JPL PUBLICATION 83-64

The Development of Efficient Coding for an Electronic Mail System

Robert F. Rice

July 15, 1983

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL PUBLICATION 83-64

The Development of Efficient Coding for an Electronic Mail System

Robert F. Rice

July 15, 1983

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The work described in this publication was performed by the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the United States Postal Service through an agreement with the National Aeronautics and Space Administration. Publication support was sponsored by the National Aeronautics and Space Administration.

ABSTRACT

This report presents the results of a one-year study to investigate and develop techniques for efficiently representing scanned electronic documents. Major results include the definition and preliminary performance results of a Universal System for Efficient Electronic Mail (USEEM), offering a potential order of magnitude improvement over standard facsimile techniques for representing textual material.

ACKNOWLEDGEMENT

The author wishes to thank A. Kegel, J. McGinn, N. M. Mikhael and A. Tersoff of the United States Postal Service for their able managerial and technical guidance.

He also wishes to offer sincere thanks to colleague Dr. Jun-Ji Lee for his crucial technical inputs and extensive software development. The competent software support of Cathy Schindel, Harish Chopra, and Alan Schlutsmeyer is gratefully acknowledged.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Tables

TABLE OF CONTENTS (Continued)

Figures

THE DEVELOPMENT OF EFFICIENT CODING FOR AN ELECTRONIC MAIL SYSTEM

I. INTRODUCTION

PROGRAM OVERVIEW

This paper presents the results of research by the Information Processing Research Group of the Jet Propulsion Laboratory for the U.S. Postal Service (USPS) from April 1980 to April 1981. This research was directed towards the development of efficient preprocessing and coding algorithms for binary facsimile imagery.

Objectives

The program objectives at inception were to provide algorithm specifications, characteristics, and functional implementations which would lead to the realization of a prototype model, by others, for test and evaluation. Such a model would later be incorporated within a planned USPS Electronic Message Service System. The stated algorithm performance goal was to achieve a 2:1 improvement over the CCITT (International Consultative Committee for Telephone and Telegraph) standard technique for one-dimensional facsimile coding.

Redirection. A December 1980 interim presentation of the results of algorithm investigations introduced a new concept called a "Universal System for Efficient Electronic Mail" (USEEM). These initial results promised such overwhelming potential for improved performance that the USPS redirected JPL to concentrate its remaining efforts on defining USEEM further. Functional implementation diagrams, which would allow a prototype system to be built, would instead be completed during anticipated follow-on programs.

SUMMARY OF MAJOR RESULTS †

Noiseless Coding

A constraint that binary facsimile images must be reconstructed precisely (i.e., reversible coding, noiseless coding) bit-for-bit, will limit achievable compression factors to the following:

---

† 200 lines/inch compression factors, higher for 300 lines/inch.

- Less than 10:1 on text. (1)

- An average of 20:1 over all documents. (2)

Further, the stated goal of a 2:1 improvement in performance over the CCITT one-dimensional standard is not generally possible on textual material. While JPL noiseless coding techniques combined with the CCITT standard one-dimensional run-length preprocessing did improve performance, major gains, as in (1) and (2), can only be achieved by utilizing two-dimensional preprocessing techniques. Subtle variations in performance exist among all the best techniques but they are really quite minor. They all fit the summary conclusions in (1) and (2).

USEEM

A concept for a Universal System for Efficient Electronic Mail (USEEM) was conceived which:

- Can perform the noiseless coding function as above, but by: (3)
  - removing the restrictive bit-for-bit noiseless constraint,
  - utilizing unsupervised-character-recognition, and
  - utilizing adaptive noiseless coding of "text".

This concept offers **the potential for**:

- 100:1 compression on dense text (higher on lighter documents), (4)
- improved reproduced quality, (5)
- compatibility with direct entry electronic mail with (6)
  - automated translation to word processor form, and
  - efficient coding of direct entry text.

In addition, the further development and subsequent implementation of USEEM could be scheduled in states corresponding to distinct modular increments to performance. (7)

Observe that the major result of (4) in essence means that USEEM offers the potential for an **order of magnitude** reduction in the transmission and buffering requirements for a predominant component of digitized electronic mail. †

Cost Impact[1]

It has been estimated that a future facsimile-based USPS electronic mail system would handle between 3 and 25 billion messages per year. Estimates of high-speed storage and communication channel annual costs for standard facsimile compression techniques and USEEM are compared below in Table 1 and Table 2 assuming 1978 dollars.

Table 1. Annual Cost Comparison 3 Billion Messages/Year.

	Annual Costs ($ Million 1978)		
	High-Speed Storage	Communication Channel	Total
Standard 2-Dimensional Compression	1.3	7.0	8.3
USEEM	0.4	1.2	1.6
Cost Reduction	0.9	5.8	6.7

† This is roughly a factor of "twenty" improvement over the CCITT one-dimensional standard.

Table 2. Annual Cost Comparison 25 Billion Messages/Year.

	Annual Costs ($ Million 1978)		
	High-Speed Storage	Communication Channel	Total
Standard 2-Dimensional Compression	6.1	37.0	43.1
USEEM	1.9	6.0	7.9
Cost Reduction	4.2	31.0	35.2

Note that in 1990 dollars (a time frame closer to a full USEEM implementation) the savings at 25 billion messages per year would be over **$1 billion** in a ten-year period.

II. NOISELESS FACSIMILE COMPRESSION

This section investigates the relative and absolute performance of various alternative algorithms designed to provide efficient **noiseless** representations of the binary images which are characteristic of electronic mail systems. The results of extensive simulations using both familiar, well documented techniques as well as new approaches are preceded by some necessary background development. A summary of major observations is given at the end of this section.

BACKGROUND

A data sequence can be said to be coded noiselessly when the original data sequence can be recovered, without error, from the coded one. Noiseless coding to achieve "data compression" can be applied to advantage on virtually any data source which exhibits statistical characteristics having memory and/or non-uniform symbol probability distributions. For example, grey scale images have memory because adjacent picture elements (pixels) tend to be similar. Consequently, a sequence of differences between adjacent pixels will be distributed about zero in a unimodal fashion. Use of a variable length code allows shorter codewords to be assigned to the smaller difference values which occur more often. On the average the number of bits required may be much less than if a fixed length representation had been used. These same two steps of preprocessing to make use of source memory (differences here) and assigning codewords apply to numerous problems.

Application of noiseless coding to images which only have two brightness values is known as predictive facsimile data compression.

JPL Noiseless Coding Software

Other JPL work resulted in general purpose adaptive variable length coding techniques which were expected to be applicable to the facsimile problem when combined with appropriate preprocessing algorithms.[2]-[3] To facilitate investigations of this possibility, JPL placed these variable length coding techniques in an easily used Fortran software package. A description is provided in Ref. 4 and Appendix A.

Measures of Performance

The "entropy" of a probability distribution with values $p_0, p_1, p_2, \ldots p_q$ is given by

$$H = -\sum_i p_i \log_2 p_i \qquad (8)$$

and represents an upper bound to the average performance of any noiseless coder in representing a long sequence of symbols which are generated **independently** from some **unchanging** data source with these symbol probability values. If the conditions of independence and stationarity (unchanging statistics) are met, then H in (8) represents the best that any noiseless coding operation can do. However, when these conditions are not met, performance under H may be possible. In this case, H is **a practical guide** to good performance of a coder which does not utilize knowledge of this source memory or non-stationarity to its advantage. Hence, using H as an absolute bound to performance should be viewed with caution.

Example. As an example, a binary image whose right half is white and left half black has the same **average** symbol probability distribution ($p_0 = \frac{1}{2}, p_1 = \frac{1}{2}$) as an image of randomly occurring black and white spots. An entropy calculation based on this distribution would yield an entropy of 1 bit/sample. However, the first source could be coded with nearly zero bits in each half by recognizing the memory and/or non-stationarity in this source (the right half has $p_0 = 1, p_1 = 0, H = 0$; and the left half has $p_0 = 0, p_1 = 1, H = 0$).

Entropies in this task. In the noiseless coding of binary images containing text and graphics we will use entropy measures for average symbol distributions which emanate from preprocessors which have already made use of source memory. The entropies then represent good guides to performance assuming that particular preprocessor is used.

For example run-length preprocessing is a widely used and effective technique for facsimile data compression[5]-[6]. **One-dimensional run-length entropies** will be used here as a guide to facsimile compression ratios for algorithms which do not make use of information from adjacent lines. We will also quote entropy values associated with the output of two-dimensional run-length preprocessors of binary data. Finally, to deal with efficient representations of textual materials we will use entropies based on distributions of characters, etc. In each case the entropies are guides to good performance if the preprocessing is followed by appropriate variable length coding. This general subject is more thoroughly treated in Refs. 2-4 and Appendix A.

Test Sets

The test sets used to investigate performance included:

a) Four pictures (Fig. 1) from the U.S. Postal Service, digitized to both 200 and 300 points/inch (p/inch);

b) the standard eight document CCITT test set (Figs. 2 and 3) digitized to 200 p/inch[7];

c) several pages of typical word processor output, digitized to 200 p/inch by the Naval Ocean System Center.

INVESTIGATIONS OF DOCUMENTED ALGORITHMS

CCITT 1-D Standard[7]

In this standardized approach run-lengths are generated for sequences of black and white runs along individual lines, as illustrated in Fig. 4.

The black runs are coded with a separate standardized variable length code which has been optimized with a goal of efficiently representing black runs at an entropy of H_b bits/run. The white runs are also coded with a separately optimized code with a goal of H_w bits/run. The code words are concatenated (* in Fig. 4) to produce an overall output sequence.

The "one-dimensional (1-D) run-length entropy" for the binary source itself is given by

$$H_{1D} = \frac{\text{Average Bits/Run}}{\text{Average Input Pixels/Run}} = \frac{H_w + H_b}{r_w + r_b} \text{ bits/pixel} \qquad (9)$$

where r_w and r_b are the average run-lengths for white and black runs respectively.

Bell Labs Markov Prediction[8]

The basic Markov prediction approach by Bell Labs is illustrated in Fig. 5.

Coding is accomplished by 1-D run-length coding of the error sequence resulting from this process and some useful tricks in ordering the runs.[8]

ORIGINAL PAGE IS
OF POOR QUALITY

THE GEORGE WASHINGTON UNIVERSITY

School of Engineering and Applied Science
Continuing Engineering Education

July 13, 1976

SPECIAL NOTICE

The George Washington University, in cooperation with the U.S. Department of Labor, is initiating a unique program which may be of interest to members of your family. Designed primarily for unemployed high school graduates interested in the Medical and Health Care Sciences, but with no previous experience, this program will combine formal instruction with practical job experience. The areas of concentration available to the student are

* Biomedical Engineering Technician
* Medical and Health Care Institutions Safety Technician
* Automated Medical Information Systems (AMIS) Technician

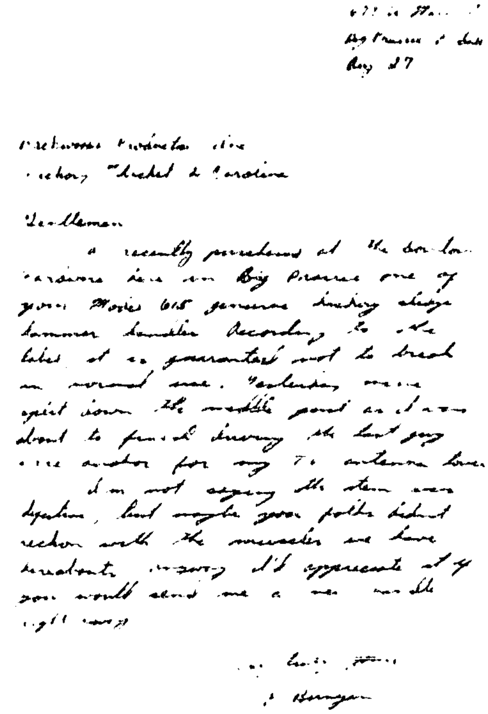
Assistance from the Department of Labor has enabled the University to provide this opportunity at approximately one-half tuition. In addition, the University will find the students jobs in government and industry for the period of enrollment. Arrangements are now being made so that the student will be employed for the first four months prior to beginning his studies at GWU. This initial employment phase will start September 7, 1976. The student will alternate periods of work and study (four months each) through a total of eight periods.

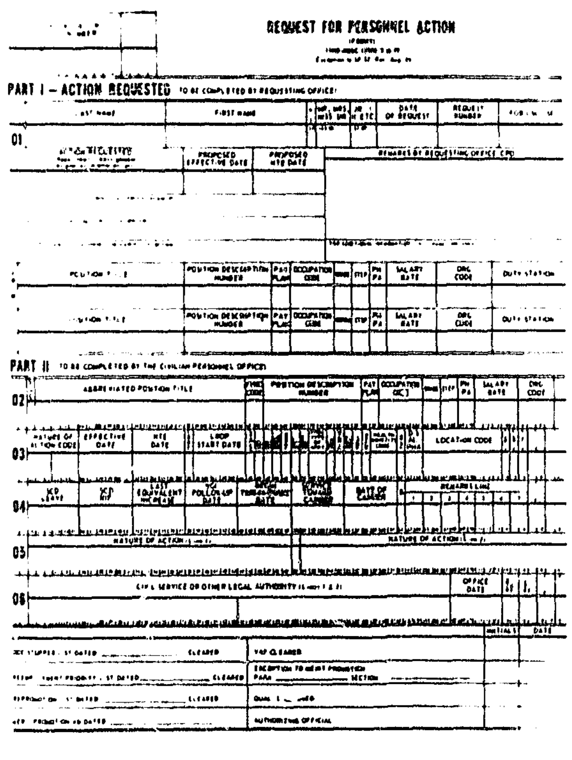
Should your son or daughter wish additional information on this Cooperative Education and Training Program, Mr. Goldman's office should be contacted at telephone number 676-6430. This office will be able to provide you explanatory brochures, applications for enrollment and answer questions pertaining to the program.

If you do not have a son or daughter interested in the above mentioned programs, you may wish to bring this notice to the attention of one of your friends who does.

We appreciate the interest you have shown in our Continuing Engineering Education Program, and hope that the George Washington University may continue to be of assistance to you in meeting the educational requirements of you and your family.

J. E. Mansfield, Director
Continuing Engineering Education





22 February 1976

Mr. W. J. Miller, Director
Office of Advanced Mail Systems Development
11711 Parklawn Avenue
Rockville, Maryland 20852

Gentlemen:

This is a sample of the letter we propose to use as a "standard" for imaging experiments at NELC, San Diego. It was made on a Wang System 1222 Dual Cassette Typewriter which consists of a modified IBM Selectric typewriter, two cassette holders, and a magnetic core memory capable of storing pages of data such as this letter. The cassette tapes are being made to store the data for each character in United States of America Standard Code for Information Interchange (USASCII) format. This is a standard seven bit binary code for each character which is widely used in industry. In USASCII form this page as written can be exactly defined by 15099 bits of data (excluding signature, logo or header information). When scanned at 200 x 200 picture elements per inch with six bits per element for grey scale the page is defined by 22,440,000 bits.

By recording the contents of this letter on cassette tape, it is possible to reproduce a quantity of duplicate originals, all nominally exactly the same. Since the typewriter is an IBM Selectric it is also possible to change the type font without changing the message. It is also possible to change ribbons (a five- or ten-minute process) to yield copies of differing colors. It is of course possible to write on all textures, colors and weights of paper with or without letter head. It will also allow copies of this text to be analyzed both with and without signatures of various colors.

This ability to provide complete parameter selection and consistency control for analysis of thresholds, contrasts, color separation, compressability coefficients, and character fonts will be of great benefit in quantifying the requirements of U. S. Postal Service Scanner technology.

Frank Martin
NELC Code 3100
Problem N451

Fig. 1. Postal Service Test Set.

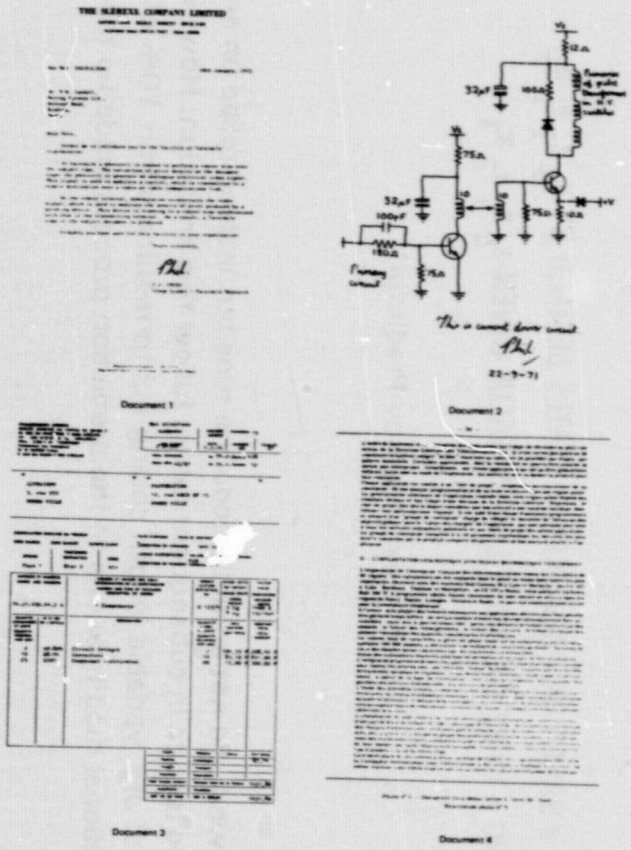


Fig. 2. CCITT Test Set, Images 1-4.

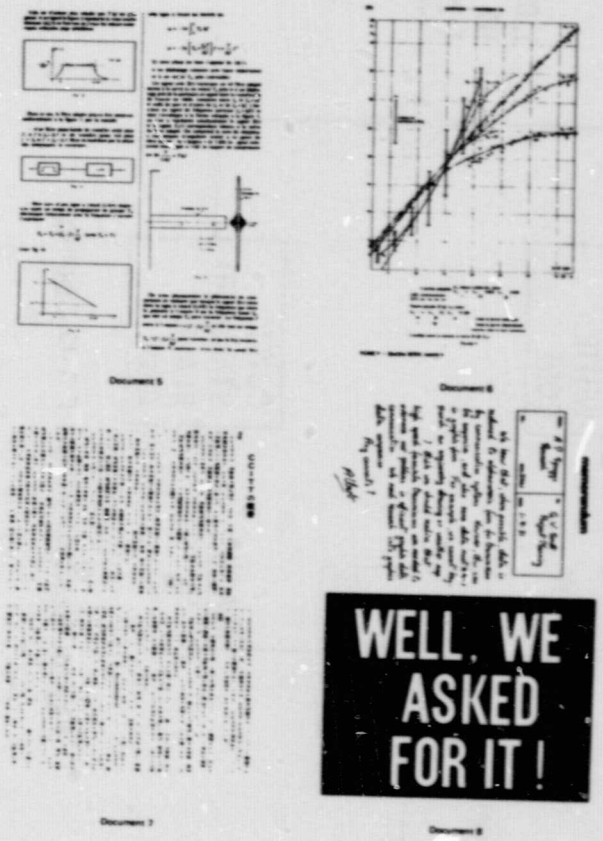


Fig. 3. CCITT Test Set, Images 5-8.

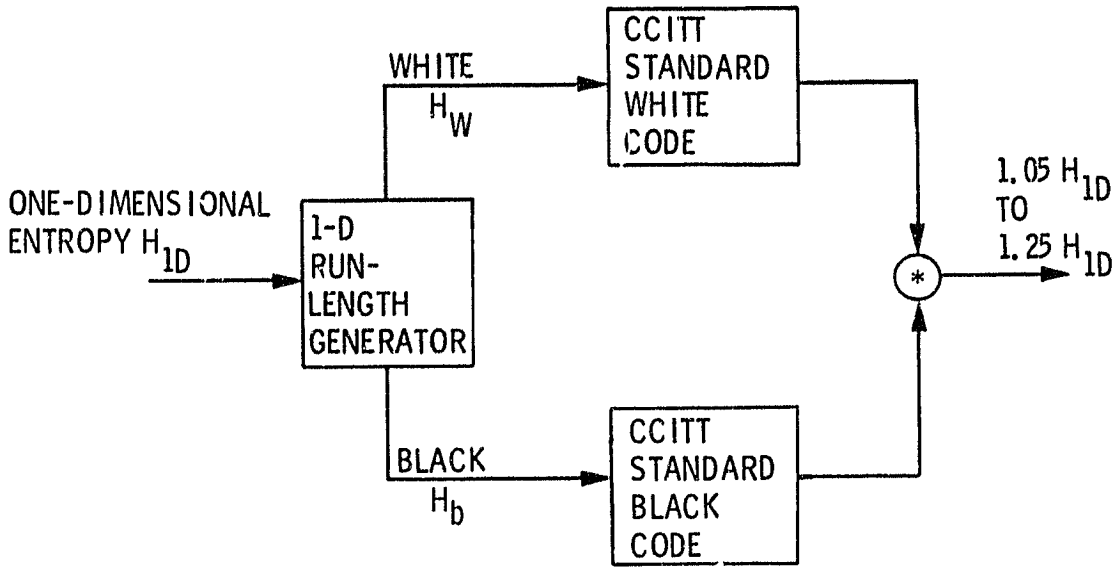
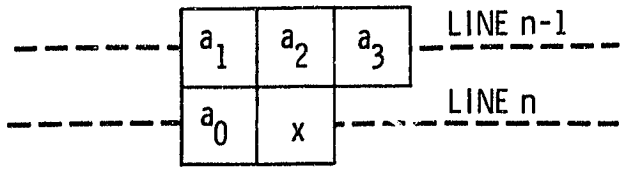


Fig. 4. Standard CCITT 1-D Coding.



PREDICT x BASED ON PREVIOUS 16 STATES DEFINED BY $a_0\ a_1\ a_2\ a_3$,
CHOOSING x TO BE THE MOST LIKELY VALUE GIVEN $a_0\ a_1\ a_2\ a_3$

Fig. 5. Basic Markov Predictor.

The Bell Labs work did not actually code the runs but instead relied on entropies as an estimate. Stated results in subsequent tables follow this limitation. However, appropriate application of JPL variable length coding led to results within 10% of these entropy values. Additional modifications to the prediction process made up the additional 10%.

IBM Dual Mode[9]

The basic idea of this approach is to make use of the observation that a run in one line will tend to be closely followed in the next. The error in this prediction can usually be variable length coded to advantage. When it can't, the system defaults to a standard, one-dimensional, run-length mode.

Performance Runs

Tables 3 – 6 show the results of applying these algorithms to the Postal Service test set of 4 images using both horizontal and vertical scan lines at 200 and 300 points/inch.

ADDITIONAL APPROACHES

Replacing the CCITT Standard Code

A direct replacement of the CCITT one-dimensional, standard, variable length code (see Fig. 4) with an appropriate JPL algorithm (Appendix A) yielded average performance lying half-way between one-dimensional run-length entropies and the corresponding performance of the CCITT code.

Histogram update. The input to a JPL variable length coder is the integers 0, 1, 2, ... which is the result of mapping the most likely run-lengths into the smaller integers. The condition

$$p_0 \geq p_1 \geq p_2 \cdots \tag{10}$$

should be maintained to assure the best variable length coding performance. This can be done adaptively by adjusting the run-length mapping sample-by-sample to reflect any changes in run-length probabilities. This adjustment can be accomplished without any additional data rate requirements since the necessary "histogram update" needs only previous samples. The result of this configuration, illustrated in Fig. 6, was performance uniformly within

$$2\% \text{ of the 1-D Entropy.} \tag{11}$$

Table 3. Baseline Performance: 200 Points/Inch, Vertical Scan.

IMAGE NAME	SIZE LINES X PIXEL	ALGORITHM PERFORMANCE IN BITS/PIXEL (B/P)							
				ENTROPY BELL LABS ALGORITHMS					
		1-D RUN-LENGTH ENTROPY	CCITT RUN-LENGTH CODING	1-D RUN-LENGTH CODING OF ORDERING WITH REF. TO PREV. LINE	1-D RUN-LENGTH CODING OF 2-STATE ORDERED PRED. ERR.	1-D RUN-LENGTH CODING OF 16-STATE PREDICTED ERRORS	1-D RUN-LENGTH CODING OF 16-STATE ORDERED PRED. ERR.	IBM DUAL MODE K = 4	IBM DUAL MODE K = ∞
WJM TYPED	1700 x 2200	0.091	0.114	0.078	0.072	0.085	0.066	0.081	0.063
GWU TYPED	1700 x 2200	0.116	0.139	0.088	0.081	0.075	0.076	0.095	0.073
BUNYAN WRITTEN	1700 x 2200	0.069	0.089	0.060	0.053	0.056	0.037	0.056	0.038
FORM	1700 x 2200	0.197	0.238	0.101	0.093	0.097	0.093	0.138	0.096
AVERAGE B/P		0.118	0.145	0.082	0.075	0.078	0.068	0.092	0.067
AVERAGE COMPRESSION FACTOR			6.9	12.2	13.3	12.8	14.7	10.9	14.9

Table 4. Baseline Performance: 200 Points/Inch, Horizontal Scan.

IMAGE NAME	SIZE LINES X PIXEL	ALGORITHM PERFORMANCE IN BITS/PIXEL (B/P)							
				ENTROPY BELL LABS ALGORITHM					
		1-D RUN-LENGTH ENTROPY	CCITT RUN-LENGTH CODING	1-D RUN-LENGTH CODING OF ORDERING WITH REF. TO PREV. LINE	1-D RUN-LENGTH CODING OF 2-STATE ORDERED PRED. ERR.	1-D RUN-LENGTH CODING OF 16-STATE PREDICTED ERRORS	1-D RUN-LENGTH CODING OF 16-STATE ORDERED PRED. ERR.	IBM DUAL MODE K = 4	IBM DUAL MODE K = ∞
WJM TYPED	2200 x 1700	0.103	0.128	0.078	0.069	0.083	0.068	0.087	0.063
GWU TYPED	2200 x 1700	0.097	0.118	0.080	0.073	0.084	0.071	0.085	0.066
BUNYAN WRITTEN	2200 x 1700	0.053	0.070	0.059	0.055	0.053	0.042	0.055	0.041
FORM	2200 x 1700	0.134	0.157	0.087	0.073	0.088	0.077	0.103	0.074
AVERAGE B/P		0.097	0.118	0.076	0.067	0.077	0.064	0.082	0.061
AVERAGE COMPRESSION FACTOR			8.5	13.2	14.9	13.0	15.6	12.2	16.4

Table 5. Baseline Performance: 300 Points/Inch, Vertical Scan.

IMAGE NAME	SIZE LINES X PIXEL	ALGORITHM PERFORMANCE IN BITS/PIXEL (B/P)							
				ENTROPY BELL LABS ALGORITHMS					
		1-D RUN-LENGTH ENTROPY	CCITT RUN-LENGTH CODING	1-D RUN-LENGTH CODING OF ORDERING WITH REF. TO PREV. LINE	1-D RUN-LENGTH CODING OF 2-STATE ORDERED PRED. ERR.	1-D RUN-LENGTH CODING OF 16-STATE PREDICTED ERRORS	1-D RUN-LENGTH CODING OF 16-STATE ORDERED PRED. ERR.	IBM DUAL MODE K = 4	IBM DUAL MODE K = ∞
WJM TYPED	2550 x 3304	0.069	0.090	0.058	0.052	0.062	0.046	0.060	0.046
GWU TYPED	2550 x 3304	0.088	0.097	0.065	0.058	0.068	0.051	0.068	0.051
BUNYAN WRITTEN	2550 x 3304	0.052	0.072	0.045	0.039	0.041	0.026	0.042	0.027
FORM	2550 x 3304	0.147	0.198	0.075	0.068	0.073	0.068	0.104	0.068
AVERAGE B/P		0.089	0.114	0.061	0.054	0.061	0.048	0.068	0.048
AVERAGE COMPRESSION FACTOR			8.8	16.4	18.5	16.4	20.8	14.7	20.8

Table 6. Baseline Performance: 300 Points/Inch, Horizontal Scan.

IMAGE NAME	SIZE LINES X PIXEL	ALGORITHM PERFORMANCE IN BITS/PIXEL (B/P)							
				ENTROPY BELL LABS ALGORITHMS					
		1-D RUN-LENGTH ENTROPY	CCITT RUN-LENGTH CODING	1-D RUN-LENGTH CODING OF ORDERING WITH REF. TO PREV. LINE	1-D RUN-LENGTH CODING OF 2-STATE ORDERED PRED. ERR.	1-D RUN-LENGTH CODING OF 16-STATE PREDICTED ERRORS	1-D RUN-LENGTH CODING OF 16-STATE ORDERED PRED. ERR.	IBM DUAL MODE K = 4	IBM DUAL MODE K = ∞
WJM TYPED	3304 x 2550	0.078	0.103	0.057	0.051	0.061	0.046	0.064	0.045
GWU TYPED	3304 x 2550	0.077	0.097	0.059	0.053	0.062	0.050	0.063	0.046
BUNYAN WRITTEN	3304 x 2550	0.040	0.054	0.044	0.040	0.039	0.031	0.041	0.030
FORM	3304 x 2550	0.102	0.121	0.065	0.057	0.065	0.054	0.075	0.052
AVERAGE B/P		0.074	0.094	0.056	0.050	0.057	0.045	0.061	0.043
AVERAGE COMPRESSION FACTOR			10.6	17.8	20.0	17.5	22.2	16.4	23.3

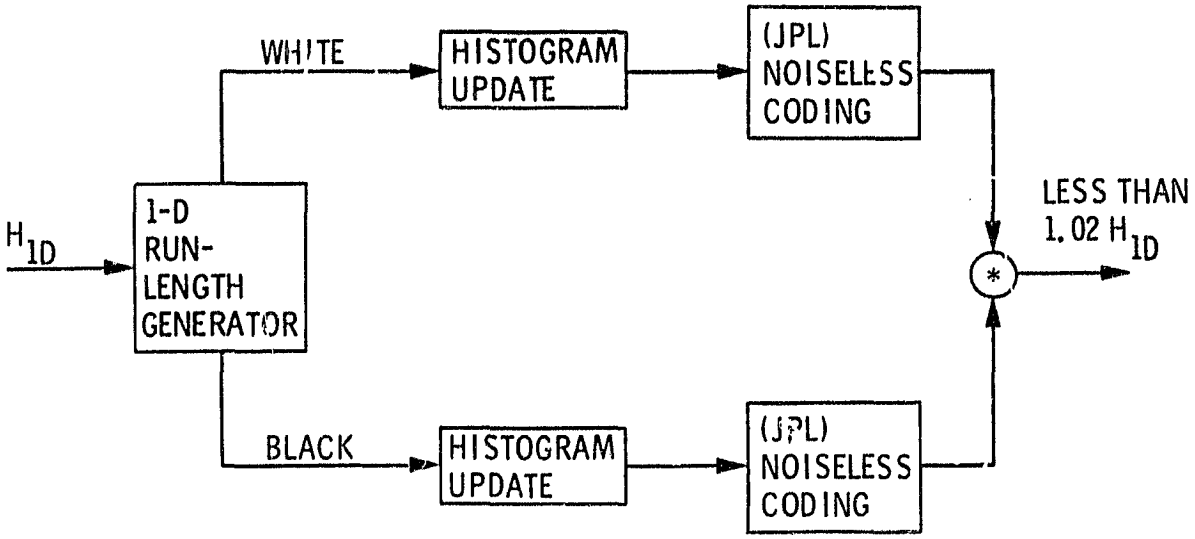


Fig. 6. Histogram Update Coding.

Markov/Histogram Update/JPL Coding

The Histogram Update/JPL Coding was applied to the output of a Bell Labs Markov predictor (Fig. 7), yielding performance within 10% of the two-dimensional prediction entropy.

Directed Markov Prediction

We also investigated an extension of the Bell Labs approach called Directed Markov Prediction (Fig. 8). In general this prediction technique reduced two-dimensional entropies by 10 to 15% and improved performance correspondingly. This approach is discussed further in Appendix B.

Segmentation Coding

Segmentation coding is illustrated in Fig. 9, which shows a small region of a document containing the word "where". Segmentation representation of a document consists of:

a) Identifying such regions of text or non-text,

b) Further partitioning character strings into individual regions containing single characters,

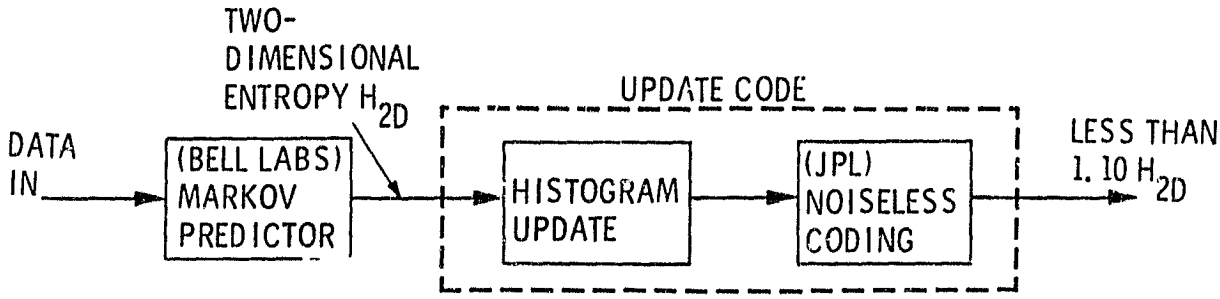


Fig. 7. Markov Predictor/Update Code.

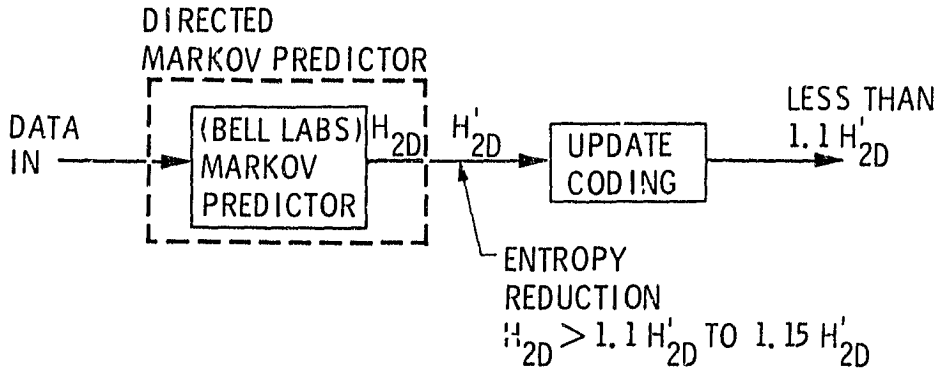


Fig. 8. Directed Markov/Update Code.

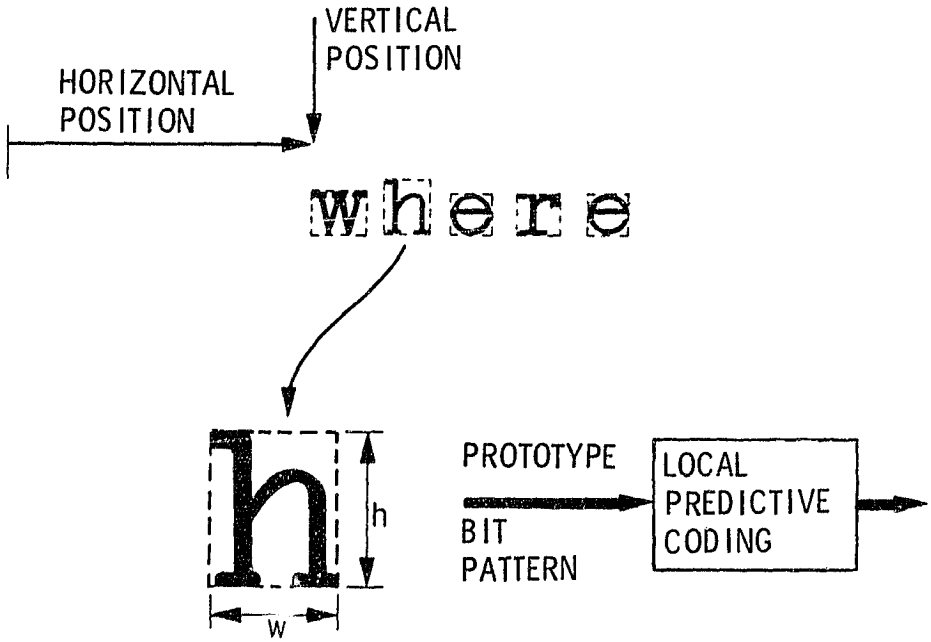


Fig. 9. Segmentation Coding.

c) Coding information defining the size and location of each region,

d) Coding the interior of each region (called a prototype bit pattern) using modified versions of the predictive coding algorithms just discussed.

- The major observation here is that the performance of this approach is **equivalent** to the best two-dimensional predictive techniques that don't segment. (12)

- A secondary observation to be used later is that the interior of regions containing 200 p/inch characters can be typically coded with an average of 135 bits/char. (13)

MAJOR OBSERVATIONS

If we supplement the latter investigations with additional runs using the CCITT standard test set, we can make the summary observations given below.

One-Dimensional Techniques

- Entropies
  - Vertical scan is 20% more than horizontal.
  - Entropy at 300 p/inch is 1.74 times the 200 p/inch entropy.
- CCITT standard at 200 p/inch
  - Coding is 4-25% above entropy.
  - Compression factors range from 5:1 to 16:1.
  - The average compression factor is 10:1.

Two-Dimensional Techniques

- All sophisticated techniques are quite similar in performance. This includes segmentation coding (see (12) and (13)).

- The performance difference between vertical and horizontal scan decreases to 5%.

- Compression factors range from 7:1 to 35:1.

- The average compression factor is 20:1. This is a 2:1 gain over the 1-D CCITT Standard.

- These techniques cannot provide a 2:1 improvement on dense text (only 33%).

III. A UNIVERSAL SYSTEM FOR EFFICIENT ELECTRONIC MAIL (USEEM)

The removal of a constraint that binary facsimile images must be reconstructed precisely leads to the possibility of the substantial performance advantages noted in (4) and (5) and the cost savings noted in Tables 1 and 2. This potential resides in a concept called a "Universal System for Efficient Electronic Mail (USEEM)" which achieves these notable results primarily from the use of unsupervised character recognition and the adaptive noiseless coding of text. It represents an extension of similar work pioneered by W. Pratt, P. Capitant, W. Chen, E. Hamilton and R. Wallis of Compression Labs, Inc.[10]. Some of their original algorithms were used intact whereas others were substantially modified or replaced to achieve improved characteristics.

This section first provides an introduction and summary of the USEEM concept. Further details on performance, segmentation, character recognition and the coding of text follow.

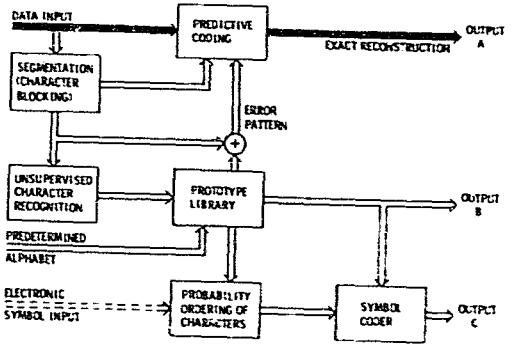
INTRODUCTION/SUMMARY

Functional Capability

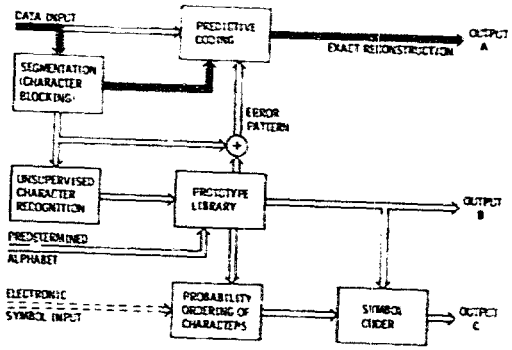
USEEM can be described as a modular, six-level system concept where each higher numbered level provides some further capability to communicate electronic mail. It is expected that each such incremental supplement to functional capability will correspond to an implementation increment as well. That is, a lower level may be implemented with the expectation that the advantages of a higher level may be added later without requiring complete redesign.

Figure 10 illustrates the modular form of USEEM. The primary data flow for the added capability at each step is indicated by heavy arrows.

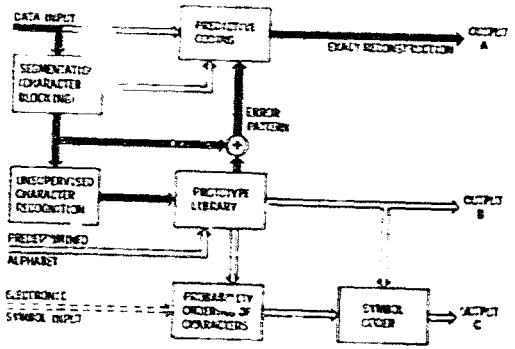
The **first level** of USEEM, labelled Predictive Coding, refers to all the standard one and two-dimensional techniques for **noiselessly** coding (bit-by-bit, exact reconstruction) scanned and digitized binary images. These include the familiar techniques of run-length coding, Markov state prediction, etc., as discussed in Section II. When averaged over all forms of potential electronic mail, the best of these techniques can be expected to reduce the bit rate requirements by about 20:1, compared to the uncoded binary output of a Postal Service scanner. However, this advantage drops to about 7:1, when dense text is considered.



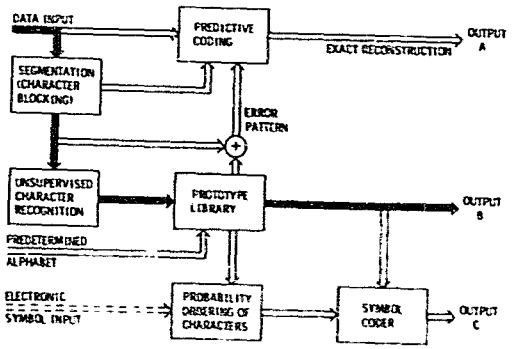
1. PREDICTIVE CODING



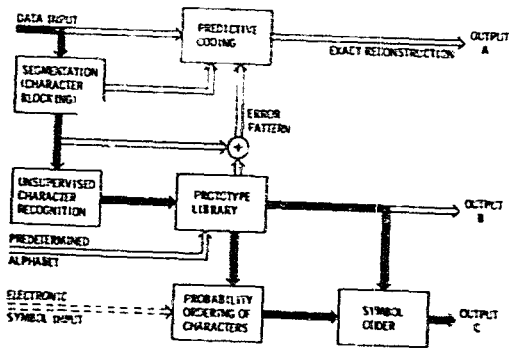
2. CHARACTER PREDICTIVE CODING



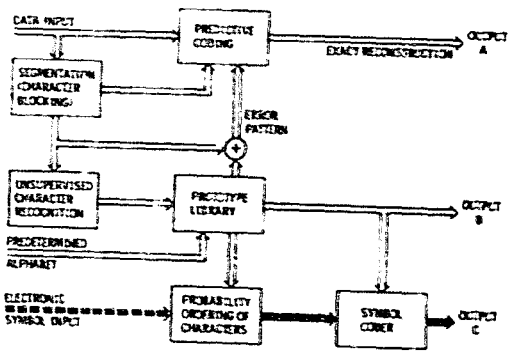
3. PREDICTION BY CHARACTER RECOGNITION



4. TRANSMISSION OF LIBRARY ID



5. CODING OF LIBRARY CHARACTERS



6. CODING OF DIRECT ELECTRONIC INPUT

Fig. 10. Six Levels of USEEM.

USEEM Level 2 is not expected to provide performance improvements in the ability to noiselessly code electronic mail. However, it will provide a functionally crucial capability to enable the significant performance advantages of higher USEEM levels.

USEEM Level 2 (Segmentation Coding, Section II) achieves essentially the same performance as Level 1 by first partitioning a document into all white regions and regions containing black and white transitions. Since the vast majority of electronic mail will contain principally typewritten text, the primary consequence of this "segmentation" procedure is to place rectangular blocks around characters. USEEM Level 2 then identifies the location of each segmented region (usually a character) and sends a coded message identifying its contents. The latter coding procedure may be derived from the collection of predictive techniques for a Level 1 USEEM.

USEEM Level 3 is really a consequence of the principal innovation of Level 4. It offers the possibility for some improvement in performing noiseless coding of electronic mail but as yet has not been investigated enough to allow an accurate estimate of this potential.

USEEM Level 4 clearly offers the potential for a significant improvement in the ability to communicate and store electronic mail. To achieve these gains, we must replace the restrictive fidelity criterion of "bit-by-bit exact reconstruction" by the looser, but demanding, criterion, namely, that the user be satisfied with the product he gets. This simply returns to the basic system requirements for electronic mail and, as will later be illustrated, can actually result in better fidelity.

USEEM Level 4 initiates a "prototype" library by storing the bit patterns of "new" incoming segmented characters. An incoming character is new if it does not **look like** any prototype already stored in the library. The bit pattern for a new character is communicated with the predictive coding techniques noted for USEEM Level 2. However, when an incoming character has been perceived to resemble a prototype in the library, only an identifier of which library character has been observed is needed. A decoder will output a replica of the corresponding prototype bit pattern from its own library, when it receives this identifier. The effect on performance can be quite significant since a prototype character bit pattern may require an average of 135 bits, whereas a library identifier needs only 8. JPL's initial results indicated that greater than 90% of a document's characters could be expected to be communicated using a library identifier. On the most

difficult form of document, one containing dense text, this yields overall compression factors of 45:1 to 70:1 versus only 7:1 by the best facsimile techniques (USEEM Levels 1 and 2).†

Character recognition. The criterion for judging an incoming character alike or unlike a prototype library character must almost totally prevent a character, say an "a," from being interpreted as another form, say a "c." A USEEM Level 4 would replace the "a" by a "c" in this case with unacceptable results. Replacing one "a" with another (of the same font) has little impact. The character recognition techniques developed so far for USEEM have sought to maximize the occurrence of replaceable characters while minimizing the occurrence of such errors. Although tested on only a limited data set, the results fully support these objectives and the Level 4 performance projections noted above. Testing was performed on several medium density documents containing 1500 segmented characters per page. By the first page, 95 percent of the characters were correctly identified as replaceable by a library prototype. Most of the remaining 5 percent represented first-time library entries because the library was initially empty. By the second page almost 99% of the characters were correctly identified as replaceable by library characters. The latter percentage corresponds to compression factors exceeding 200:1.

USEEM Level 5 seeks to capitalize on the redundancy existing in language by applying noiseless coding to characters and words resulting from Level 4 operations. This can directly reduce the bit cost of prototype library identifiers 2 to 3 times, as evidenced by initial simulations. In combination with an improvement in the repeat fraction for dense text to 0.95, the noiseless coding of text could improve overall compression factors to 100:1. For medium-density text, compression factors of over 200:1 on the first page of a document and over 400:1 on subsequent pages could be expected. All figures represent roughly an order of magnitude improvement over the best two-dimensional (noiseless) facsimile techniques (USEEM Levels 1 and 2).

By appropriate segmentation of characters the strings of characters exiting a USEEM Level 4 processor will look much like strings of characters exiting typical automated office equipment. Consequently, the efficient character coding algorithms developed for Level 5 should apply to most forms of electronic mail entered directly from terminals. **USEEM Level 6** is merely a statement of this observation. A 2-to-3:1 reduction in the bits needed to communicate this nonscanned form of input should be possible

---

†Note that a USEEM Level 4 can still functionally execute the noiseless coding requirements of Level 1 and 2.

Quality

USEEM levels 1-3 reduce the number of bits required to represent incoming data while yielding exact bit-by-bit reconstruction of that data. An exact replica of the binary image produced by a Postal Service scanning device will be reconstructed as a final product. The constraint which requires exact reconstruction is a benevolent attempt to ensure that final product; unfortunately it may result in mediocre reconstructed quality in practice. This point is illustrated in the photograph in Fig. 11 showing two sequences of the lower case letter "a."

A close inspection will reveal that the sequence of ten a's at the top is clearly of better quality than those at the bottom. Further, the better set at the top can be communicated at one-fifth the data rate as the inferior set at the bottom can be. A similar statement could be made for storage requirements. The "exact reconstruction" limitation would eliminate this preferable option of better quality along with significant reductions in rate requirements since it would preclude USEEM Levels 4-6.

Actually, the inferior quality set represents ten lower case a's taken from a document scanned at 200 points/inch and are reproduced "exactly," bit-by-bit. The improved set of a's simulates the application of USEEM Level 5 to documents scanned at 300 points/inch. The last nine a's are replicas of the first, simulating the use of a prototype character library as discussed earlier. The reduction in data rate requirements by a factor of five takes into account both the increased number of pixels at the higher resolution and the significant improvement in compression factors from USEEM Level 5. To achieve 300 points/inch quality under an "exact reconstruction" criterion would, by comparison, require increasing the data rate and buffering requirements by almost a factor of two.

Added advantages. An indirect consequence of the character segmentation process (USEEM Level 2) is actually an improvement in text readability at a given scan resolution. The segmentation process can be used to straighten slightly crooked lines and to separate "run-together" characters.

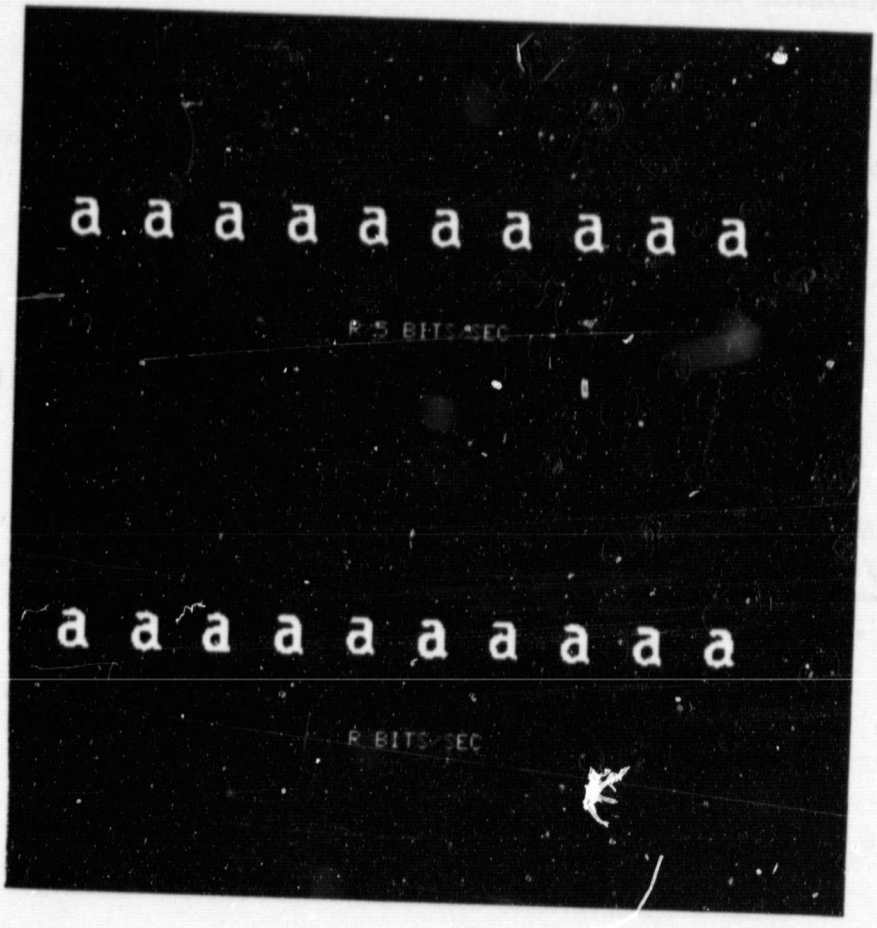
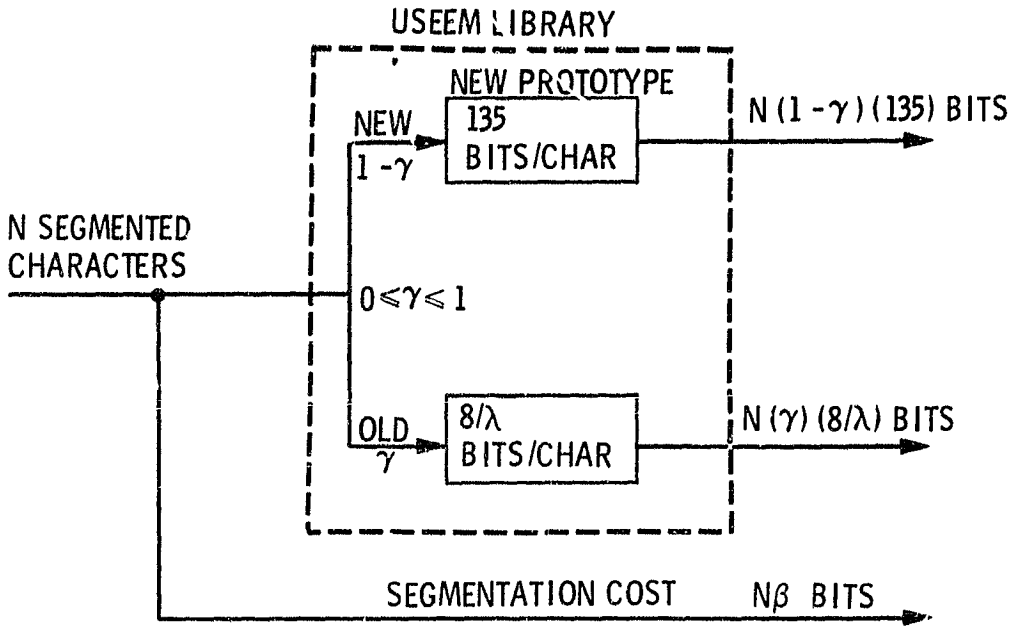


Fig. 11. Better Quality at Lower Rate.

USEEM PERFORMANCE ANALYSIS

Origin of the Bit Costs

Consider the three components of data emanating from a USEEM library shown in Fig. 12.



λ = TEXT COMPRESSION FACTOR

β = SEGMENTATION COST IN BITS/CHAR

γ = FRACTION OF INCOMING CHARACTERS REPLACEABLE BY LIBRARY CHARACTERS

Fig. 12. Bit Costs from Library.

Identical library identification systems reside at the sending and receiving sites. Each system contains a library of prototype characters which (for now) are assumed empty at the start of a document. Characters entering the identification system are tested against the library prototypes to determine whether they are "OLD" or "NEW". An "OLD" character looks enough like a library character to be represented by it. In this case only a library identifier need be communicated to notify the receiving end that it can reconstruct the specific library prototype. A 256 character library needs only 8 bits/

character, if a fixed length code is used. Use of noiseless text compression by a factor λ would reduce this to

$$8/\lambda \text{ bits/character} \tag{14}$$

A "NEW" character does not look like one of the existing prototypes in the library and must be communicated as a bit pattern. Using modified predictive facsimile compression algorithms from Section II, 200 points/inch characters can typically be represented with

$$135 \text{ bits/character} \tag{15}$$

including cost for identifying the size of the character.

A "NEW" character is also added to the prototype library (replacing the oldest entry if necessary) at both the sending and receiving sites.

The parameter

$$\gamma, 0 \leq \gamma \leq 1 \tag{16}$$

is used to specify the fraction of incoming characters which can be communicated as library identifiers (OLD). Achieving a high value of γ will soon materialize as the key to high performance.

An additional "segmentation cost" of

$$\beta \text{ bits/character} \tag{17}$$

is associated with identifying the location of each character bit pattern.

Then from Fig. 12, N segmented characters will require

$$N\left[\beta + (135)(1-\gamma) + \frac{8}{\lambda}\gamma\right] \text{ bits} \tag{18}$$

to code.

Text density. 4000 characters on a standard 8-½ x 11" page is extremely dense text, particularly at a scan density of 200 points/inch. 1500-2000 characters on a page can be considered medium-density text and 800 characters/page as light text. Such densities can more generally be expressed in characters/pixel by

$$D_T = \frac{\text{Number of characters}}{\text{(no pixels in area containing characters)}} \tag{19}$$

For example, D_T for dense text is given by $4000/3.7 \times 10^6 \approx 0.001$.

Text performance equation. Substituting (19) in (18) we obtain the performance equation for text regions given by

$$\frac{1}{\omega_T} = R_T = D_T\,[\beta + (135)(1-\gamma) + 8\gamma/\lambda] \text{ bits/pixel} \tag{20}$$

where we recognize ω_T as the text compression factor. ω will similarly be used in subsequent equations.

USEEM performance equation. Non-text regions are coded by the two-dimensional techniques of Section II. We denote the corresponding rate for non-text regions by

$$\frac{1}{\omega_{NT}} = R_{NT} \text{ bits/pixel.} \tag{21}$$

Then letting

$$\alpha \tag{22}$$

denote the fraction of a document which is non-text we get the USEEM performance equation†

$$\frac{1}{\omega_U} = R_U = \alpha R_{NT} + (1-\alpha)R_T \tag{23}$$

where R_T is given by (20).

---

†Observe, we have left out the situation where some regions of text are incorrectly treated as non-text. However, such text would then be coded by 2-D facsimile with roughly the same results as being treated as text and being coded as "NEW" prototype patterns (see Section II).

Graphical Analysis

We will now use these equations to graphically investigate the impact of parameters α, D_T, β, γ, and λ on USEEM compression factors.

- α: Changing α varies the ratio of text to non-text in a document. When $\alpha = 0$, a document consists of text only.†

- ω_{NT}: When considering USEEM non-text regions we will assume the two-cases where facsimile compression factors are $\omega_{NT} = 7{:}1$ (dense) and $\omega_{NT} = 20{:}1$ (medium).

- D_T: Text density values will correspond to dense, medium and light text, as defined above.

- β: We will consider the two values 10 and 0 for the cost in segmenting characters. The former (β = **10 bits**) is a requirement of a Compression Labs, Inc.[10] process called here "**character segmentation**", and the latter (β = **0 bits**) is the result of a procedure called "**text segmentation**". Both of these will be briefly discussed later.

- γ: This parameter, perhaps the most important of all, specifies the fraction of text characters which can be represented by, and hence communicated as, an existing library character. γ will appear in all graphs as the abscissa, varying between 0 and 1.

- λ: A USEEM prototype library containing 256 elements requires $8/\lambda$ bits/character to communicate an identifier where λ is the reduction factor obtained by noiseless coding of text. We will use the values of 1 (no text compression) and 2, the latter supported for dense text by preliminary simulations.

---

† Observe, we have left out the situation where some regions of text are incorrectly treated as non-text. However, such text would then be coded by 2-D facsimile with roughly the same results as being treated as text and being coded as "NEW" prototype patterns (see Section II).

Dense text only. Figure 13 displays plots of text compression factor ω_T versus γ under the conditions of dense text only (corresponding to the CCITT standard test image (No. 4 in Fig. 2)).

Included for comparison is a horizontal line $\omega_T = 7$ labelled "facsimile". This represents the expected performance of 2-D predictive facsimile techniques.

The dashed curve represents a basic USEEM configuration using the simpler "character segmentation" approach ($\beta = 10$) and no text coding ($\lambda = 1$). Even in this case a $\gamma = 0.9$ yields a compression factor of 30:1 compared to only 7:1 by facsimile. At the bottom and where $\beta = 0$, the segmentation overhead reduces the compression factor to slightly less than facsimile. However, γ need only be 0.08 to yield equal performance.

The next curve shows the advantage of a text segmentation approach ($\beta = 0$). At the low end ($\gamma = 0$) text segmentation raises performance slightly (to an equivalence with facsimile). The advantage of text segmentation increases rapidly at higher values of γ where at $\gamma = 1$ the (potential) advantage is over 2:1. In absolute terms USEEM text segmentation provides a 45:1 compression factor at $\gamma = 0.9$ and 110:1 at $\gamma = 1.0$.

The impact of adding noiseless text coding (with $\lambda = 2$) is shown by the next curve. Improvements to USEEM compression factors are negligible until $\gamma = 0.7$, achieving a maximum gain equal to $\lambda = 2$ when $\gamma = 1.0$. In absolute terms, the combination of text segmentation **and** text coding yields overall compression factors of 55:1 when $\gamma = 0.9$ and 220:1 when $\gamma = 1.0$. The latter figure represents performance 30 times better than facsimile.

Effects of text density. Figure 14 shows the effect of text density on the potential compression factor of a USEEM system equipped with text segmentation ($\beta = 0$) and text coding ($\lambda = 2$). The lower curve for dense text has been transferred from Fig. 13.

Observe that in all cases, USEEM performance equals facsimile performance for $\gamma \rightarrow 0$. At the other end, limiting performance for $\gamma \rightarrow 1$ reaches almost 600:1 for medium density text and over 1000:1 for light text.

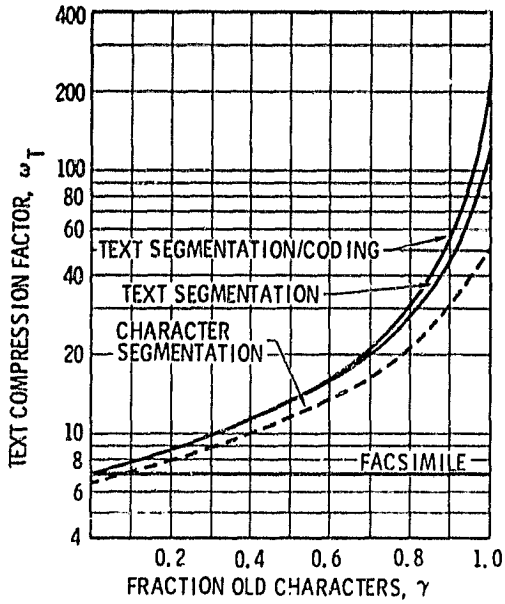


Fig. 13. USEEM Compression Factor Estimates: Dense Text.

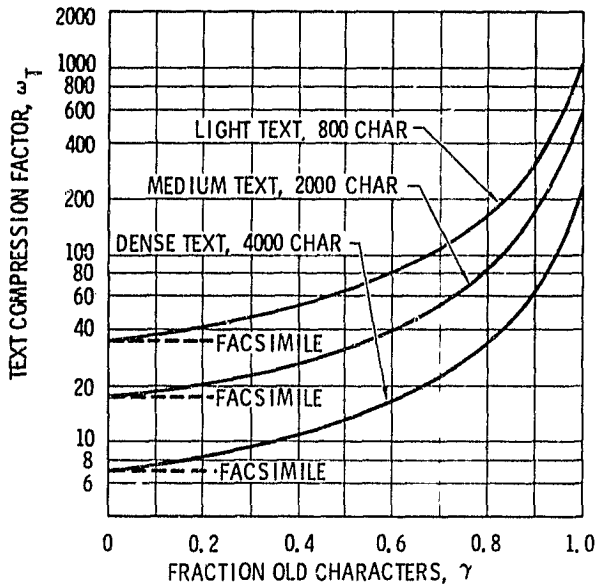


Fig. 14. Effect of Text Density.

Variations in α. The impact of varying percentages of text and non-text regions within a document is shown for dense text in Fig. 15, medium text in Fig. 16, and light text in Fig. 17.

In each figure the document percentage of text has been varied by letting†

$$\alpha = 0.0, 0.2, 0.5 \text{ and } 1.0 \tag{24}$$

for

$$\omega_{NT} = 7 \text{ and } 20 \tag{25}$$

Compression factors for text regions assume that both text segmentation ($\beta = 0$) and text coding ($\lambda = 2$) have been incorporated.

We have just investigated the case where $\alpha = 0.0$ such that a document is all text. Each of these corresponding curves for dense, medium, and light text have been transferred to Figs. 15, 16, and 17, respectively. They appear as the uppermost curve (for high values of γ) in each case.

By Eq. 23 when α equals 1 a document is considered all non-text and compression factors equal the factor ω_{NT} regardless of γ. This case then appears as horizontal lines at $\omega_{NT} = 7$ and $\omega_{NT} = 20$.

Observe that the effect of **high** percentages in non-text can significantly alter the overall compression factor. The worst case situation occurs when the non-text regions can only be compressed by $\omega_{NT} = 7$ and the text itself is light (Fig. 17). For example a potential compression factor of 1000:1 ($\gamma = 1$) is reduced by almost two orders of magnitude to 13:1 when half the document is non-text (40:1 if $\omega_{NT} = 20$).

---

†The non-text contributions to these graphs can also be viewed as text regions which have been incorrectly segmented.

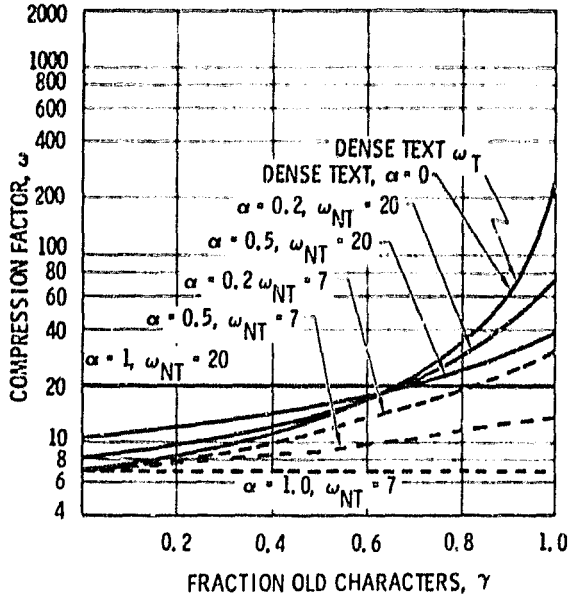


Fig. 15. Mixing Dense Text and Non-Text.

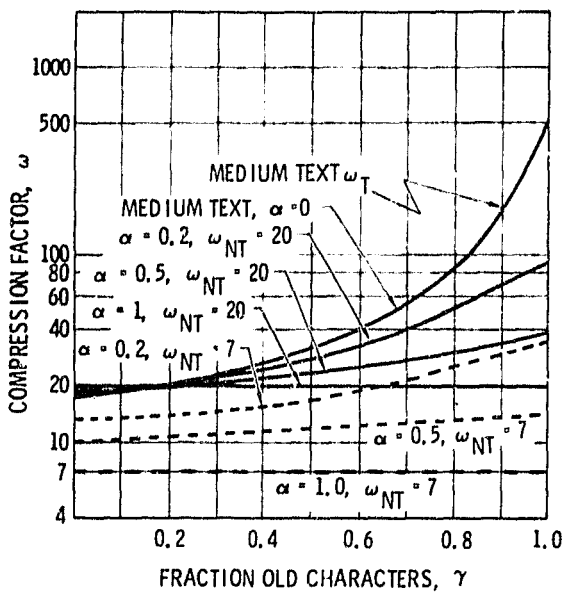


Fig. 16. Mixing Medium Text and Non-Text.

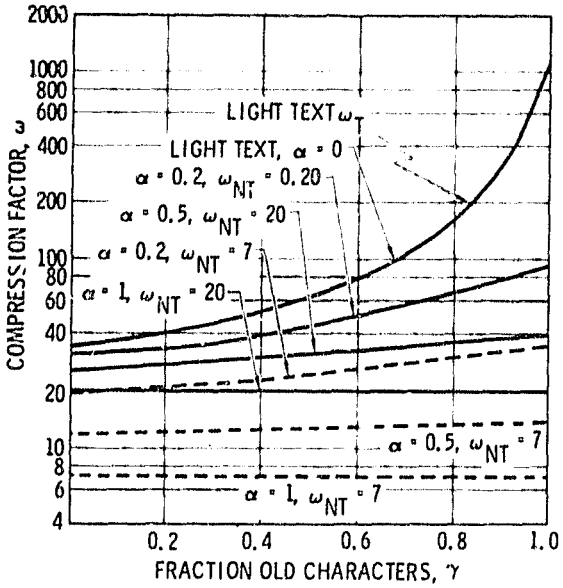


Fig. 17. Mixing Light Text and Non-Text.

Simulation Results

Table 7 shows the results of actual USEEM performance runs on binary images containing only text. These include the extremely dense CCITT French document No. 4 in Fig. 2 and four medium-density letters generated by a word processor and then scanned at 200 points/inch.

The repeat fraction γ in Table 7 is the same parameter as in Figs. 13-17 except that these are actual results. First page fractions include the fact that the USEEM library starts out empty. Second page results assume an established library.

The "errors" indicated can be viewed as minor errors which would not be misinterpreted by a reader. Most of the CCITT errors were due to the special symbols of the French language which were not accounted for in the initial pattern recognition development.

Table 7. Actual USEEM Performance.

	IMAGE					
	CCITT NO. 4	WORD PROCESSOR GENERATED				
		1	2	3	4	
CHARACTERS	4000 (DENSE)	1500 (MEDIUM)	1500	1500	1500	
REPEAT FRACTION γ	0.927	0.945	0.954	0.954	0.933	1st PAGE
	0.943	0.980	0.988	0.988	0.972	2nd PAGE
ACTUAL COMPRESSION FACTOR ω (NO TEXT CODING)	53	166	182	182	150	1st PAGE
	70	232	257	257	212	2nd PAGE
POTENTIAL ω WITH TEXT CODING	68	222	252	252	195	1st PAGE
	90	370	439	439	319	2nd PAGE

The dense document started at $\gamma \approx 0.93$ on the first page and would increase to $\gamma \approx 0.94$ if a second page was presented. Compression factors (actual) were 53 and 70 for first and second page respectively. **Estimates** of additional gains possible from text coding for first and second page respectively ($\lambda = 2$) are shown.

First page γ values near 0.95 were produced on the four medium-density documents. By the second page, values close to $\gamma = 0.99$ were achieved. These yielded **actual** compression factors of from 150:1 to 250:1.

Adding text coding in these cases offers the **potential** for a range from 200:1 to 400:1 ($\lambda = 2$).

SEGMENTATION

The basic approach to the segmentation of textual material is to isolate individual characters and independently identify their locations. We call this approach **character segmentation**. If the context of a typed page is accounted for so that individual characters belong to "text lines," then significant reductions in identification overhead (β in the previous section) can be achieved. The output of this **text segmentation** process can, in fact, be made compatible with word processor formats, bringing scanned electronic mail much closer to direct entry electronic mail systems.

Character Segmentation

The basic character segmentation approach of Compression Labs, Inc.[10] is illustrated in Fig. 18 and uses the sequence of text "then. It ..." to illustrate that the prototype storage and processing of characters is in order of a) the scan line that first touches a character top (i.e., the tallest on a level line) and b) left to right.

The horizontal position of each individual character must be communicated. Position is determined by the highest point on a character called a Keypoint, as shown in Fig. 19.† Unfortunately, the communication of character Keypoints in a somewhat **random** fashion contributes significantly to an unnecessarily high character segmentation cost of $\beta = 10$ bits/character. That adjacent characters have adjacent Keypoints is essentially discarded by this approach.

---

†Height and width information of a character is assumed to be part of a prototype definition.

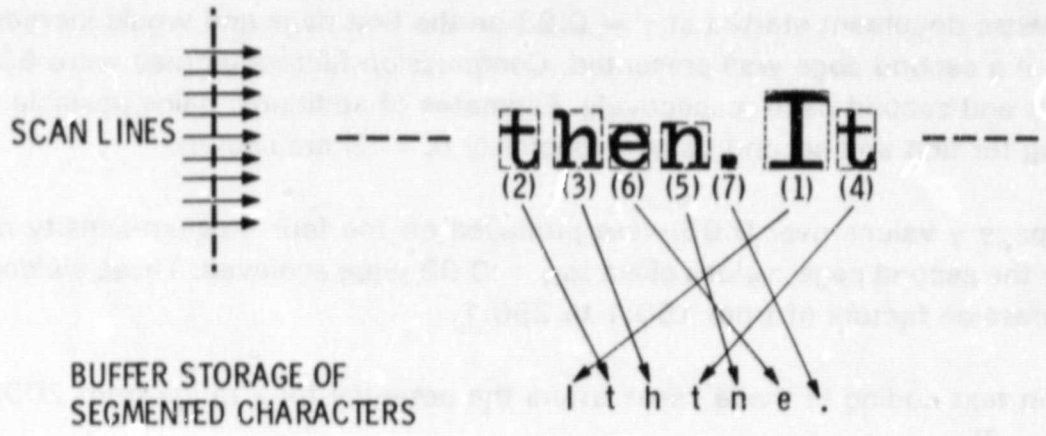


Fig. 18. Character Segmentation Ordering.

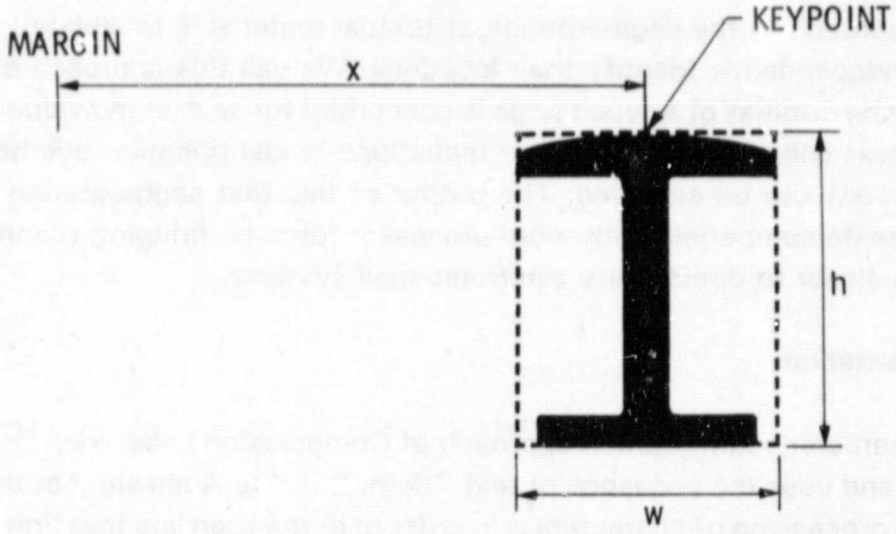


Fig. 19. Character Segmentation Keypoint.

Text Segmentation

Handling spaces. A normal sequence of characters making up words consists of the regular alphabetical characters separated by small spaces of one or more pixels. A sequence of words consists of the words separated by larger spaces originally generated as "blank characters".

If one simply assumes that any character is followed by a character space then such spaces can be inserted as part of the reconstruction process without increasing communication cost. The longer spaces which originated as blank characters can be treated by defining a "blank" prototype in the USEEM library.

Figure 20 shows a sequence of regular characters, character spaces, and a single blank character. Since **any** character is followed by a character space, the blank character has a character space on either side. We define the length of a **character space** to be:

$$\bar{s} = \text{Average of the minimum observed space (in pixels) between regular characters.} \tag{26}$$

Now assume that the minimum observed blank area, L, greater than $3\bar{s}$ must be a "blank character" **and** two character spaces. Then the length of a "prototype" blank character is

$$\bar{B} = L - 2\bar{s} \tag{27}$$

Both $\bar{B}$ and $\bar{s}$ can be determined quickly at the start of a document, if the font is unknown.

The number of blank prototypes in an observed blank area of length L' is given as†

$$n_B = \left\lceil \frac{L' - \bar{s}}{\bar{B} + \bar{s}} \right\rceil \tag{28}$$

provided $L' \geq 3\bar{s}$. This defines the prototype "blank" character.

The initial reconstruction process for a string of text would insert $\bar{B} + \bar{s}$, pixels of blank area for each prototype blank character transmitted. Character spaces of basic widths would follow each regular character. Any adjustment needed to fit the text precisely within a prescribed area (e.g., a line) could be accommodated by lengthening or shortening the actual reconstructed character spaces as needed (spread out across the area).

There is nothing magic about $\bar{s}$ except that its use will tend to place reconstructed text over the same area. In fact, reconstruction could include proportional spacing, even if it didn't exist in the original.

---

† $\lceil x \rceil$ is the smallest integer greater than or equal to x.

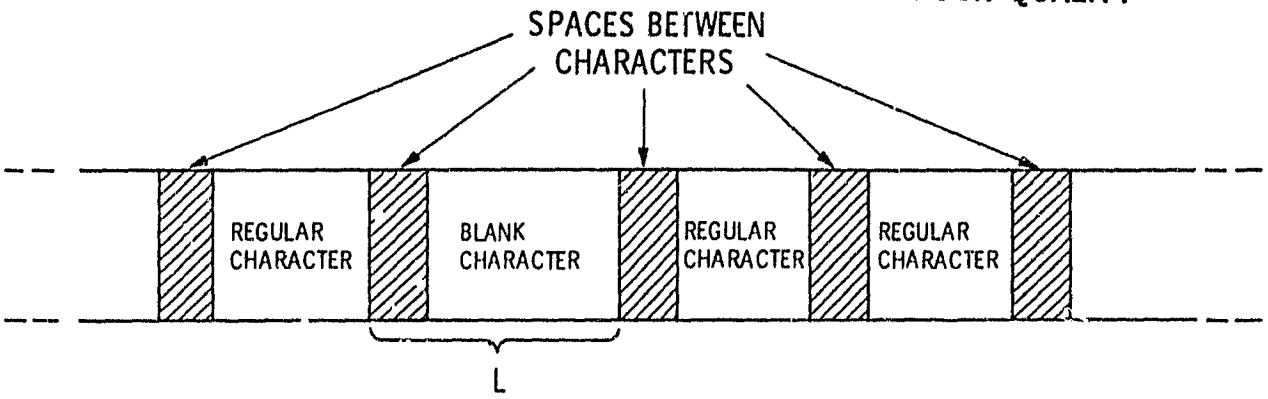


Fig. 20. Spaces in Text.

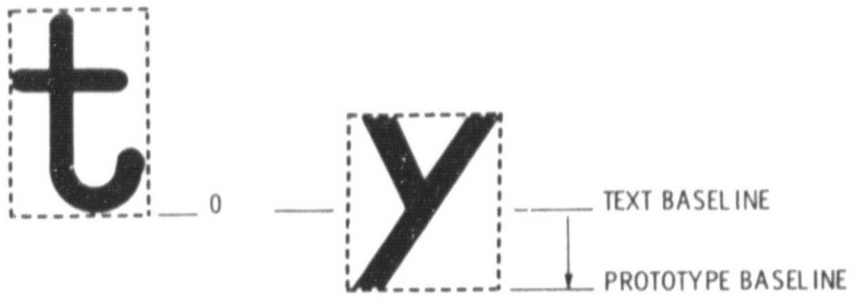
Text baseline. Text is intended to run along horizontal lines; therefore, it can be assumed that some form of malfunction has occurred when this is not the case. In addition, the base of most characters sits on the same horizontal line (i.e., a, b, c but not y, j, g). We define the latter horizontal line as the **text baseline** relative to the alphabet in use.

When a character prototype is first entered into the USEEM library, it is stored with information relating the position of the prototype relative to the currently assumed "text baseline" (see Fig. 21).

A prototype library is initialized only when a text baseline can be unequivocally determined as the scan line which has the most prototype baselines. Henceforth, text baselines in a local area (even tilted lines) can be determined from repeated prototypes. Conversely, the position of a prototype relative to a baseline can be used as a screening feature.

Example. Figures 22 and 23 illustrate the advantage of text segmentation. Fig. 22 is a copy of a letter in original form whereas Fig. 23 is the same letter after text segmentation. A close look will reveal that the original has numerous letters which have been run together (e.g., the m's in communication). This problem has been cleared up in Fig. 23 because the characters have been uniformly spaced as described in (26)–(28). Additionally, as already noted, the process of text segmentation would allow the straightening of crooked lines.

TEXT BASELINE IS THE SCAN LINE THAT THE MAJORITY OF CHARACTERS
SIT ON, E.G., a, b, c, NOT y, j, g



THE POSITION OF THE BOTTOM OF A CHARACTER PROTOTYPE RELATIVE
TO TEXT BASELINE IS PART OF PROTOTYPE INFORMATION

Fig. 21. Text Baseline.

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,

P.J. CROSS
Group Leader - Facsimile Research

Fig. 22. Original Cross Letter.

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,

P.J. CROSS
Group Leader - Facsimile Research

Fig. 23. Text Segmented Cross Letter.

UNSUPERVISED CHARACTER RECOGNITION

Basic Procedure

Figure 24 illustrates the basics of unsupervised character recognition. Input segmented bit patterns $\tilde{P}_{in}$ are compared to "prototype bit patterns" $\tilde{P}_i$, already stored in a library, by computing a "distance" between them. This distance measure $D_2(\cdot,\cdot)$ as used here is a modified Exclusive-OR procedure developed by Compression Labs Inc. called Template Matching[10]. The smallest D_2 value

$$D = \min_i D_2 (\tilde{P}_{in}, \tilde{P}_i) \tag{29}$$

is computed and if it is small enough

$$D < T_2 \tag{30}$$

a match with the corresponding $\tilde{P}_i$ is determined.

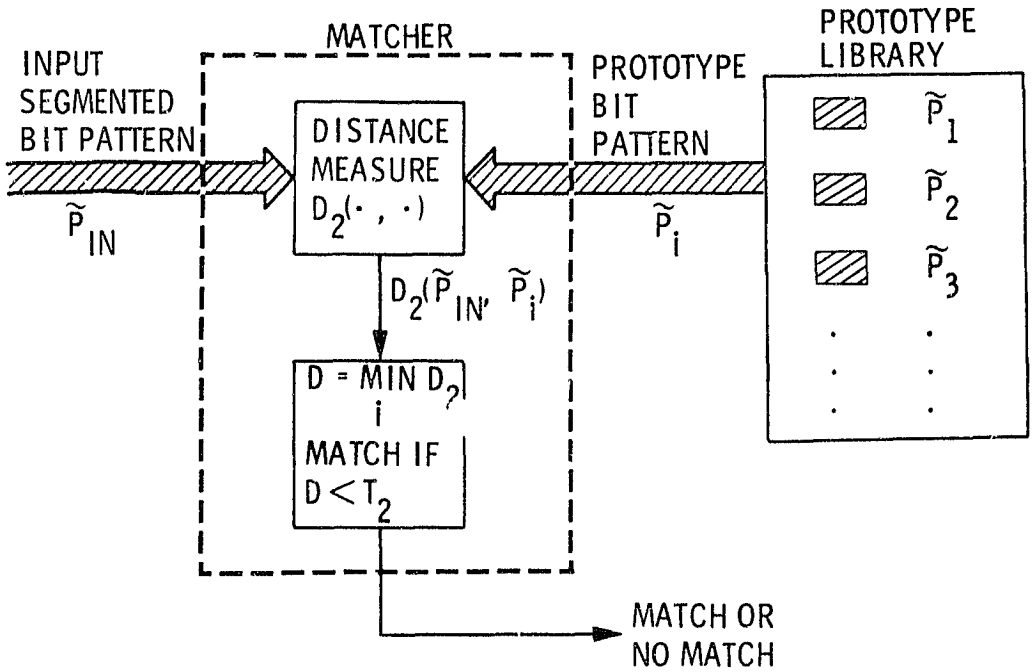


Fig. 24. Basic Character Recognition.

If a match is determined, only a library identifier for $\tilde{P}_i$ need be communicated. If there is no match, then P_{in} becomes a new library prototype and must be communicated separately as a bit pattern (to be added to an equivalent library at the alternate site).

Threshold T_2. The basic reason for threshold T_2 is illustrated in Fig. 25 which displays several **conditional** probability distributions of the distance measure D_2. Each curve represents the distribution of D_2 comparing the patterns of a specific **input character** "C" and the library patterns corresponding to specific characters. It should come as no surprise that the closest library pattern to an input "C" is another "C" as shown. However, an "O" is not too far off so that there is some overlap in the distributions. Deciding a match has occurred based only on choosing the minimum D_2 could then lead to errors since, as illustrated, there is some chance that a **library** "O" may be closer to an **input** "C" than a **library** "C". In fact, a library "D" may be still closer. The solution is to provide a threshold $T_2 = \lambda$ which avoids making such errors.

Thresholding in this manner, while preventing errors, can drastically reduce the chances of a match. For example, setting a threshold $T_2 = \lambda$ in Fig. 25 means that the chances of matching an input "C" with a library 'C' are represented by the area in the crosshatch region. We will return to this problem in subsequent paragraphs.

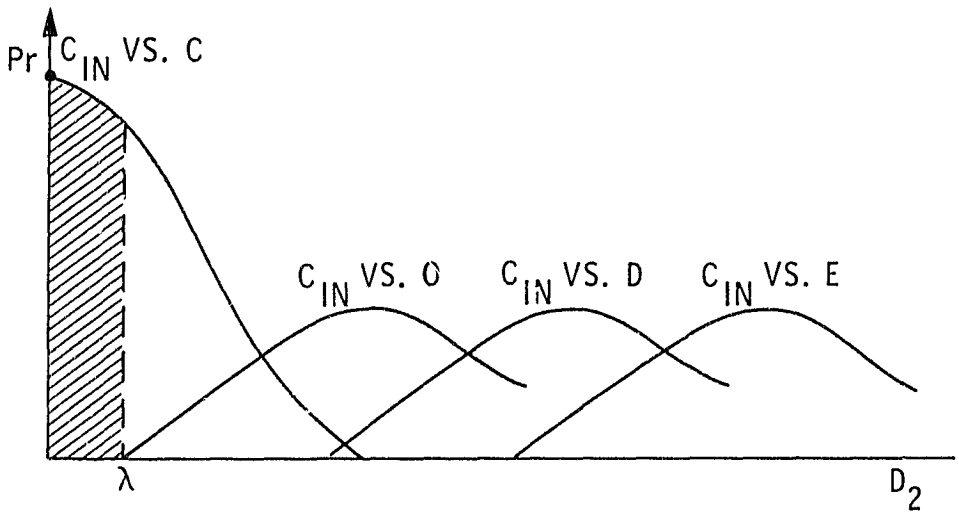


Fig. 25. Basic Thresholding.

Basic Prototype Screener

Performing the matching operations in Fig. 24 to find the minimum "D" would require that each member of the prototype library be compared. This would incur an exorbitant and unnecessary computational load. To avoid matching against each pattern Compression Labs Inc. implemented the basic "Prototype Screener" shown in Fig. 26.

The function of such a screener, a standard technique in pattern recognition, is to avoid the matching operations on those prototypes which aren't even close. This is accomplished by extracting and maintaining a feature vector $\tilde{F}_j$ with each prototype. The corresponding feature vector, $\tilde{F}_{in}$, of incoming patterns is compared to each $\tilde{F}_j$ using distance measure $D_1(\bullet,\bullet)$. If

$$D_1(\tilde{F}_{in}, \tilde{F}_j) < T_1 \tag{31}$$

the corresponding prototype bit pattern, $\tilde{P}_j$, is passed on to be compared in the matcher as described above.

While reducing matcher computation requirements, this procedure does nothing to improve performance since it only excludes prototypes which would never have been chosen by the matcher. Matching performance is the same as if all bit patterns had been passed through.

Two-Stage Character Recognition

A return to Fig. 25 reveals a possible way to improve performance. As shown, the area under the C_{IN} vs C curve above λ represents the probability that a "C" will have to be communicated as a prototype pattern.

The threshold could be raised if C_{IN} did not have to be matched with a prototype "O" or "D". This situation is illustrated in Fig. 27 where the threshold T_2 can be moved up to λ', allowing all C_{IN} to be correctly classified as the library prototype C.

A structure which achieves the desired situation is shown in Fig. 28. The prototype library is treated as made up of N classes. An incoming pattern is first **pre-classified** as belonging to one of these N classes. Subsequent steps of comparing feature vectors (within the selected class) to determine which patterns should be compared in the matcher is essentially the same as before. However:

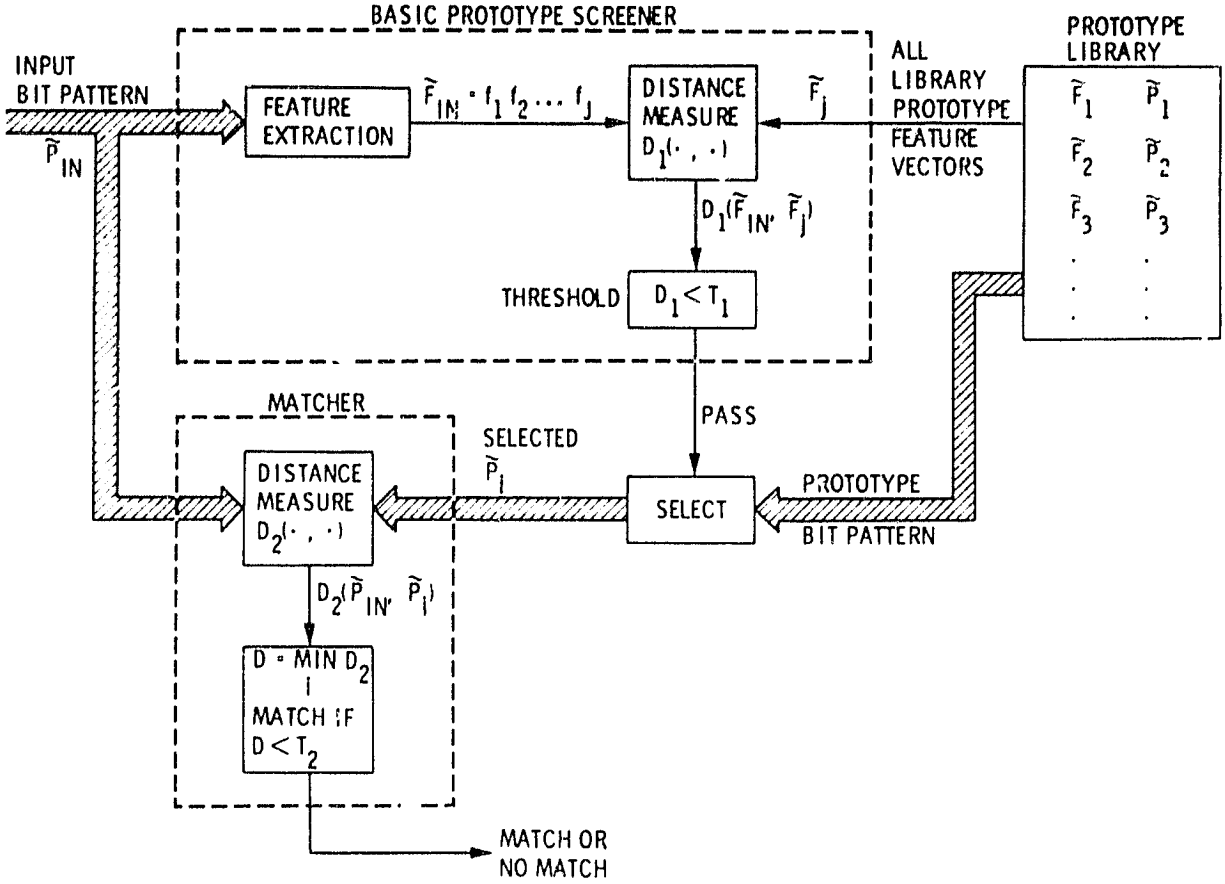


Fig. 26. Character Recognition with Prototype Screener.

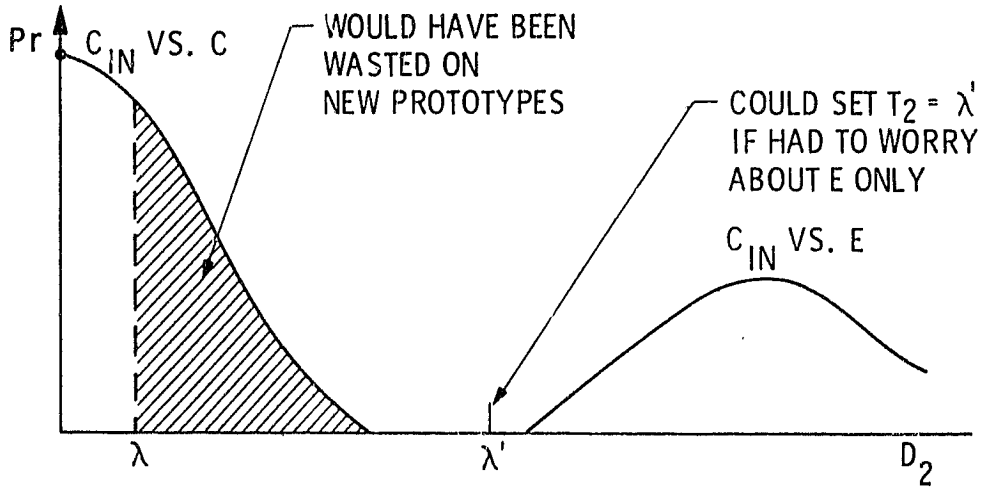


Fig. 27. Raising the Threshold, D_2.

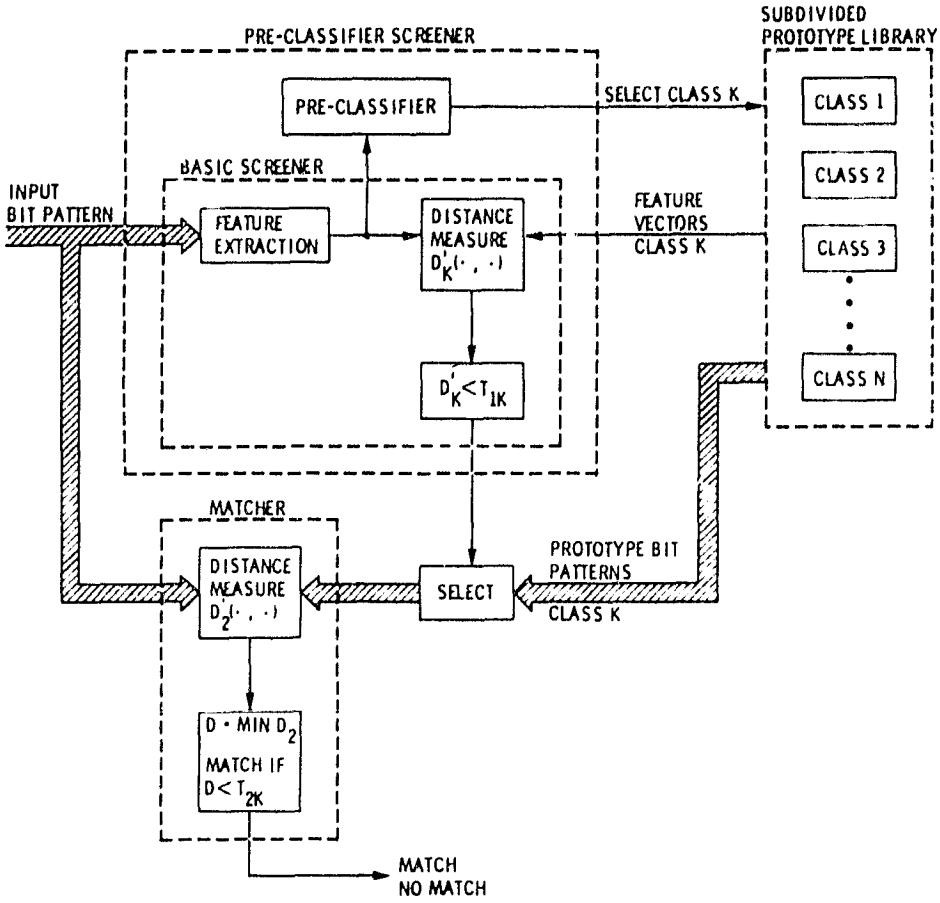


Fig. 28. Two-Stage Character Recognition.

1) A different screener distance measure $D_{1K}(\cdot,\cdot)$, and threshold, T_{1K}, can be used for each class, and similarly;

2) Matcher threshold T_2 can be adjusted for each class.

Pre-classification tree. The pre-classification process is accomplished by identifying a new set of features which distinguish between characters that appear close from a **matcher** point of view, for example, the "C" and "O" in Fig. 25.

The desired set of features can be placed into a decision tree as shown in Fig. 29.

As shown, a character is identified as containing one or two parts. If there is only one part, does it contain enclosures? If not, is the character large or small? If it does

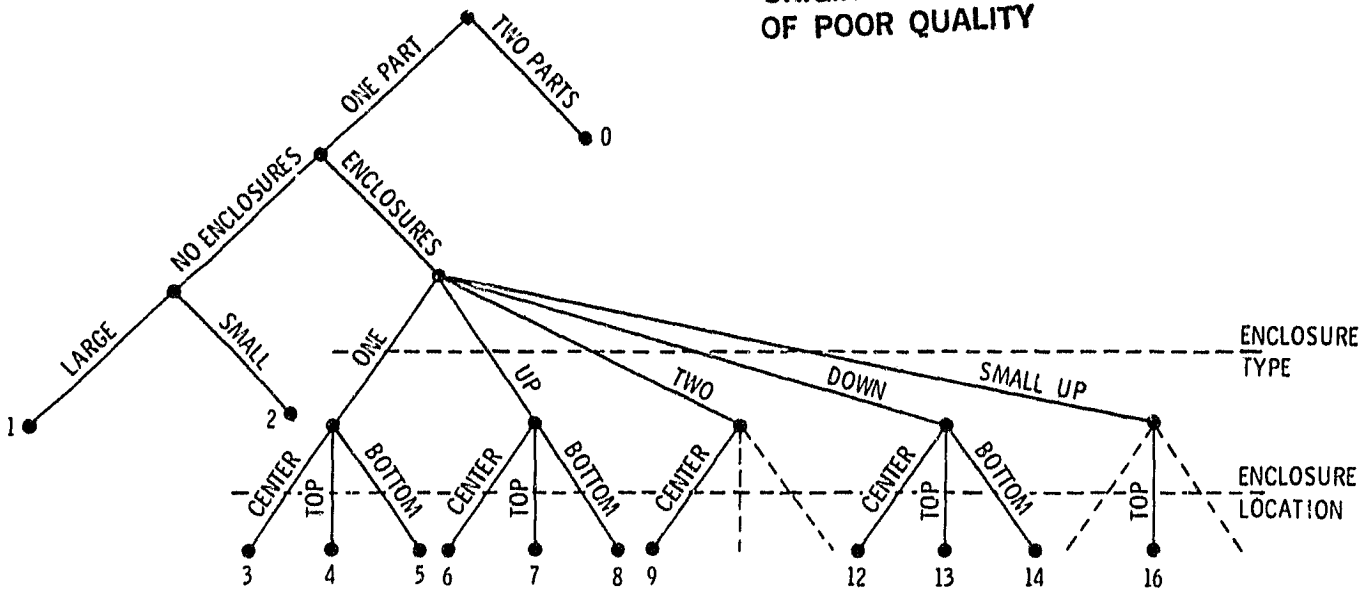


Fig. 29. Pre-classification Decision Tree.

have enclosures, how many are there and where are they located relative to a "text baseline" (see previous section)?

The result of an actual USEEM pre-classification is shown in Table 8.

Observations. The implementation of the pre-classification tree within the framework of the structure in Fig. 28 had a **dramatic** impact on performance. Many characters which had been quite difficult to distinguish before by matching no longer posed a problem since they were separated by the earlier steps.

An unintentional but important bonus was an equally dramatic **reduction in overall computation.** The matcher would now have to perform an average of only two template matchings per incoming character.

NOISELESS CODING OF TEXT

Preliminary work was performed to assess the potential reduction in bits needed to represent the string of identifiers emanating from a USEEM prototype library. By the assumption of text segmentation, the output of such a library is essentially in word processor form, without the benefit of information specifying which identifier corresponds

Table 8. Actual Pre-classification.

Class	Characters Identified In Class
0	i j
1	f 5 / 1 7 t
2	. ,
3	O o 0 D A
4	R e P p 9
5	a d b
6	u y N v w Y M
7	k
8	J C E S L G
9	8 g B
10	3 2 c T F
11	n h
12	m
13	r

Observe that "C" has been separated from "O" and "D".

to which alphabetical character. An assessment was made by defining a fairly complex coding structure and then using the output of an actual word processor to estimate expected code performance. These results showed that a reduction of 2 to 3 times could be obtained. The factor $\lambda = 2$ was used in earlier estimates projecting overall USEEM performance if text coding was fully implemented.

Since these simulations were preliminary, we will only briefly discuss the text coding structure used in these investigations.

Code Structure

Figure 30 illustrates the approach. Text composed of sequences of library identifiers (including one to indicate that a new library prototype pattern must be inserted) enters on the left. This data stream is then split into several new data streams to be separately coded and then concatenated to form the complete output. The function of the latter step is to allow the various non-stationary statistical components of USEEM text to be separately treated.

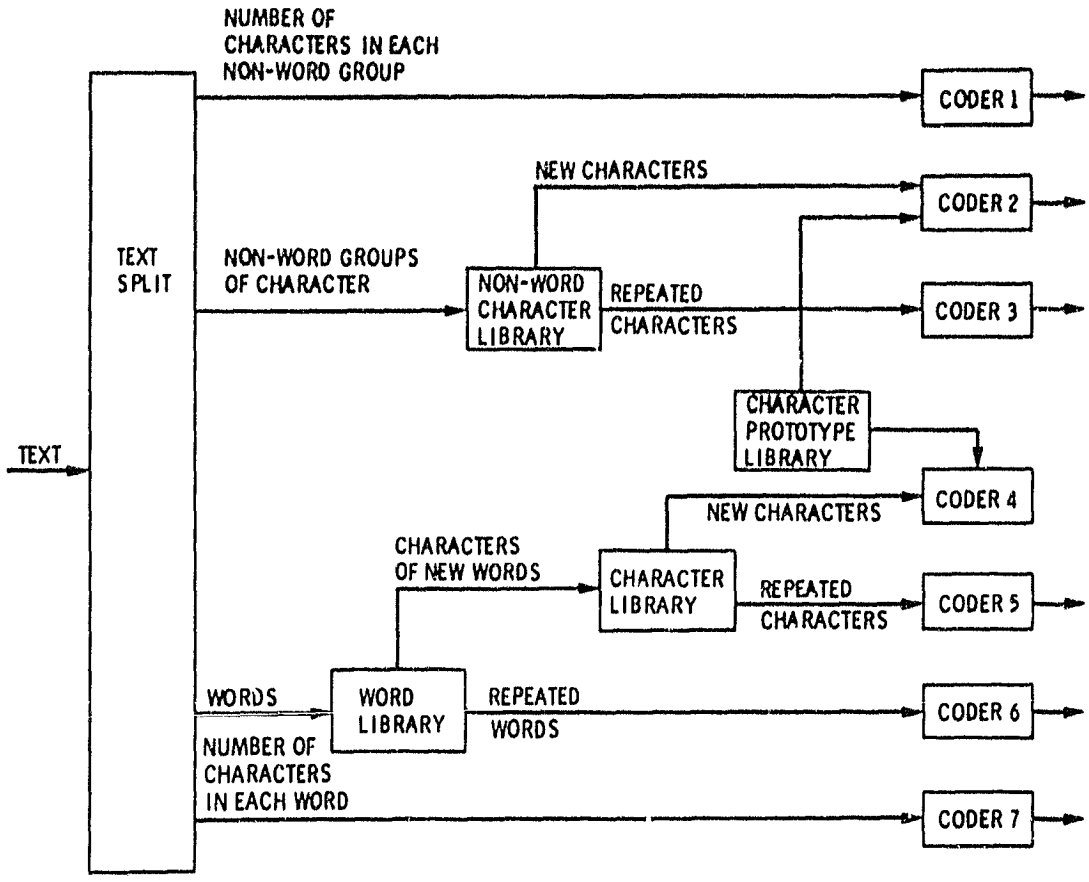


Fig. 30. Text Coding Structure.

Data is treated as making up either words or non-word groups of characters. Non-word groups include blank characters, special infrequently occurring symbols, end-of-line markers, etc. Sequences of words and non-word groups are accompanied by coded data sequences identifying how many characters are in each group. The coding of word and character sequences follows a structure much like that of the USEEM prototype library itself. Identical word and character libraries are developed and maintained at sending and receiving sites. The purpose of this step is to "learn" the statistical characteristics of the language. Initial work accounted for the relative frequency of occurrence of words and characters as well as the correlation between adjacent characters (e.g., a 'u' following a 'q').

All of the steps leading to the boxes labelled "coder 1" to "coder 7" act as preprocessors which produce memoryless sources with symbols 0, 1, 2, ... such that

$$p_0 \geq p_1 \geq p_2 \geq \cdots \tag{32}$$

is well approximated. This is the required condition to make effective use of the adaptive variable length coding techniques defined in Refs. 2-4. Hence, each of the "coders" in Fig. 30 can be a particular version of the latter algorithms.

APPENDIX A

Software for Universal Noiseless Coding

Reprint of Paper Published in the Proceedings of the 1981
International Conference on Communications
Denver, Colorado, June 1981

SOFTWARE FOR UNIVERSAL NOISELESS CODING

Robert F. Rice and Alan P. Schlutsmeyer

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Abstract

Discrete data sources arising from practical problems are generally characterized by only partially known and varying statistics. Practical adaptive techniques for the efficient noiseless coding of a broad class of such data sources have been developed at JPL. These techniques have now been implemented in ANSI-standard Fortran IV and made available to researchers through NASA's Computer Software Management and Information Center (COSMIC).

This paper describes the software package and the algorithms upon which it is based. These algorithms have exhibited performance only slightly above all entropy values when applied to real data sources with stationary characteristics. However, performance considerably under a measured average data entropy may be observed when data characteristics are changing over the measurement span.

These easily implemented algorithms are applicable to virtually any alphabet size arising in practice. A subset of these results is a large class of efficient adaptive coders for binary memoryless sources characterized by an unknown or varying statistic.

INTRODUCTION

Discrete data sources arising from practical problems are generally characterized by only partially known and varying statistics. Earlier papers[1], [2] provided the development and analysis of some practical adaptive techniques for the efficient noiseless coding of a broad class of such data sources. Specifically, these algorithms were developed for efficiently coding discrete memoryless sources which have known symbol probability ordering but unknown values. A general applicability of these algorithms to solving practical problems is obtained because most real data sources can be simply transformed into this form by appropriate preprocessing.

As part of the evolution of this set of "code operators" described in Refs. 1 and 2 a Fortran IV software package was developed which basically matches each code operator and inverse with a pair of corresponding subroutines. The existence of this software, now available through the Computer Software Management and Information Center (COSMIC)[3], should ease the burden of a potential user to determine the applicability, performance and appropriate options of these algorithms for his specific [illegible] system problem.

The intent of this paper is to provide an overview of the universal noiseless coding algorithms as well as their relationship to the now available Fortran implementations. Readers considering investigating the utility of these algorithms for actual applications should consult both COSMIC and Refs. 1 and 2. Examples of applying these techniques are given in Refs. 4-6.

Reversible Preprocessing

Removing correlations. In real problems where samples of a data sequence are correlated with themselves or with a priori information, there is usually some simple transformation which results in new sequences wherein the samples are approximately independent. More important, the uncertainty in what the sample values will be is usually greatly reduced. The less uncertainty there is the greater the potential for reducing the average bits required to code. Examples of such memory reducing operations include taking differences between adjacent samples along a television line, successive states of a Markov source or run lengths from a run length coder.

Symbol probability ordering. Given q possible symbols resulting from correlation removing

operations it is a simple matter to first relabel them into the integers 0, 1, 2, ··· q-1. Then letting $\tilde{P} = \{p_i\}$ be the probability distribution of 0, 1, 2, ·· q-1 we note that for a wide class of practical problems, the probability ordering of symbols is a priori known (or at least well approximated). In fact for many problems this ordering tends to change very little even as the actual $\tilde{P}$ values may be changing dramatically (consider the independent difference samples along a television scan line). It is then a simple matter to relabel source symbols, if necessary, so that the following conditions are well approximated.

$$P_0 \geq P_1 \geq P_2 \cdots \geq P_{q-1} \qquad (1)$$

Changing $\tilde{P}$. Most real world problems are characterized by changing and poorly defined values of $\tilde{P}$. $\tilde{P}$ may vary simply because short sequences are the result of preprocessing different data sources. There may be long and short term statistical variations in a single data source. Other than meeting condition (1), $\tilde{P}$ may not be known at all.

The modified coding problem may now be summarized in Fig. 1. The net result of correlation removing operations and symbol relabeling is, for many practical problems, an approximately memoryless source with symbols 0, 1, 2, ··· q-1 with generally unknown and varying probability distributions approximating condition (1). It is the efficient coding of such modified sources that the code operators of Refs. 1-3 provide practical solutions for.

Notations and Definitions

Using the notation of Refs. 1 and 2,

$$\mathscr{V}(\tilde{X}) \qquad (2)$$

will be used to specify the length of any sequence $\tilde{X}$ in samples or in bits (of its standard fixed length binary representation if $\tilde{X}$ is not already binary).

Performance measures. Given the discrete symbol probability distribution $\tilde{P} = \{p_i\}$ the entropy $H(\tilde{P})$ is defined by

$$H(\tilde{P}) = -\sum_i p_i \log_2 p_i \text{ bits/sample} \qquad (3)$$

When properly used, $H(\tilde{P})$ can be a useful practical tool in assessing how well a particular coding algorithm performs.

If $\tilde{Z}$ is an infinite sequence of samples from a memoryless source with fixed and known symbol probability distribution then $H(\tilde{P})$ represents the minimum possible expected bits/sample required to represent $\tilde{Z}$ using any coding technique. But as we have just noted, most practical problems which can be transformed by preprocessing into equivalent memoryless problems are characterized by

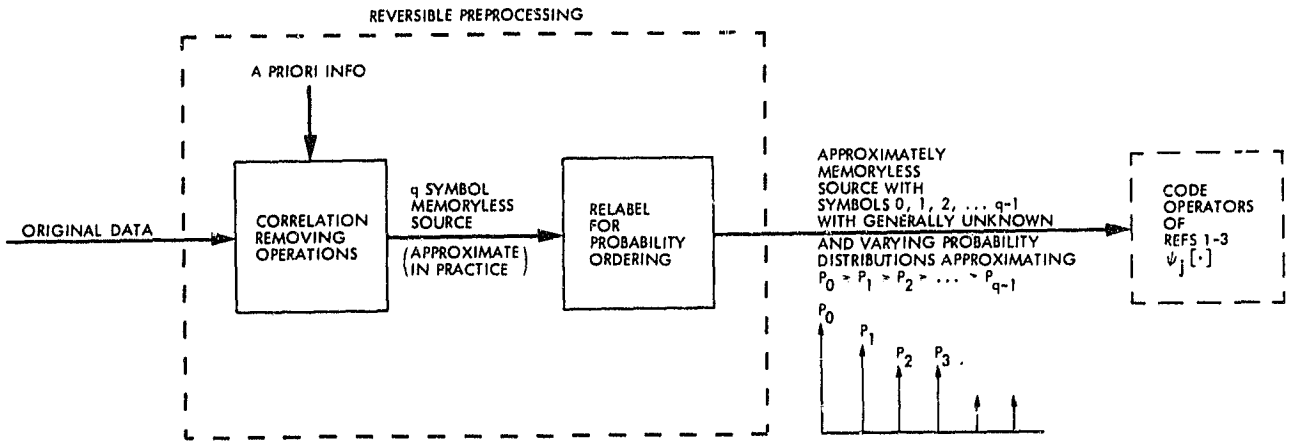


Fig. 1. Reversible Preprocessing

changing or possibly unknown distributions. In practice, it is generally difficult if not impossible to meaningfully model the way in which $\tilde{P}$ changes, although the fact that it changes may be quite obvious. Consequently, the equivalent "bounds" for real data sources with changing $\tilde{P}$ are difficult to come by.

Except where explicitly noted, the stated performance of a particular code operator will be based on measured performance using real data. $H(\overline{P})$ provides the desired practical measure of performance where $\overline{P}$ is the average symbol probability distribution over the measurement span. If the real data has a somewhat uniform statistical character over the measurement span then $H(\overline{P})$ represents a practical bound to average per sample performance of any code operator. An algorithm is performing efficiently if its measured average performance is close to $H(\overline{P})$. However, if data character changes significantly over the measurement span, it may be possible to obtain average per sample performance under the measured $H(\overline{P})$ by adapting the coding to suit the changes. $H(\overline{P})$ is still a useful guide in those cases and does in fact bound the best performance available with a single code (e.g. a Huffman code designed for $\overline{P}$).

Code operators. A code operator is a reversible mapping of a preprocessed data sequence (Fig. 1) into a binary output sequence. Code operator structures developed in Refs. 1 and 2 generally have many possible internal parameters. The notational convention adopted for identifying operators (structures) is to subscript and superscript the symbol psi (ψ) and occasional other symbols and implicitly assume that a detailed specification could be obtained by reference to a parameter string (e.g. the calling parameters of the COSMIC software). For example the operation of code operator $\psi_j[\cdot]$ on data sequence $\tilde{X}$ produces the coded binary sequence

$$\tilde{Y} = \psi_j[\tilde{X}] \tag{4}$$

requiring

$$\mathscr{L}(\psi_j[\tilde{X}]) \text{ bits} \tag{5}$$

$\tilde{Y}$ can be decoded by applying the inverse operation $\psi_j^{-1}[\cdot]$ so that

$$\psi_j^{-1}[\tilde{Y}] = \tilde{X} \tag{6}$$

Elements of a parameter string generally needed in the definition of $\psi_j[\cdot]$ would be the input alphabet size, q, and the length of $\tilde{X}$. Many other parameters may be required depending on the specific $\psi_j[\cdot]$.

Estimator and bound. An extremely practical characteristic of these noiseless code operators is that estimates (actually bounds) of actual performance can be obtained basically as simple functions of the sum of input samples. This can simplify performance assessments as well as aid in the creation of new algorithms. The notation convention adopted for identifying these estimators is to subscript and superscript gamma (γ) in the same manner as the corresponding code operator. Continuing the example above we have the estimate and bound

$$\gamma_j(\tilde{X}) \approx \mathscr{L}(\psi_j[\tilde{X}]) \tag{7}$$

and

$$\gamma_j(\tilde{X}) \geq \mathscr{L}(\psi_j[\tilde{X}]) \tag{8}$$

Performance Summary

The potential user of these code operators would first seek to model his data source to develop an appropriate "reversible preprocessor" which meets the objectives outlined in Fig. 1. The remaining problem is then to efficiently code (i.e. close to the entropy, $H(\overline{P})$) a discrete memoryless source with varying or unknown entropy values. The algorithms in Refs. 1-3 offer a practical solution to this problem by providing options which exhibit efficient performance for any range

of data entropies. A special subset of these algorithms is a class of binary memoryless code operators capable of performing close to the binary entropy function as the (a priori) unknown probability of a zero or one varies between 0.0 and 1.0.

In each application a user may avoid unnecessary complexity by selecting operators which assure efficient performance only over the entropy range of his specific problem.

CODE OPERATORS

The discussions within this section assume that input data is the result of reversible preprocessing operations as described in Fig. 1.

Basic Compressor

The "Basic Compressor" is defined as code operator $\psi_4[\cdot]$. $\psi_4[\cdot]$ selects between four code operators, $\psi_0[\cdot]$, $\psi_1[\cdot]$, $\psi_2[\cdot]$, and $\psi_3[\cdot]$, choosing the operator which will most efficiently represent the input block of samples $\tilde{X}$. The result of applying $\psi_4[\cdot]$ to $\tilde{X}$ is given by the binary sequence

$$\psi_4[\tilde{X}] = ID * \psi_{ID}[\tilde{X}] \qquad (9)$$

where * denotes concatenation, $\psi_{ID}[\tilde{X}]$ is the binary output sequence resulting from the application of the selected code operator $\left(\psi_0[\cdot] - \psi_3[\cdot]\right)$ to $\tilde{X}$, and ID is a two bit binary sequence (00, 01, 10 or 11) indicating which code operator was chosen. $\psi_4[\cdot]$ performs efficiently over the entropy range of approximately 0.7 to 4.0 bits/sample because at least one of the four "options" performs well at any given point of this range. $\psi_4[\cdot]$ is thus adaptive. The parameters needed for $\psi_4[\cdot]$ are the input block length J and the number of values that a sample of $\tilde{X}$ may take on, q. A smaller block size increases the rate at which the code options may be changed but also incurs a higher per sample overhead of 2/J bits/sample. For most applications net performance is rather insensitive to changes in J for all but very small or very large J. A convenient practical choice is J=16.

Typical average measured performance for the Basic Compressor $\psi_4[\cdot]$ is shown in Fig. 2 compared to a plot of measured average entropy, $H(\overline{P})$. Note that when distributions are stable performance lies slightly above $H(\overline{P})$, whereas when distributions vary considerably performance under $H(\overline{P})$ may be observed.

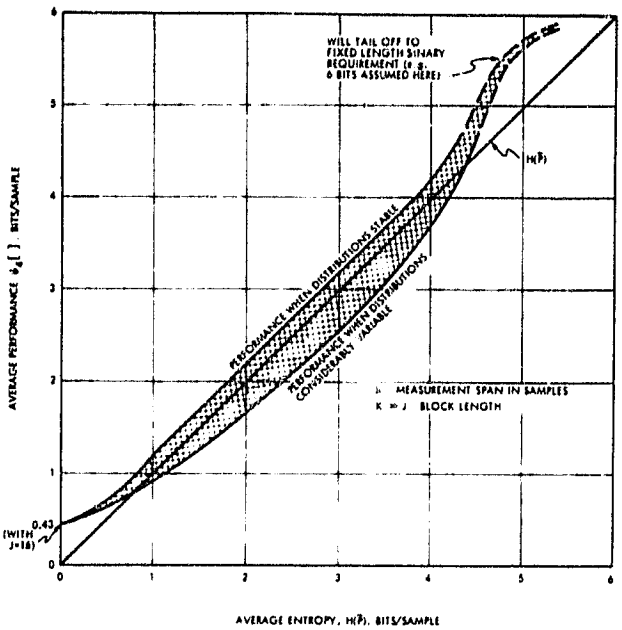


Fig. 2. Typical Average Performance of Basic Compressor $\psi_4[\cdot]$

Related operators. Given a sequence $\tilde{Y}$ of N preprocessed samples, operator $\psi_6[\cdot]$ will first partition $\tilde{Y}$ into blocks of J samples each (lengthening the last block if J does not divide N) and then code each block with $\psi_4[\cdot]$. $\psi_5[\cdot]$ is simply a special case of $\psi_6[\cdot]$ where J divides N.

Higher Entropies

Operator $\psi_8[\cdot]$ adds to $\psi_6[\cdot]$ additional "split-sample" modes which extend efficient performance above 4 bits/sample entropies. $\psi_6[\cdot]$ by itself is considered split-sample '1'. A coded $\tilde{Y}$ sequence is preceded by a mode identifier in much the same manner as for $\psi_4[\cdot]$ in (9). The first bit of mode identification provides one additional mode and extends efficient performance upwards by one bit/sample to 5 bits/sample. Adding a second identification bit allows two more split-sample modes and extends the range of efficient performance by another two bits/sample, and so on. There is no

fundamental limit to this operation. It is up to the user to select the number of modes which meet his objectives.

A special simplification of $\psi_8[\cdot]$ can be defined when N=J. This operator, $\psi_8^J[\cdot]$, has been implemented in an application to NOAA weather satellite images[5] and is part of the image compression system on the Galileo Project.[6]

Lower Entropies

$\psi_9[\cdot]$ provides the ability to achieve efficient performance at very low entropies without sacrificing performance at the higher entropies. The structure of $\psi_9[\cdot]$ is illustrated in Fig. 3 for a preprocessed data sequence $\tilde{Z}$.

As shown, the SPLIT$[\cdot]$ function splits $\tilde{Z}$ into a sequence of the non-zero samples $\tilde{\theta}$ and a binary sequence $\tilde{D}$ which identifies which samples of $\tilde{Z}$ are non-zero. Separate coding of $\tilde{D}$ and $\tilde{\theta}$ using $\psi_\beta^j[\cdot]$ and $\psi_\theta[\cdot]$ yields the desired result. $\tilde{\theta}$ has characteristics satisfying the preprocessing requirements of Fig. 1 so that $\psi_\theta[\cdot]$ can be identified as a form of $\psi_8[\cdot]$. $\tilde{D}$ appears as a binary memoryless source with unknown and changing statistic

$$p_0 = \Pr\left[\begin{array}{l}\text{sample of } \tilde{D} \\ \text{equals zero}\end{array}\right] \qquad (10)$$

$\psi_\beta^j[\cdot]$ in Fig. 3 is a class of code operators to efficiently code such sources. Appropriate choice of $\psi_\beta^j[\cdot]$ yields $\psi_9[\cdot]$ performance characteristics shown in Fig. 4 for data with slowly changing characteristics and input alphabet size of 256.

The operator $\psi_{10}[\cdot]$ chooses between $\psi_8[\cdot]$ and $\psi_9[\cdot]$. Roughly the same performance is observed but certain subtle implementation advantages emerge. The reader is referred to Refs. 1 and 2.

Binary Coders

The coders $\psi_\beta^j[\cdot]$ noted above are generally useful with or without the structure of $\psi_9[\cdot]$ preceding them. Basically $\psi_\beta^j[\cdot]$ first preprocesses the binary data into a form which allows the non-binary operators $\psi_4[\cdot]$ - $\psi_{10}[\cdot]$ to be used. A binary operator of this form which used $\psi_9[\cdot]$ internally following this additional preprocessing would be labeled $\psi_\beta^9[\cdot]$ whereas one that used $\psi_4[\cdot]$ internally would be labeled $\psi_\beta^4[\cdot]$. Since $\psi_\beta^9[\cdot]$ or $\psi_\beta^{10}[\cdot]$ use a $\psi_9[\cdot]$ operator internally they must also employ a binary operator internally (see the structure of $\psi_9[\cdot]$ in Fig. 3). This internal binary operator could also be $\psi_\beta^9[\cdot]$ and so on. Thus the $\psi_\beta^9[\cdot]$ or $\psi_\beta^{10}[\cdot]$ structure could allow for an infinite tree of code operators. From

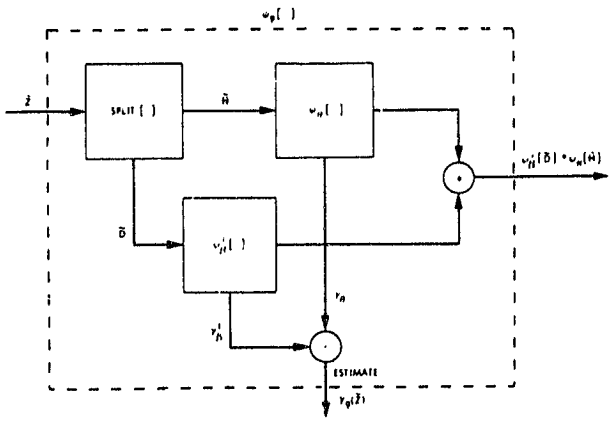


Fig. 3. General Form, Operator $\psi_9[\cdot]$

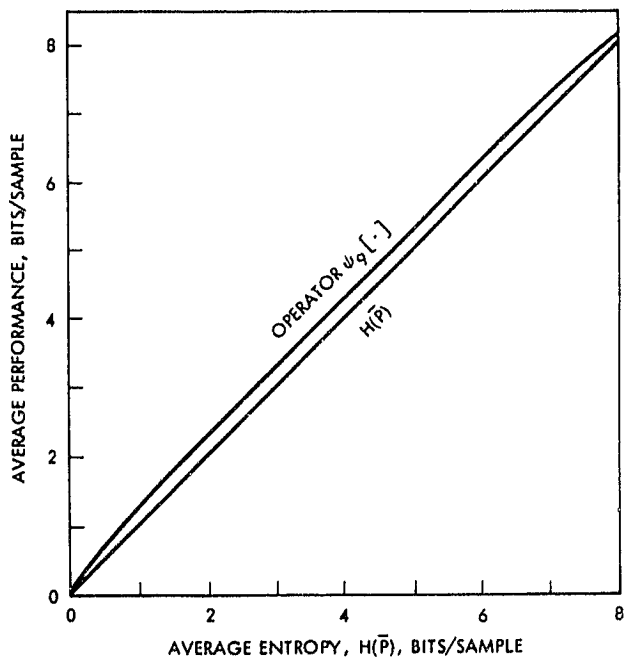


Fig. 4. Measured Average Performance, $\psi_9[\cdot]$

a performance point of view, each additional level of $\psi_\beta^9[\cdot]$ or $\psi_\beta^{10}[\cdot]$ improves performance in the vicinity of zero entropy. A user must assess whether his particular data problem merits the added complexity. The COSMIC routines allow for $\psi_\beta^{10}[\cdot]$ up to three levels deep. A bound to the performance of such a code operator for an ideal binary memoryless source with unknown p_0 is shown in Fig. 5 in comparison to the binary entropy function

$$H_\beta(p_0) = -p_0 \log_2 p_0 - (1-p_0) \log_2 (1-p_0) \tag{11}$$

A simplification of $\psi_\beta^j[\cdot]$ can be obtained when it is a priori known that $p_0 \geq 1/2$ or $p_0 \leq 1/2$. Such a modified operator is identified by $\psi_{\beta'}^j[\cdot]$.

SOFTWARE RELATIONSHIPS

The COSMIC software package[3] implementing the noiseless coding algorithms described above is written in ANSI Fortran IV. It has been run in present form on an SEL 32/55 and an IBM 370/158 and we believe it could be used on most other systems without modification. All computations are done using integer arithmetic. Binary input and output strings are accomplished using 1-bit-per-integer-word (16 bits) representation. While demanding more memory than 1-bit-per-bit representation the processing is generally much faster using Fortran.

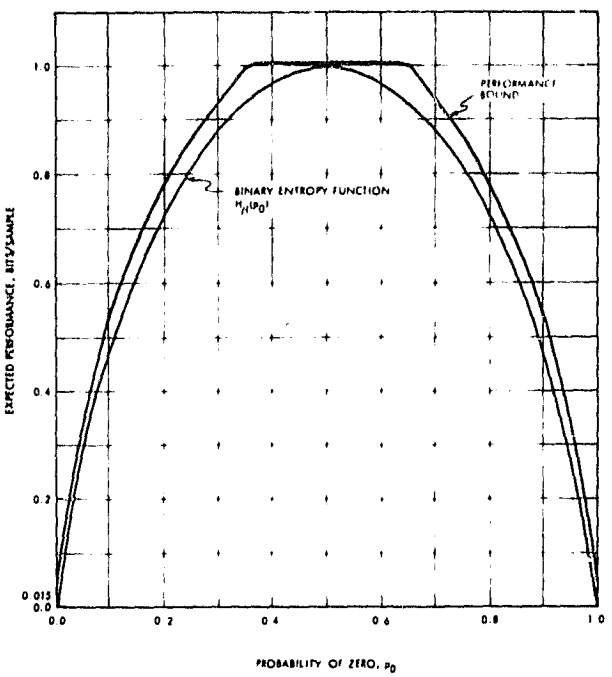


Fig. 5. Bounds to Expected Performance of Multi-Level $\psi_\beta^{10}[\cdot]$ on Ideal Memoryless Source, Unknown but Constant p_0

These routines were written with the research and/or engineering user in mind. As such they have not been optimized for execution efficiency or memory conservation. A conscientious application of optimizing techniques would doubtless effect significant improvements.

General Implementation Instructions

The software package consists of two files of Fortran-IV source code. The first file contains all the subroutines for coding and decoding, and comprises about 2800 Fortran source statements. It is most convenient to compile the routines and place them in a disk subroutine library for subsequent use by calling programs.

A second file provides a test program. It is recommended that this program be compiled and executed to test the subroutines in file 1, to ensure they are operating correctly.

Neither file contains job control statements since these will vary from machine to machine. The required operations of compiling, cataloging and executing should be familiar enough to any programmer to make the job control problem an easy one.

Naming Conventions

The subroutines provided are named by a standard naming convention. Basic support subroutines have functionally mnemonic names. The names for coders, decoders and estimators were chosen to closely match the notation established in Refs. 1 and 2 as well as earlier sections. For example, subscripting and superscripting psi (ψ) to identify code operators is replaced by subroutine names beginning with 'SI' and ending with 'I' if the routine is a decoder (inverse). Letters or numbers between 'SI' and 'I' modify the name further. For example operator $\psi_{10}[\cdot]$ with two levels of

internal tree structure becomes SI102 in software. Similarly the decoder for binary operator $\psi_\beta^{10}[\cdot]$ with two levels of internal tree structure becomes SB102I (where the 'I' in SI was dropped to enable a six-letter word).

The estimators for operator performance, identified by subscripting and superscripting gamma (γ) in Refs. 1 and 2 become GAxxx in the COSMIC software.

In summary, the naming convention for subroutines should enable a smooth transition from technical definitions to practical software application. Further details are provided in Ref. 3.

Calling Arguments

In order to facilitate understanding of the various coding, decoding and estimating routines, calling argument names and variable names within the Fortran code have been used consistently as much as possible. Each subroutine contains documentation describing the calling arguments specific to itself. More general documentation concerning the rules of use of the arguments, their inter-relationships and the overall structure of the subroutine set is provided as part of the software distribution package available from COSMIC.

ACKNOWLEDGMENT

The research described in this paper was carried out by the Information Processing Research Group of the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the U. S. Postal Service through an agreement with the National Aeronautics and Space Administration.

REFERENCES

1. R. F. Rice, "Some Practical Universal Noiseless Coding Techniques," JPL Publication 79-22, Jet Propulsion Laboratory, Pasadena, CA, March 15, 1979.

2. R. F. Rice, "Practical Universal Noiseless Coding," SPIE Symposium Proceedings, Vol. 207, San Diego, CA, August 1979.

3. A. Schlutsmeyer, R. Rice, "Universal Noiseless Coding Routines," Case NPO 15451, Computer Software Management and Information Center (COSMIC), Athens, Ga., November 1980.

4. R. F. Rice, "An Advanced Imaging Communication System for Planetary Exploration," Vol. 66 SPIE Seminar Proceedings, Aug. 21-22, 1975, pp. 70-89.

5. R. F. Rice, A. P. Schlutsmeyer, "Data Compression for NOAA Weather Satellite Systems," Vol. 249, Proceedings 1980 SPIE Symposium, San Diego, CA., July 1980.

6. R. F. Rice et. al, "Block Adaptive Rate Controlled Image Data Compression," Proceedings of 1979 National Telecommunications Conference, Washington, D.C., Nov. 1979.

APPENDIX B

DIRECTED MARKOV PREDICTION

The following discussion describes the concepts behind the Bell Labs Markov Predictor[8] and an extension called "Directed Markov Prediction."

STANDARD MULTI-STATE PREDICTOR

The standard approach to a 16 state predictor, much like the Bell Labs predictor, is diagrammed below in Fig. B-1.

Here y is the binary sample to predict and the $\{x_j\}$ are known data samples to be used in the prediction. The $\{x_j\}$ take on 16 possible states, $S_i = x_1\, x_2\, x_3\, x_4$ (e.g., $S_0 = 0000$, $S_1 = 0001$, etc.).

For a given state, S_i, the prediction y_p is chosen such that

$$\begin{aligned} \Pr[y = y_p | S_i] &= \max\{\Pr[y = 0 | S_i], \Pr[y = 1 | S_i]\} \geq \tfrac{1}{2} \\ &= \Pr[\text{correct} | \text{ state } S_i]. \end{aligned} \tag{B-1}$$

The overall probability of being correct is then

$$P_c = \sum_i \Pr[S_i] \Pr[\text{Correct} | S_i]. \tag{B-2}$$

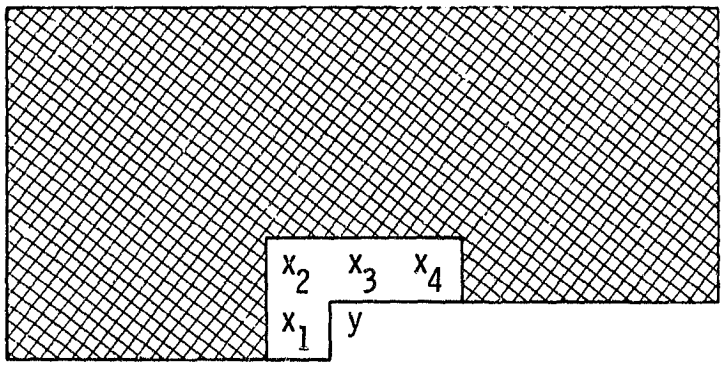


Fig. B-1. Prediction Samples.

One should expect to improve performance by increasing the state set S_i. Others have done so to a limited degree but have been limited by the exponential growth of states as more samples are added. Ideally the rather large crosshatched area in Fig. 1 (perhaps encompassing whole letters) should be used in the prediction. The following approach seeks to obtain most of these benefits without the corresponding complexity problem.

EXPANDING THE USE OF SURROUNDING DATA

Letting S_i remain as any one of the original 16 states and ζ as any one of the known data samples of the crosshatched region of Fig. B-1, we can rewrite (B-1) as follows:

$$\Pr [y = y_p | S_i] = \Pr [y = y_p \text{ AND } \{\zeta = 1 \text{ OR } \zeta = 0\} | S_i]. \quad \text{(B-3)}$$

This can be written as

$$\Pr [\zeta = 1 | S_i] \Pr [y = y_p | S_i, \zeta = 1] + \Pr [\zeta = 0 | S_i] \Pr [y = y_p | S_i, \zeta = 0]. \quad \text{(B-4)}$$

Now choose ζ^* such that

$$\Pr [y = y_p | S_i, \zeta^*] < \Pr [y = y_p | S_i, \overline{\zeta}^*]. \quad \text{(B-5)}$$

(That is, knowing $\zeta = \zeta^*$ yields a smaller probability that y equals the original prediction y_p than does knowing that ζ equals the complement of ζ^*.)

Note that only the left hand side of (B-5) can be less than ½, otherwise (B-1) could not be true.

Change of Decision

Now find the set $\chi(S_i)$ of all ζ that satisfy

$$\Pr [y = y_p | S_i, \zeta^*] < \tfrac{1}{2}. \quad \text{(B-6)}$$

For each $\zeta = \zeta^*$ the best prediction should be changed from $y = y_p$ to its complement $\overline{y}_p$.

This changes state S_i probability of error from

$$P_e(S_i) = \Pr[\zeta = \zeta^* | S_i] \{1 - \Pr[y = y_p | S_i, \zeta^*]\} + \{\text{term when } \zeta = \overline{\zeta}^*\} \quad \text{(B-7)}$$

to,

$$P_e(S_i, \zeta = \zeta^*) = \Pr[\zeta = \zeta^* | S_i] \Pr[y = y_p | S_i, \zeta^*] + \{\bullet\}. \quad \text{(B-8)}$$

The improvement is given by

$$\Delta P_e(S_i, \zeta = \zeta^*) = \Pr[\zeta = \zeta^* | S_i] \{1 - 2 \Pr[y = y_p | S_i, \zeta = \zeta^*]\} . \quad \text{(B-9)}$$

Now order the $\zeta \epsilon \chi(S_i)$ by the measure in (B-9). The first one on the list will provide the largest improvement in P_e.

At this point we have a predictor which will make its original prediction for each state S_i unless the selected $\zeta \epsilon \chi(S_i)$ for that state has a value ζ^*, in which case the prediction is complemented. The actual selected samples from the crosshatched region may be different for each S_i.

Extending the Tree

By selecting the $\zeta^* \epsilon \chi(S_i)$ for each state S_i we have started the first branch of a predictor tree. To extend these branches note that the set of data samples $\zeta \epsilon \overline{\chi}(S_i)$ are those data samples which would not alter our initial decision. We can in fact order them by how much they tend to make the original decision more certain (those $\zeta \epsilon \chi(S_i)$ with values ζ^* we think can be excluded).

Now just as we found the $\zeta = \zeta^* \epsilon \chi(S_i)$ which caused a decision change from y_p to $\overline{y}_p$ we find the $\zeta^* \epsilon \overline{\chi}(S_i)$ which cause us to change that decision back, picking the one with the biggest impact. (Here conditioning is on the new states $\{S_i, \zeta^*\}$.)

The tree can also be extended from the state $\{S_i, \zeta^*\}$ for which we did not change initial prediction, y_p. The candidates here are of course the unused elements of the ordered $\zeta \epsilon \chi(S_i)$. Intuitively the best candidates should be at the top of the list.

The general procedure is outlined in Fig. B-2 where we have subscripted the different ζ^* by letters.

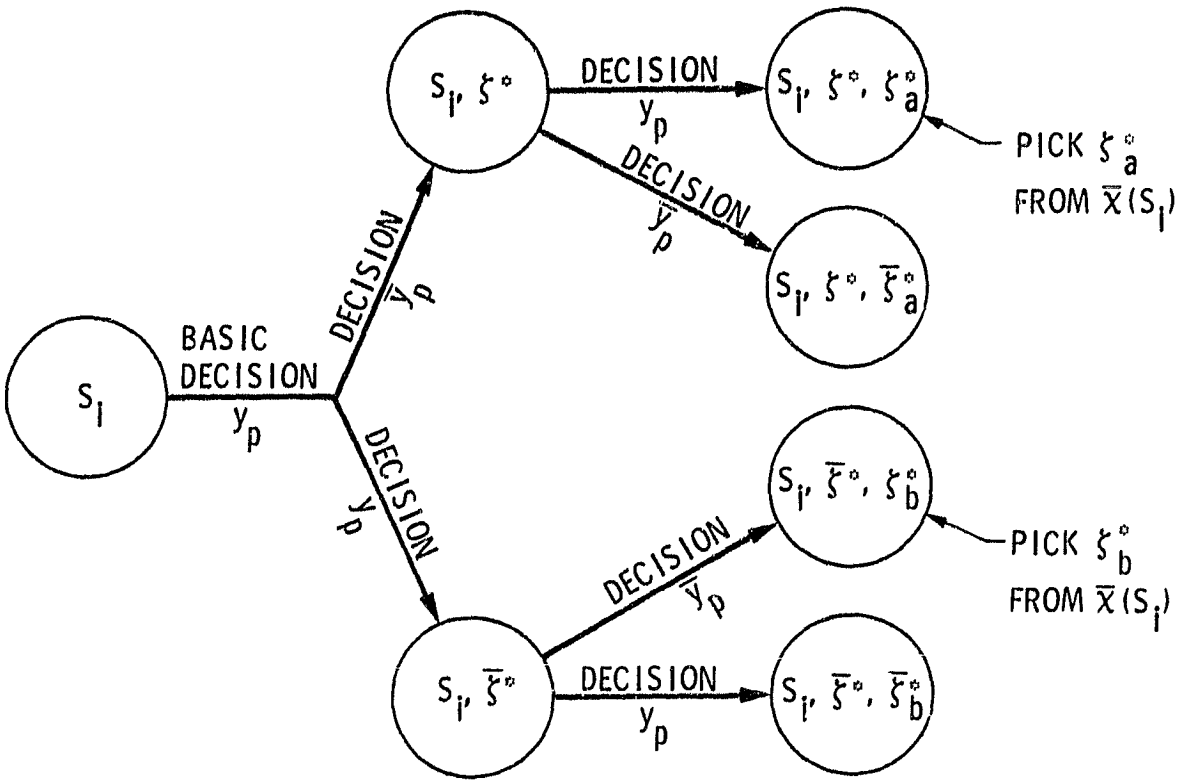


Fig. B-2. Prediction Tree.

EXAMPLES

Expanding Fig. B-1 to include a numbering of pixels in the cross hatched region leads to the diagram in Fig. B-3 which shows 56 additional pixels which would be used for prediction.

1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30	31	32	33
34	35	36	37	38	39	40	41	42	43	44
45	46	47	48	x_2	x_3	x_4	49	50	51	52
53	54	55	56	x_1	y					

Fig. B-3. Arrangement of Potential Predictor Samples.

In the following we will investigate extending state S_6 to include additional samples. This is a result taken from a 256 x 256 region. The basic state S_6 information is displayed in Fig. B-4.

1	1	0	POPULATION	227
0	y		PREDICTION	$y_p = 0$
			ERRORS = 22	

Fig. B-4. State S_6 Prediction.

The next step is to search all surrounding S_6 samples to see if adjoining any one of them to S_6 can improve our prediction ability.

In this example sample z_{48} is the best (and only choice). The predictor for state S_6 becomes that shown in Fig. B-5.

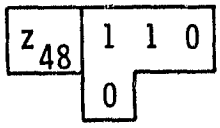


Fig. B-5. Extended S_6 Predictor.

For all the 227 times S_6 occurs, z_{48} = 1 33 times and zero the remaining 194 times. But of the 33 times z_{48} = 1, y equals the original prediction y_p = 0 17 times. Since 17/33 > 0.5 we should instead predict y_p = 1 when z_{48} = 1. This result is depicted below in Fig. B-6.

While this reduced the number of errors by only one, we can branch again from the newly defined state S_6 and z_{48} = 1. The best option in this case turns out to be z_{40}. The result is shown below in Figs. B-7 and B-8.

As shown when z_{40} = 1 the prediction is changed back to y_p = 0. The number of total errors is now reduced to 9.

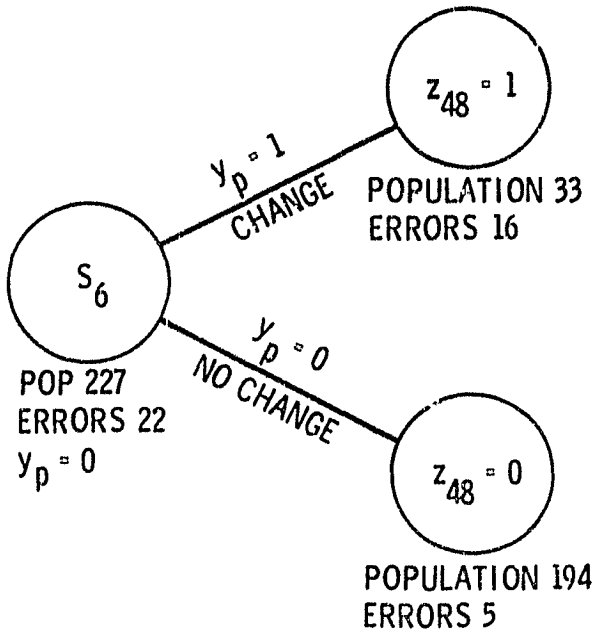


Fig. B-6. Adding z_{48}.

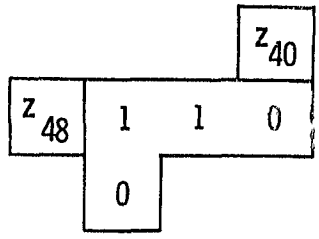


Fig. B-7. Extended S_6, $z_{48} = 1$ Predictor.

A predictor using this approach would check z_{48} when state S_6 was present. If $z_{48} = 0$ the predictor would predict zero and go onto the next sample. If $z_{48} = 1$ it would next check z_{40}. If $z_{40} = 1$ it would predict zero, whereas if $z_{40} = 0$ it would predict 1.

Our preliminary average results are not as overwhelming as this example. For a double branch as above, results suggested that a basic 16 state (Bell Lab) predictor would require a 10 to 20 percent greater rate.

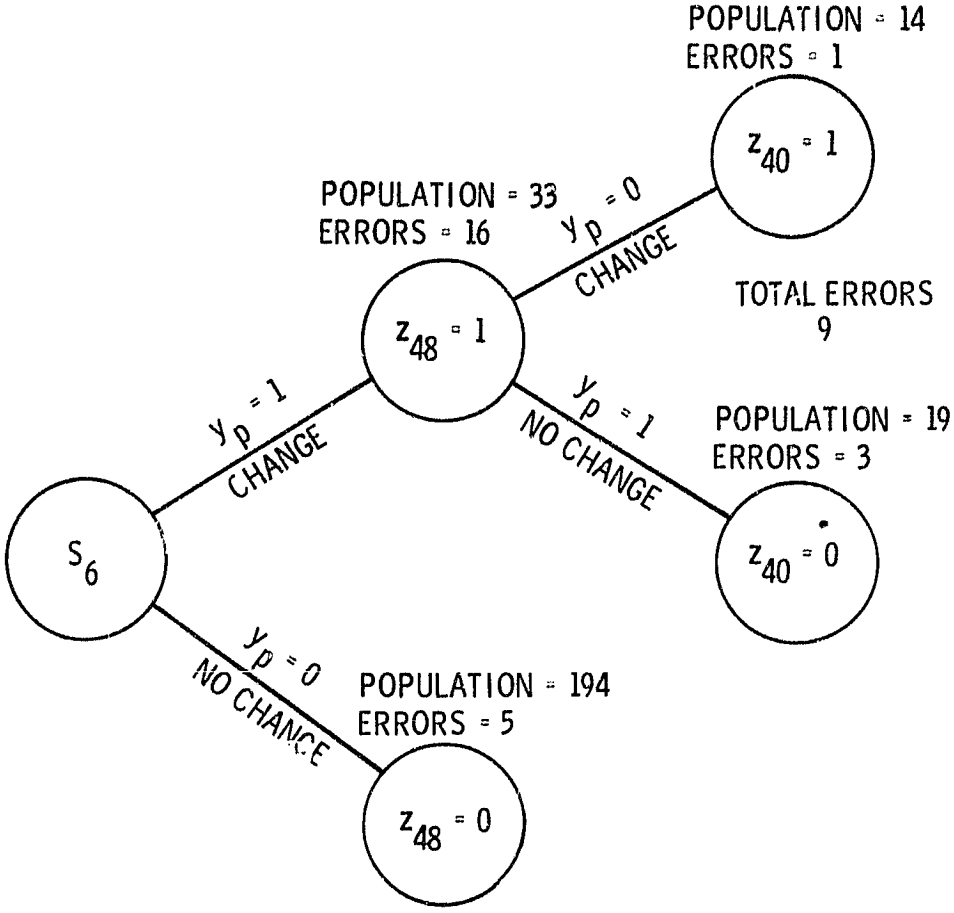


Fig. B-8. Adding z_{40}.

REFERENCES

1) A. Kegel, "The case for Postal-Service sponsored research in data compression techniques," **Internal Postal Service Report**, June 1981.

2) R. F. Rice, "Some practical universal noiseless coding techniques," **JPL Publication 79-22**. Jet Propulsion Laboratory, Pasadena, California, March 15, 1979.

3) R. F. Rice, "Practical universal noiseless coding," **SPIE Symposium Proceedings**, Vol. 207, San Diego, California, August, 1979.

4) R. F. Rice, A. P. Schlutsmeyer, "Software for universal noiseless coding," **Proceedings of the 1981 International Conference on Communications**, Denver, Colorado, June, 1981.

5) R. B. Arps, "The statistical dependence of run-lengths in printed matter," **Codierung, Nachrichtentech. Fachberichte,** Vol. 40, pp. 218-226, 1971.

6) H. G. Mussman and D. Preuss, "Comparison of redundancy reducing codes for facsimile transmission of documents," **IEEE Transactions on Communications**, Vol. COM-25, pp. 1425-1433, Nov. 1977.

7) R. Hunter, A. H. Robinson, "International digital facsimile coding standards," **Proceedings of the IEEE**, Vol. 68, No. 7, pp. 854-868, July 1980.

8) A. N. Netravali and F. W. Mounts, "Ordering techniques for facsimile coding: a review," **Proceedings of the IEEE**, Vol. 68, No. 7, July 1980.

9) J. L. Mitchell and G. Goertzel, "Yorktown dual-mode data compression and decompression system," **IBM Research Report 7490 (32362)**. Thomas J. Watson Research Center, Yorktown, New York, January 1979.

10) W. K. Pratt, et al., "Combined symbol matching facsimile data compression system," **Proceedings of the IEEE**, Vol. 68, No. 7, July 1980.

NASA—JPL—Coml., L.A., Calif.