. */
}
```

File `AC.h`:

```
/**************************************************************************
Name:    AC.h
Purpose: This include file contains constants and parameters used by
         the arithmetic coding/decoding routines (AC.c).

Last Modified on 12/21/93
**************************************************************************/

/* Define structure for coding model. */
```

```
typedef struct
  { unsigned long *totals; /* Array containing total number of occurrences of
                              all elements less than or equal to the array
                              index.  Used as upper value in range for
                              encoding and decoding. */
    int n_elements;  /* Total number of elements in model. */
  } AC_MODEL;

/* Define structure containing all state variables for arithmetic encoder. */
typedef struct
  { unsigned long low_range, high_range; /* encoding range */
    int underflow;                        /* number of underflow bits */
    void (*output_func) ();               /* function to absorb encoded bits */
  } AC_ENCODER;

/* Define structure containing all state variables for arithmetic decoder. */
typedef struct
  { unsigned long low_range, high_range; /* decoding range */
    int (*input_func) ();                 /* function to provide encoded bits */
    unsigned long encoded_bits;           /* buffer containing encoded bits */
  } AC_DECODER;

/* declarations for the arithmetic coding and decoding routines */
AC_MODEL *initialize_model();
int add_element_to_model();
AC_MODEL *update_model();
void open_AC_encoder();
void open_AC_decoder();
void encode_element();
int decode_element();
void close_AC_encoder();
void close_AC_decoder();
```