https://ntrs.nasa.gov/search.jsp?R=19920005393 2020-03-17T14:16:26+00:00Z

JPL Publication 91-3

Some Practical Universal Noiseless Coding Techniques, Part III, Module PSI14,K+

Robert F. Rice

November 15, 1991

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. 91-3	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Some Practical Universal Noiseless Coding Techniques, Part IIIm Module PSI14, K+		5. Report Date November 15, 1991
		6. Performing Organization Code
7. Author(s) Robert F. Rice		8. Performing Organization Report No.
9. Performing Organization Name and Address JET PROPULSION LABORATORY California Institute of Technology 4800 Oak Grove Drive Pasadena, California 91109		10. Work Unit No.
		11. Contract or Grant No. NAS7-918
		13. Type of Report and Period Covered JPL Publication
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION Washington, D.C. 20546		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

This publication provides the algorithmic definitions, performance characterizations and application notes for a high-performance adaptive coding module. Subsets of these algorithms are currently under development in custom VLSI at three NASA centers. This report extends the generality of coding algorithms recently reported.

The module incorporated a powerful adaptive noiseless coder fro Standard Data Sources (i.e. sources whose symbols can be represented by uncorrelated nonnegative integers, where smaller integers are more likely than the larger ones). Coders can be specified to provide performance close to the data entropy over any desired Dynamic Range (of entropy) above o.75 bit/sample. This is accomplished by adaptively choosing the best of many efficient variable-length coding options to use on each short block of data (e.g., 16 samples). All code options used for entropies above 1.5 bits/sample are "Huffman Equivalent," but they require no table lookups to implement.

The coding can be performed directly on data that have been preprocessed to exhibit the characteristics of a Standard Source. Alternatively, a built-in predictive preprocessor can be used where applicable. This built-in preprocessor includes the familiar one-dimensional predictor followed by a function that maps the prediction error sequences into the desired standard form. Additionally, an external prediction can be substituted of desired.

This report further addresses a broad range of issues dealing with the interface between the coding module described here and the data systems it might serve. These issues include: multidimensional prediction, archival access, sensor noise, rate control, code rate improvements outside the module, and the optimality of certain internal code options.

17. Key Words (Selected by Author(s)) Spacecraft Instrumentation, Communications, Electronics and Electrical Engineering Information Theory		18. Distribution Statement Unclassified Unlimited	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 124	22. Price

ACKNOWLEDGEMENTS

The author gratefully acknowledges the support of numerous individuals who have contributed to the evolution of the techniques described herein, especially to Alan Schlutsmeyer for his assistance in editing this report and for his extensive simulation support that was instrumental to early developments. Particular thanks are offered to Jun-Ji Lee for his key long-term technical assistance and simulation support; to Pen-shu Yeh for an innovative analysis that demonstrates the optimality of certain algorithms described here; to Warner Miller for his insight in initiating the NASA standardization of these algorithms and for the development of a program to implement very high-speed custom VLSI chip sets; and to Merv MacMedan for continued support of the standardization effort and assistance in final preparation of this report. Thanks also to the University of Idaho VLSI team, particularly Jack Venbrux, under the direction of Gary Maki, for numerous insights into hardware implementation requirements.

ABSTRACT

This publication provides the algorithmic definitions, performance characterizations and application notes for a high-performance adaptive noiseless coding module. Subsets of these algorithms are currently under development in custom VLSI at three NASA centers. This report extends the generality of coding algorithms recently reported.

The module incorporates a powerful adaptive noiseless coder for Standard Data Sources (i.e., sources whose symbols can be represented by uncorrelated non-negative integers, where smaller integers are more likely than the larger ones). Coders can be specified to provide performance close to the data entropy over any desired Dynamic Range (of entropy) above 0.75 bit/sample. This is accomplished by adaptively choosing the best of many efficient variable-length coding options to use on each short block of data (e.g., 16 samples). All code options used for entropies above 1.5 bits/sample are "Huffman Equivalent," but they require no table lookups to implement.

The coding can be performed directly on data that have been preprocessed to exhibit the characteristics of a Standard Source. Alternatively, a built-in predictive preprocessor can be used where applicable. This built-in preprocessor includes the familiar one-dimensional predictor followed by a function that maps the prediction error sequences into the desired standard form. Additionally, an external prediction can be substituted if desired.

This report further addresses a broad range of issues dealing with the interface between the coding module described here and the data systems it might serve. These issues include: multidimensional prediction, archival access, sensor noise, rate control, code rate improvements outside the module, and the optimality of certain internal code options.

CONTENTS

I. INTRODUCTION 1

BACKGROUND, REVIEW AND ORIENTATION 2

Notation 4

Standard Source $\tilde{\delta}^n$ 5

Coder Performance 7

GETTING MORE SPECIFIC 9

Input 9

To Preprocess or Not To Preprocess 10

Parameter Ranges 10

II. BUILT-IN PREPROCESSOR 13

Predictor 13

Error Signal 15

Mapping into the Integers 16

III. ADAPTIVE VARIABLE-LENGTH CODING 19

PSI14 21

Fundamental Sequence, PSI1 21

Code Operator PSI0 24

The Unity Code Operator, PSI3 28

Operators PSI2 and PSI4 28

Split-Sample Modes 29

CODE OPERATOR PSI14 36

Definition 36

Decision Criteria 40

Baseline PSI14 Performance 43

Coding for Very High Entropies 45

CODE OPERATOR PSI14,K 47

Performance for K = 0 48

Performance for K > 0 48

PSI14,K vs PSI14,0 for K > 0 51

Stretch PSI14 (sPSI14) 51

CURRENT VLSI IMPLEMENTATIONS 53

IV. OUTSIDE THE MODULE 55

EXTERNAL SPLIT-SAMPLES 55

DECODING PREPROCESSED DATA 57

One-Dimensional Predictor 59
External Predictor 61
ID CODING 73
Rearranging the Coding of $\tilde{X}$ 74
The Code Rate 75
Application to PSI14,K+ 78
RATE CONTROL 86
The FIFO Buffer Constraint 86
A Multispectral Frame Buffer Definition 101

APPENDICES

A MORE ON THE MAPPING OF PREDICTION ERRORS 108
B MORE ON THE PSI1,k SPLIT-SAMPLE OPTIONS 111
C COMPARING PSI0,1 WITH PSI1,0 114
D GAIN/OFFSET SENSOR NOISE 117

REFERENCES 123

Figures

1. General Coding Module 3
2. $\tilde{\delta}^n$ Sample Probabilities 6
3. Desired Average Coder Performance for a Specific Application 8
4. Module PSI14,K+ High-Level Functional Block Diagram 9
5 Built-in Preprocessor Functional Block Diagram 13
6. External 2-D Predictor Interfacing With Module Preprocessor 14
7. General-Purpose Adaptive Coder, PSI11 19
8. Average PSI1 Performance 23
9. Average PSI0 Performance 27
10. Average Performance, PSI4 29
11. Basic Split-Sample Operator 31
12. Split-Sample Coder, PSIi,k 33
13. PSI1,2 Example 34
14. Average Performance, PSI4,k 35

15. Baseline PSI14 Performance, $N = 8, \lambda = 0, 1, n = 12$ 43
16. Baseline PSI14 Performance, $N = 4, \lambda = 0, 1, n \geq 6$ 44
17. Split-Sample Coder PSI14,K 48
18. Skipping an Option 53
19. Internal vs External Splits 55
20. Reference Sample Extraction 60
21. Two-Dimensional Array 62
22. 2-D Prediction Regions, Common Format: Top-to-Bottom, Left-to-Right Coding 63
23. Images Formed by Swept-Line Sensors 65
24. 1-D Down-Track Prediction Strategy for an Uncorrected Swept-Line Sensor 67
25. Example: Changing the Order of Coding 2-D Sub-Arrays 69
26. PSI16, Coding the Identifiers 75
27. Code Identifier Code Rate, Fixed-Length Representation 78
28. $\Re_F(N,J)$ for a PSI14,K+ Primary Coder 79
29. Hypothetical Entropy Graph 80
30. Expanded Hypothetical Entropy Graph 80
31. PSI16 Coder of Identifiers Example 84
32. The FIFO Buffer 87
33. Standard Format Line Coding and Buffering 88
34. Example: Line-Length Distributions for Single Image, Buffer Length = Average Line Length 91
35. Truncated Image Lines 91
36. Coding Alternate Lines in Opposite Directions 93
37. A Blowup of the Upper Left-Hand Corner of Fig. 36 94
38. Arranging for Bit Priority 96
39. Example: Normalized Line-Length Distribution for Bit Priority PSI? of Fig. 38 97
40. Ordering Samples by Odd/Even 100
41. Alternating the Ordering of Samples by Odd/Even 101
42. HIRIS Entropy Variability–AVIRIS Image 102
C-1. $\gamma_{0,1}$ Performance Bounds 116
D-1. Signal Plus Noise 119
D-2. Difference Signal Plus Noise 120
D-3. Signal Plus Noise Entropies 121

Tables

1. Basic Mapping of Δ_i into the Integers, δ_i 16
2. 8-Word 3-Tuple Code, cfs[i] 25
3. PSI14 Specification Example for $N = 6$, $\lambda = 3$, $n \geq 7$ 39
4. PSI14 Decision Regions, $N = 8$, $\lambda = 0$ 42
5. PSI14 Dynamic Ranges for 8-option Coders, $J = 16$ 46

SOME PRACTICAL UNIVERSAL NOISELESS CODING TECHNIQUES, PART III, MODULE PSI14,K+

I. INTRODUCTION

References 1–5 provide the development and analysis of some practical adaptive techniques for efficient noiseless (lossless) coding of a broad class of data sources. Specifically, algorithms were developed for efficiently coding discrete memoryless sources that have known symbol probability ordering but unknown probability values. General applicability of these algorithms is obtained because most real data sources can be simply transformed into this form by appropriate reversible preprocessing.

The applicability of noiseless coding to several high data rate NASA instruments has recently fostered the definition of a specific noiseless coding "module" called PSI14,K+ [6]. Extensions and modification of key adaptive coding algorithms from that earlier work are incorporated in this module, along with a standard preprocessor (denoted by the "+"). This PSI14,K+ definition evolved in an attempt to minimize hardware requirements without incurring a loss in performance. Two very similar subsets of the module have been implemented as CMOS VLSI chip sets by the Jet Propulsion Laboratory (JPL) and Goddard Spaceflight Center (GSFC) in collaboration with the University of Idaho's Microelectronics Center. The algorithmic definitions that specifically focus on these implementations are provided in Ref. 7 and are discussed here also.

The GSFC/U. of Idaho implementations include full custom 1.0-μm CMOS coder **and** decoder chips [8]. Both types of chip were recently tested successfully under laboratory conditions at input data rates up to 700 Mbits/s. By operating on sampled data quantized from 4 to 14 bits/sample, efficient coding performance can be expected over a range of entropies from 1.5 to 12.5 bits/sample.

JPL developed and recently tested 1.6-μm CMOS coder chips based on both gate-array and standard cell technologies [9]. These chips, while essentially implementing the same subset of PSI14,K+ algorithms, have thus far included fewer general-purpose features than the corresponding GSFC/U. of Idaho coder chip. The

Comet Rendezvous/Asteroid Flyby (CRAF)/Cassini project has recently initiated efforts to flight qualify an upgraded version of the gate-array design.

The primary purpose of this report is to present the functional, algorithmic and performance characteristics embodied in the most general PSI14,K+ module definition. Appropriate material from earlier references will be consolidated here in support of that presentation.

The first chapter provides the motivation and technical framework for a PSI14,K+ module specification. This includes basic notational quirks used throughout, performance goals and a key partitioning of the coding process into separate pre-processing and adaptive variable-length coding steps.

Chapters II and III focus individually on the specific PSI14,K+ module requirements for preprocessing and adaptive variable-length coding, respectively.

Finally, Chapter IV addresses a broad range of issues dealing with the interface between a PSI14,K+ module and the data systems it might serve, including:

- Multidimensional prediction.
- Archival access.
- Sensor noise.
- Rate control.
- Code rate improvements outside the module.
- Optimality of certain internal code options.

BACKGROUND, REVIEW AND ORIENTATION

The general form of a noiseless coding module is, from Refs. 3-7, given in Fig. 1. We will use it to reintroduce notation and focus on the goals of this report.

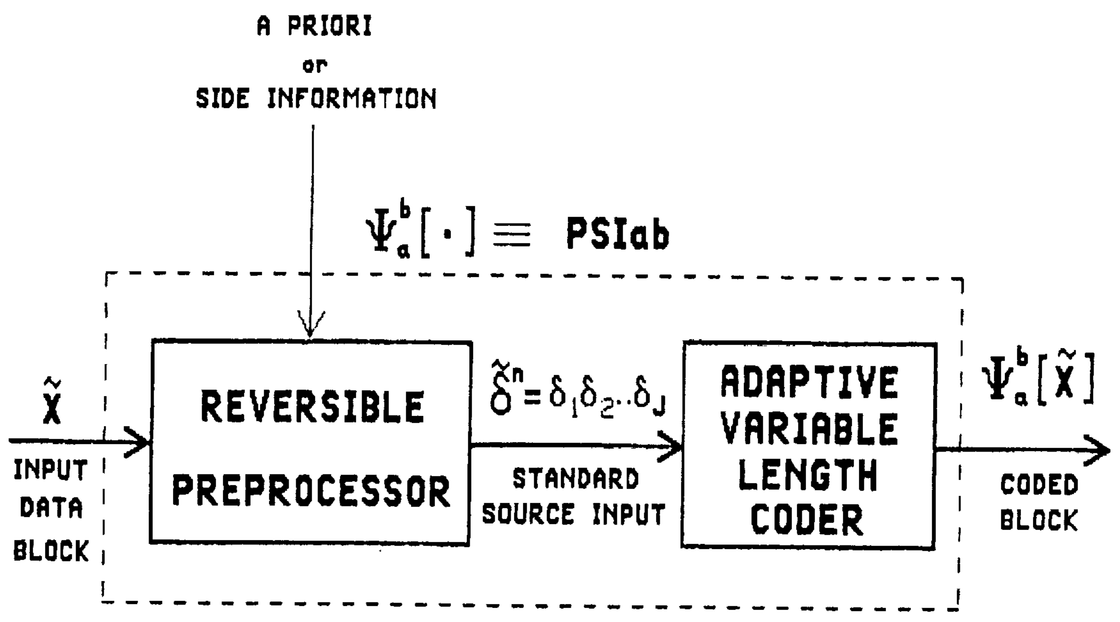


Fig. 1. General Coding Module

As shown, the coding process consists of two steps:

1) Reversible preprocessing of a block of data samples $\tilde{X}$ into another data block $\tilde{\delta}^n$ that has certain standard characteristics, and

2) Using adaptive variable-length coding to efficiently represent the standard source $\tilde{\delta}^n$ block produced by Step 1.

Ultimately we will provide the definitions and performance characteristics that convert this general-purpose coding module into PSI14,K+, which includes the subset described in Ref. 7.

Notation

In the figure, the overall coding process is specified by the "code operator"

$$\Psi_a^b[\cdot] \qquad (1)$$

That is, $\Psi_a^b[\cdot]$ operates on $\tilde{X}$ (and possibly a priori or side information) to produce the coded sequence $\Psi_a^b[\tilde{X}]$. This form of notation (subscripting and superscripting the Greek letter Ψ) was introduced in Ref. 3, and we will add to the list here. However, for space and convenience we will often call out a particular $\Psi_a^b[\cdot]$ by the English form PSIab. For example, $\Psi_{14,K}[\cdot]$ would become PSI14,K and

$$\Psi_{14,K}[\tilde{X}] \equiv \text{PSI14,K}[\tilde{X}]$$

The reversibility of these operators requires that: Given PSIab$[\tilde{X}]$ **and** any a priori or side information used in the coding process, the original data block $\tilde{X}$ can be recovered precisely.

Sequences vs. Samples. Where appropriate, we will emphasize that a quantity is a "sequence" of bits or samples by placing a tilde (~) over that quantity.

Concatenation. If $\tilde{A}$ and $\tilde{B}$ are two sequences of samples, then we can form a new sequence $\tilde{C}$ by running them back-to-back as

$$\tilde{C} = \tilde{A} * \tilde{B} \qquad (2)$$

and using the asterisk to indicate concatenation. However, the asterisk will be omitted occasionally when no confusion should result.

Length of a Sequence. The function $\mathcal{L}(\cdot)$ will be used to specify the length of a sequence. For example, if $\tilde{A}$ is a binary sequence, then

$$\mathcal{L}(\tilde{A}) \qquad (3)$$

denotes its length in bits. Without any anticipated confusion, if $\tilde{A}$ is nonbinary, we take $\mathscr{L}(\tilde{A})$ as either the length of $\tilde{A}$ in samples or the length of a standard fixed-length binary representation for $\tilde{A}$ in bits, whichever is more useful in context.

Estimated length. To indicate an "estimate" of the length of some sequence, we will use $\gamma(\cdot)$, appropriately subscripted and superscripted. For example, we have

$$\gamma_{ab}(\tilde{X}) \approx \mathscr{L}(\mathrm{PSIab}[\tilde{X}]) \tag{4}$$

as an estimate of the length of coded sequence PSIab[$\tilde{X}$].

Standard Source $\tilde{\delta}^n$

We can better understand the function of both preprocessor and coder by understanding the idealized Standard Source for the sequences

$$\tilde{\delta}^n = \delta_1 \, \delta_2 \ldots \delta_J \tag{5}$$

described as follows:

1) The J samples of $\tilde{\delta}^n$ are values from the set of the nonnegative integers

$$0, 1, 2, \ldots q\text{-}1 \tag{6}$$

2) The samples of $\tilde{\delta}^n$ have the probability distribution

$$P_\delta = \{p_0, p_1, \ldots p_{q-1}\} \tag{7}$$

3) The $\{p_i\}$ exhibit the ordering

$$p_0 \geq p_1 \geq p_2 \geq \ldots \geq p_{q-1} \tag{8}$$

as illustrated in Fig. 2, and

4) The samples of $\tilde{\delta}^n$ are independent (uncorrelated) with themselves and any "available" a priori or side information. (9)

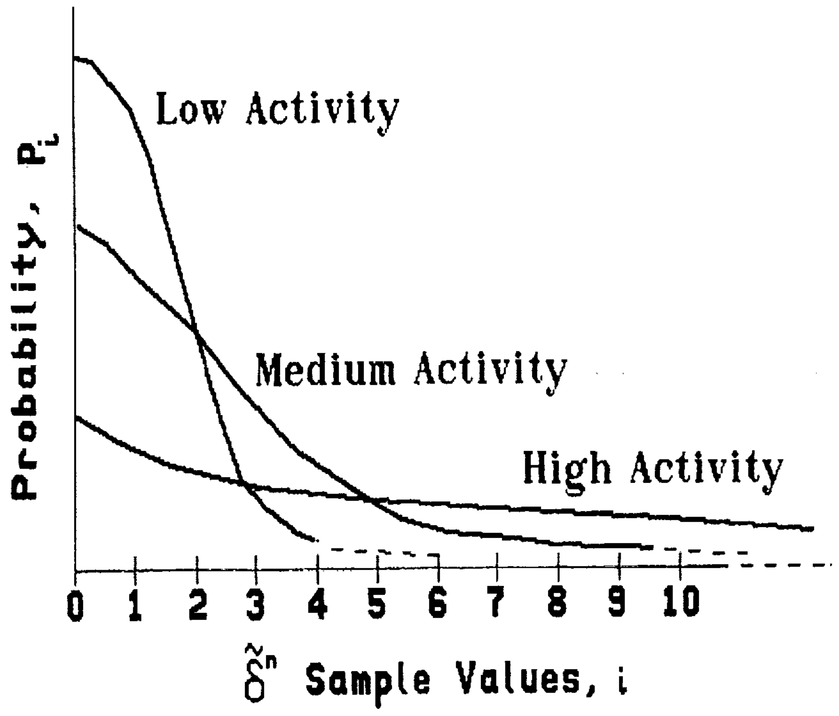


Fig. 2. $\tilde{\delta}^n$ Sample Probabilities

While idealized, these conditions can easily be well approximated for many practical problems. In any case, it is the preprocessor's task to achieve and maintain these conditions as closely as possible. Let's look at the consequence for the second step in Fig. 1 (coding).

By (6), the coder always has to deal with the same alphabet (with the exception of its size, q), regardless of the originating source.

When using **any** specific variable-length code, a maximum reduction in rate is obtained by using the shortest code words for the most frequently occurring symbols and the longer code words for the less likely symbols. Then by (8), the assignment of code words to a preprocessed $\tilde{\delta}^n$ sequence should always be to assign **the shortest code words to the smallest integers**. Under condition (8), this is the best assignment for any P_δ (meeting (8)) and any variable-length code. However, this does not say anything about whether a particular code is the best one to use.

The condition in (9) means that the burden of making the most from data correlation and a priori knowledge is placed on the preprocessor. If correlation still exists in $\tilde{\delta}^n$, then the preprocessor can probably be improved, which yields $\tilde{\delta}^n$ distributions with more frequent occurrence of the smaller integers.

Coder Performance

The entropy of a particular distribution P_δ is given by

$$H(P_\delta) = H_\delta = -\sum_i p_j \log_2 p_j \text{ bits/sample} \qquad (10)$$

If P_δ is unchanging, then H_δ is a bound to the best performance of any coding algorithm which follows. Assuming that preprocessing condition (9) has been met, H_δ is also a bound to overall performance of a coding module. For a given P_δ, the best single variable-length code can be derived from the Huffman algorithm [10]. Unfortunately, the real world hardly ever provides P_δ which do not change.

In practice, the idealized preprocessing conditions in (6)–(9) can usually be well approximated and maintained, but they change over time (see for example, Fig. 2). Consequently, the real practical problem facing the coder of preprocessed $\tilde{\delta}^n$ is to maintain efficient performance as P_δ changes. That is, usually the full burden of adapting to nonstationary data characteristics can be placed on the coder.

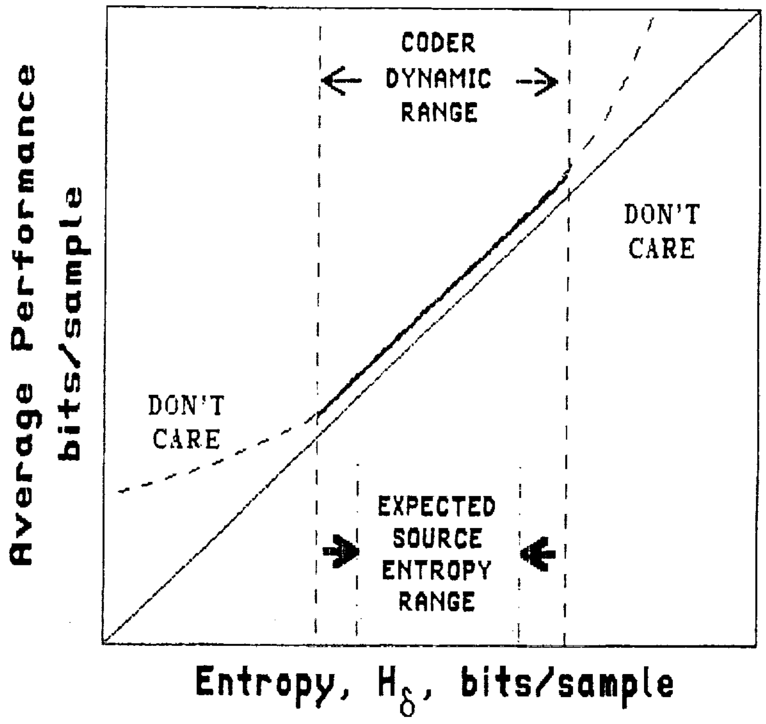
We say that that a coder is **efficient if it obtains performance "close to" an average measured entropy**, $\bar{H}_\delta$, which will vary as P_δ does. (11)

To bound performance in the classic sense, $\bar{H}_\delta$ must be measured over a span that is both long enough to be statistically significant and short enough to catch the real statistical variations. The real world is full of compromises, so $\bar{H}_\delta$ should generally be viewed as a guide to good performance rather than a bound.

The term "close to" leaves some room for interpretation, which usually means within 0.2 to 0.3 bit/sample. Thus, one efficient coder might be "more" efficient than another under certain conditions. At very low entropies, which are not of concern to us here, being within 0.2 bit/sample of $\bar{H}_\delta$ could not be considered efficient.

Dynamic Range. We will often be interested in the range of entropies over which a particular coder can be viewed as efficient in the sense just described. This is called a coder's Dynamic Range (of efficient performance).

Application Specific Goals. The performance goal for the variable-length coder is summarized graphically in Fig. 3.



CODER DYNAMIC RANGE
DON'T CARE
DON'T CARE
EXPECTED SOURCE ENTROPY RANGE
Average Performance bits/sample
Entropy, H_δ, bits/sample

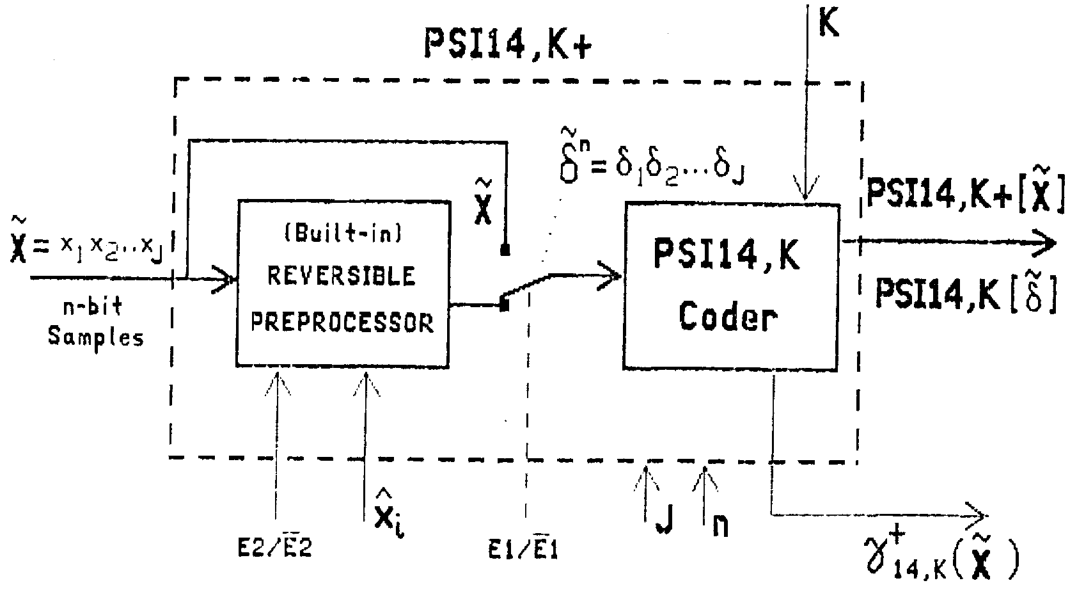
Fig. 3. Desired Average Coder Performance for a Specific Application

Graphically, the figure says that a coder's Dynamic Range should fall within the application's expected range of entropies.

The "Don't Care" regions simply indicate that entropies outside the application's range are unexpected, and so concern for good performance in these areas is not critical. However, real-world problems often produce transient situations which don't fit the norm. It is a good idea to build in some additional robustness (larger Dynamic Range) to deal with such transients.

GETTING MORE SPECIFIC

Figure 4 replaces the general coding module of Fig. 1 with the next level of detail. We can begin to unveil module PSI14,K+.



PSI14,K+
K
$\tilde{X} = x_1 x_2 .. x_J$
n-bit Samples
(Built-in) REVERSIBLE PREPROCESSOR
$\tilde{X}$
$\tilde{\delta}^n = \delta_1 \delta_2 \ldots \delta_J$
PSI14,K Coder
PSI14,K+[$\tilde{X}$]
PSI14,K [$\tilde{\delta}$]
E2/$\bar{E}$2
$\hat{x}_i$
E1/$\bar{E}$1
J
n
$\gamma^+_{14,K}(\tilde{X})$

Fig. 4. Module PSI14,K+ High-Level Functional Block Diagram

A specific predictive preprocessor that probably offers the broadest applicability to real problems will be discussed in Chapter II. The specific adaptive variable-length coder PSI14,K will be treated in Chapter III. But certain parameters and desirable features can best be introduced here without the additional detail.

Input

The input $\tilde{X}$ on the left of Fig. 4 is a J-sample data block containing n-bit samples. The built-in preprocessor will convert $\tilde{X}$ into the standard form $\tilde{\delta}^n$, which is also a J-sample sequence of n-bit samples. As already noted, $\tilde{\delta}^n$ generated in this way will satisfy or closely approximate the desired standard preprocessing conditions [6].

To Preprocess or Not To Preprocess

For those situations where the use of an external preprocessor is desirable, data can be entered directly into the coder, as shown. This option is functionally indicated in Fig. 4 by a logic signal "E1" controlling a switch. The inclusion of this feature considerably broadens the module's applicability.

Parameter Ranges

A discussion of parameter K, shown as externally controlling the coder PSI14,K in Fig. 4 will be provided in Chapter III. It suffices to note here that incrementing K by 1 will shift the Dynamic Range upward by 1 bit/sample. The number of K options included in a particular implementation depends on the goals and constraints of that implementation. Additional means for externally shifting the Dynamic Range will be discussed in Chapters III and IV.

Input Bits/Sample, n. There are numerous examples of real applications which could make use of the algorithms embodied in the PSI14,K+ coding module. The number of bits of data quantization for these examples lies in the range of

$$2 \leq n \leq 16 \tag{12}$$

with current project driven interest on n = 8, 12 and 14.

Block Size, J. Similarly, there are good reasons to consider block sizes over the range

$$1 \leq J \leq 16 \tag{13}$$

although the vast majority of applications could do just fine with a fixed J = 16.

Estimate. It is valuable to have a count of either the actual number of bits to code an individual data block or an estimate, as in (4). Thus, Fig. 4 includes

$$\overset{+}{\gamma}_{14,K}(\tilde{X})$$

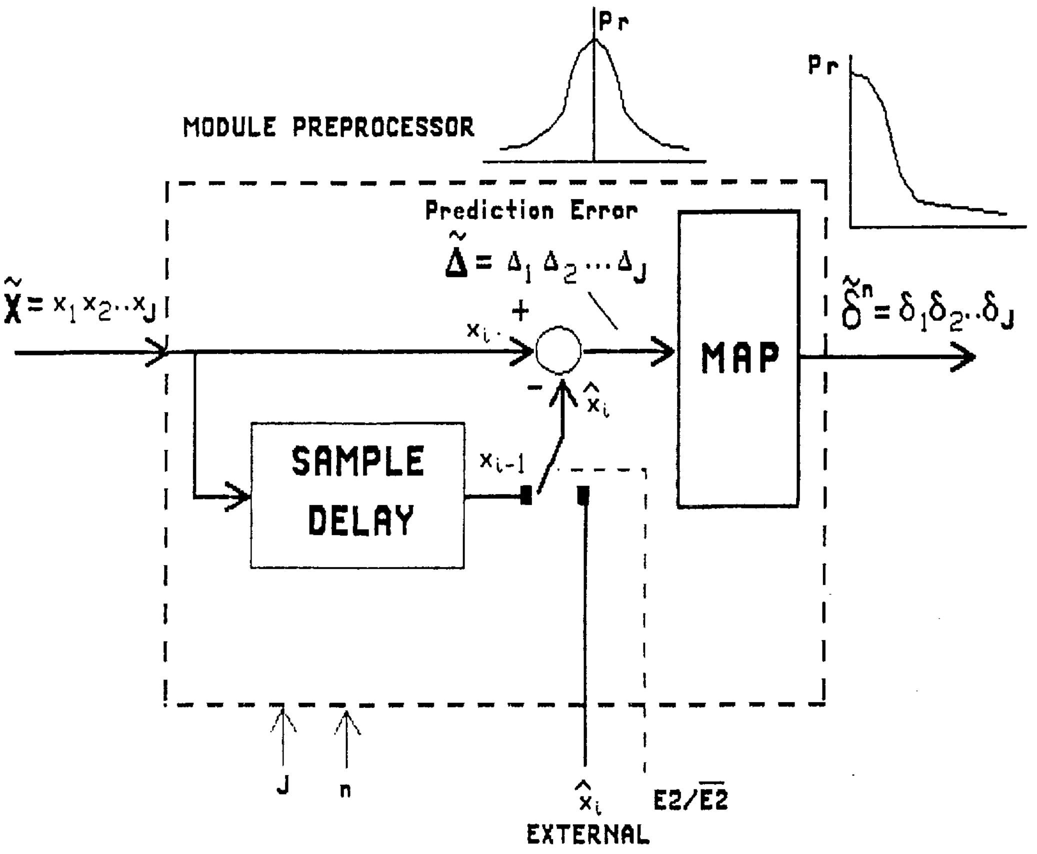
as a desirable output of the PSI14,K+ module. Such estimates can be used to guide decisions that are external to the module.

Other. Logic signal E2 and data signal $\hat{x}_i$ are shown for completeness. They are discussed in the next chapter.

PAGE ______ INTENTIONALLY BLANK

II. BUILT-IN PREPROCESSOR

A functional block diagram of the built-in PSI14,K+ Module preprocessor is shown in Fig. 5. It is derived directly from the imaging preprocessor in Ref. 5.



MODULE PREPROCESSOR
Pr
Pr
Prediction Error
$\tilde{\Delta} = \Delta_1 \Delta_2 \cdots \Delta_J$
$\tilde{X} = x_1 x_2 .. x_J$
x_i
+
−
$\hat{x}_i$
MAP
$\tilde{\delta}^n = \delta_1 \delta_2 .. \delta_J$
SAMPLE DELAY
x_{i-1}
J
n
$\hat{x}_i$
E2/$\overline{E2}$
EXTERNAL

Fig. 5 Built-in Preprocessor Functional Block Diagram

Predictor

The first part of this preprocessor is a very simple predictor consisting of a single sample delay element. With x_i as the ith sample in an input block $\tilde{X}$, this delay element "predicts" that x_i equals the previous sample:

$$\hat{x}_i = x_{i-1} \tag{14}$$

Such a predictor is broadly used as a one-dimensional (1-D) predictor. For many problems, no other predictor need be considered. However, the switch

PRECEDING PAGE BLANK NOT FILMED

controlled by logic signal "E2" provides a means for using an arbitrary external prediction, when desirable or necessary. The switch also serves the dual purpose of providing the first prediction of a data block when the internal sample delay has not been initialized (e.g., for the first sample of an image line).

An example of an external two-dimensional (2-D) predictor interfacing with the module's preprocessor is illustrated in Fig. 6. The current sample shown is the mth sample of the ith line of a raster image – $x_{i,m}$. An external predictor predicts that the value of $x_{i,m}$ will lie halfway between the sample above $x_{i-1,m}$ and the previous sample in the same line, $x_{i,m-1}$.

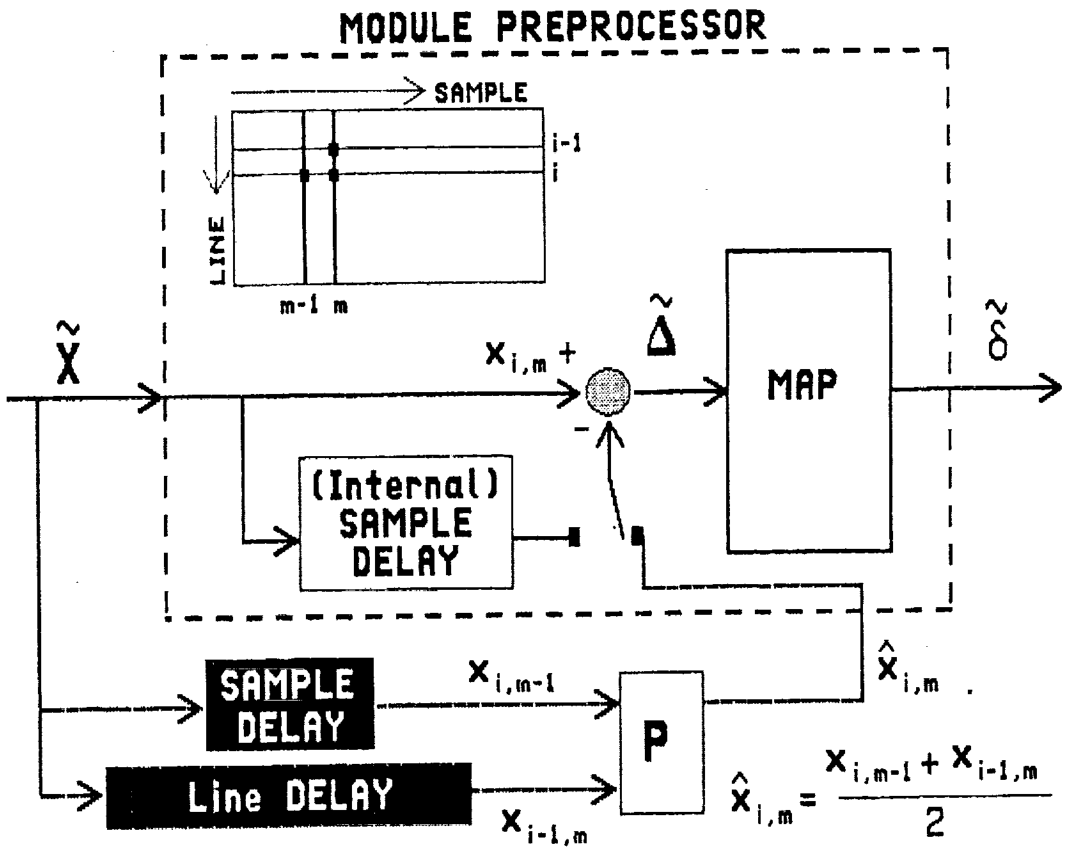


Fig. 6. External 2-D Predictor Interfacing With Module Preprocessor

Error Signal

The difference between any sample and its prediction produces the error signal

$$\Delta_i = x_i - \hat{x}_i \tag{15}$$

Sequences of Δ_i tend to display the unimodal distribution in Fig. 5 so that the condition

$$\Pr[\Delta_i = 0] \geq \Pr[\Delta_i = -1] \geq \Pr[\Delta_i = +1] \geq \Pr[\Delta_i = -2] \geq \ldots \tag{16}$$

is consistently well approximated.[1]

By using all available a priori and side information, the best predictor would produce an uncorrelated sequence of Δ_i with generally the smallest errors (so that the error distribution is more peaked around zero). Ultimately this would produce the lowest code rate. The module's external predictor option allows one to develop and use such a predictor if desired. However, the simple built-in delay predictor may come very close for many problems.

A rule of thumb for imaging data is that a 2-D predictor, such as in Fig. 6, may provide as much as 0.5 bit/sample net improvement in code rate over the simple delay predictor. In general, the benefits of improved code rate must be weighed against the implications of added complexity for each individual compression system implementation.

[1] Note that reversing the position of positive and negative differences in (16) is just as good an assumption. From a hardware-implementation point of view, there is a slight advantage to the arrangement in (16) [8]. Except where noted otherwise, the ordering of differences as in (16) will be assumed.

Mapping into the Integers

The final preprocessor step is to map the prediction errors, $\{\Delta_i\}$, into the non-negative integers so that the probability-ordering condition in (8) is well approximated. This is accomplished for the conditions in (16) by the basic mapping operation in Table 1 and Eq. 17.

Table 1. Basic Mapping of Δ_i into the Integers, δ_i

Prediction Error, Δ_i	Integer δ_i
0	0
-1	1
+1	2
-2	3
+2	4
-3	5
.	.
.	.
.	.

$$\delta_i = \begin{cases} 2|\Delta_i| - 1 & \text{if } \Delta_i < 0 \\ 2\Delta_i & \text{if } \Delta_i \geq 0 \end{cases} \tag{17}$$

But condition (16) cannot be true for all Δ_i when the signal values are close to the boundaries of the signal dynamic range. For example, if $\hat{x}_i = x_{i-1} = 0$, the error $\Delta_i = x_i - 0 = x_i \geq 0$, so that negative errors cannot occur. Then for the mapping in (17)

$$\Pr[\delta_i = \text{odd number} > 0] = 0$$

and so condition (8) can't be true either. Reference 5 provides an alternative mapping that takes advantage of these signal dynamic range constraints to avoid this problem. This mapping can be rewritten here for the ordering in (16) as

$$\mathcal{M}^{-+}(x_i,\hat{x}_i) = \delta_i = \begin{cases} 2\Delta_i & 0 \le \Delta_i \le \theta \\ 2|\Delta_i| - 1 & -\theta \le \Delta_i < 0 \\ \theta + |\Delta_i| & \text{Otherwise} \end{cases} \tag{18}$$

where for n-bit quantized samples

$$\theta(\hat{x}_i) = \min(\hat{x}_i, 2^n - 1 - \hat{x}_i) \tag{19}$$

For our example where $\hat{x}_i = 0$, if a $\Delta_i = +6$ occurred, Eq. 18 would produce a $\delta_i = 6$, whereas Eq. 17 would produce a $\delta_i = 2(6) = 12$. Appendix A provides additional information on the alternative mapping functions.

Performance Advantage. The performance benefit of using (18) and (19) instead of (17) is application dependent. Several tests on 8-bit imaging data showed typical improvements from 0.01 to 0.03 bit/sample.

Quantization Advantage. But this mapping has an additional advantage. Whereas n-bit data will produce n+1 bit prediction errors, δ_i is constrained by (18) and (19) to only n bits. That is

$$0 \le \delta_i \le 2^n - 1 \tag{20}$$

Data Line Advantage. Yet another advantage appears. Note that if an external prediction value $\hat{x}_i$ in Fig. 5 is fixed at zero, that is

$$\hat{x}_i = 0 \qquad \text{for all } i$$

and so

$$\Delta_i = x_i$$

Then by (18) and (19)

$$\theta = 0$$

$$\delta_i = |\Delta_i| = x_i$$

That is, referring again to Fig. 5, any input x_i is passed directly through the preprocessor, unchanged. This accomplishes the function of the Fig. 4 data line controlled by logic signal $E1/\overline{E1}$. The data line can be omitted.

III. ADAPTIVE VARIABLE-LENGTH CODING

The general-purpose adaptive coder was designated as PSI11 in Ref. 3. Its functional form is given in Fig. 7.

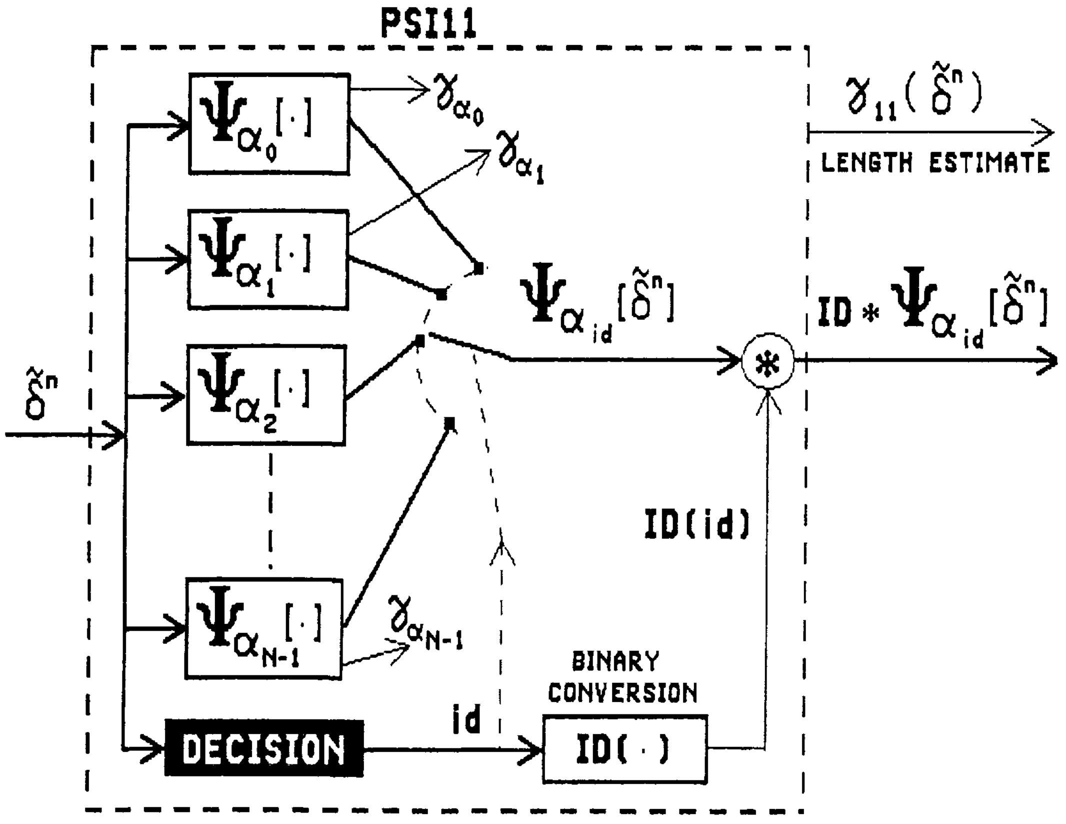


Fig. 7. General-Purpose Adaptive Coder, PSI11

The input to PSI11 is a J-sample preprocessed data block $\tilde{\delta}^n$ having the desired characteristics described in (5)–(9).

The output of PSI11 takes the form

$$\Psi_{11}[\tilde{\delta}^n] = ID(id) * \Psi_{\alpha_{id}}[\tilde{\delta}^n] \tag{21}$$

where id takes on the nonnegative integer values

$$0 \leq id \leq N-1 \tag{22}$$

and

$$ID(id) \tag{23}$$

is a binary string designation for id.

N is the number of code options available for coding data block $\tilde{\delta}^n$. The (subscript and/or superscript) designation for the ith coder is α_{i-1}. For example, if a coder named "PSI1,5" is the third coder in a list of coder options, then $\alpha_2 \equiv 1,5$.

Decision operations, discussed later, determine which coder is best to use and designate this choice with decision or "identifier" number, id. This directs the PSI11 adaptive coder to output the coded sequence

$$\Psi_{\alpha_{id}}[\tilde{\delta}^n] \tag{24}$$

prefaced with a binary code for the identifier, ID(id), to tell a decoder which type of coded sequence follows.

Unless noted otherwise, we will assume that the identifier code, ID, is an m-bit fixed-length code with[2]

$$m = \lfloor \log_2 N \rfloor \tag{25}$$

Typically N is chosen so that $N = 2^m$.

Note that we have modified the notation slightly from Refs. 3–5 to clarify the distinctions among a coder's decision, its binary representation, and the corresponding coder option designations.

[2]$\lfloor x \rfloor$ is the smallest integer greater than or equal to x.

Also shown in the figure is an estimate of the coded length of any internal code option i. By (4)

$$\gamma_{\alpha_i}(\tilde{\delta}^n) \approx \mathscr{L}(\Psi_{\alpha_i}[\tilde{\delta}^n])$$

and therefore, an estimate for PSI11 itself is

$$\mathscr{L}(\Psi_{11}[\tilde{\delta}^n]) \approx \gamma_{11}(\tilde{\delta}^n) = \mathscr{L}(\mathrm{ID(id)}) + \gamma_{\alpha_{id}}(\tilde{\delta}^n) \tag{26}$$

and where typically $\mathscr{L}(\mathrm{ID(id)}) = m$ from (25).

PSI14

We now define the individual coder options that convert the general form PSI11 into PSI14.

Fundamental Sequence, PSI1

Define the code word fs[i] by

$$\mathrm{fs}[i] = \underbrace{0\,0\,0\,\ldots\,.\,0\,0\,0}_{i \text{ zeroes}}\,1 \tag{27}$$

where $i \geq 0$ is an input integer. The length of "code word" fs[i] is

$$\ell_i = \mathscr{L}(\mathrm{fs}[i]) = i + 1 \text{ bits} \tag{28}$$

Again, let

$$\tilde{\delta}^n = \delta_1\,\delta_2\,.\,.\,.\,\delta_J$$

be an input sequence of samples meeting the preprocessing conditions described earlier. Then, the coding of $\tilde{\delta}^n$ by using fs[·] on each sample yields the "Fundamental Sequence" of $\tilde{\delta}^n$

$$\mathrm{PSI1}[\tilde{\delta}^n] = \Psi_1[\tilde{\delta}^n] = fs[\delta_1] * fs[\delta_2] * \ldots fs[\delta_J] \tag{29}$$

That is, PSI1 denotes the application of the "fs" code in (27) to all the samples of a sequence. The length of a Fundamental Sequence is

$$F_0 = \gamma_1 = \mathscr{L}(\mathrm{PSI1}[\tilde{\delta}^n]) = J + \sum_{j=1}^{J} \delta_j \tag{30}$$

Note that the code defined in (27) is probably the simplest nontrivial variable-length code there is. It is defined for all input alphabet sizes by the simple expression in (27). This simplicity carries into both software and hardware implementations.[3]

Example. Let

$$\begin{aligned}\tilde{\delta}^n &= \delta_1\ \delta_2 \ldots \delta_{14}\\ &= 0\,0\,0\,0\,0\,4\,0\,0\,4\,9\,0\,0\,1\,0\end{aligned} \tag{31}$$

By applying the rules in (27) and (29), we get

$$\Psi_1[\tilde{\delta}^n] = 1\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,1\,1 \tag{32}$$

and from (30), or by counting,

$$F_0 = 14 + (4 + 4 + 9 + 1) = 32 \text{ bits} \tag{33}$$

---

[3]Clearly, if q = max i, then (27) and (28) can be replaced by

$$fs[q] = \underbrace{0\,0\,0\ldots0\,0\,0}_{q \text{ zeroes}} \text{ and } \ell_q = q \text{ bits}$$

We refer to this refinement in the fs code definition as the "Optimized FS Code." However, except for very small alphabets, the added performance benefits can be expected to be insignificant [11]. Thus, except where noted otherwise, we will presume the simpler definition in (27) and (28).

Performance. The average performance for PSI1, based on measured results from numerous data sources, is plotted in Fig. 8 as a function of entropy H_δ (Eq. 10).

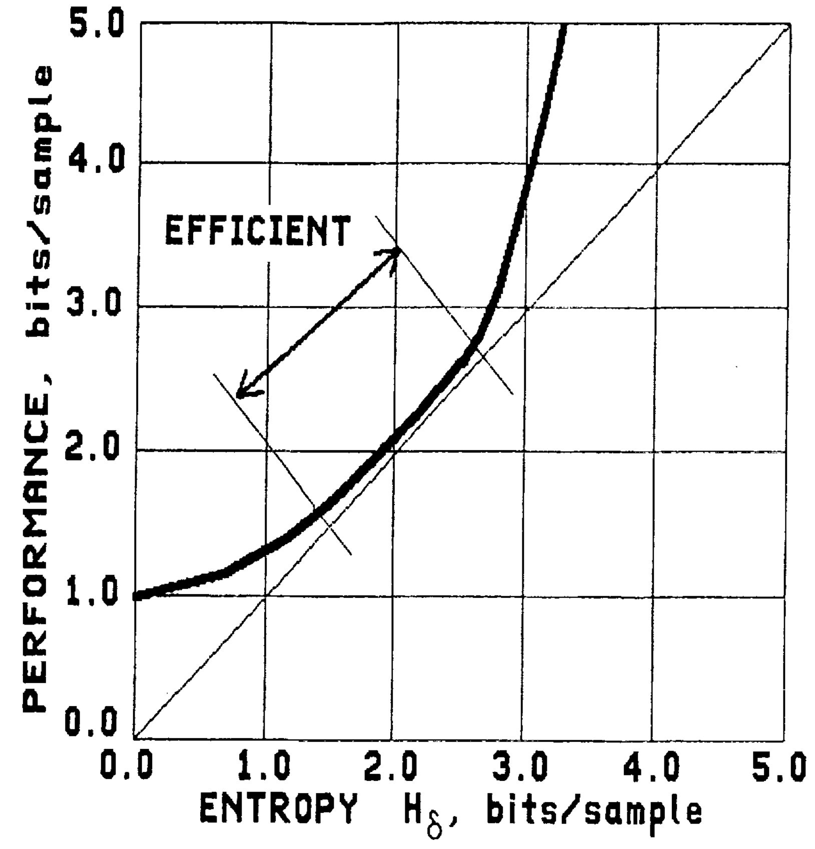


Fig. 8. Average PSI1 Performance

Performance close to the entropy should not be a surprise since the fs[·] code in (27) is clearly a Huffman code for some distribution. (The latter subject is later investigated in conjunction with other coding operations in Ref. 11). The **Dynamic Range** for this code is the entropy region of approximately $1.5 \leq H_\delta \leq 2.5$ bits/sample. Such a narrow range is typical for individual variable-length codes.

Code Operator PSI0

From (29) and (30),

$$\tilde{FS} = \Psi_1[\tilde{\delta}^n] = \zeta_1 \zeta_2 \cdots \zeta_{F_0} \tag{34}$$

denotes a Fundamental Sequence of length F_0 bits, where now ζ_i denotes the ith bit.

Complement. The complement of $\tilde{FS}$ is given as

$$\text{COMP}[\tilde{FS}] = \bar{\tilde{FS}} = \bar{\zeta}_1 \bar{\zeta}_2 \cdots \bar{\zeta}_{F_0} \tag{35}$$

where $\bar{\zeta}_i$ is the complement $(2^n - 1 - \zeta_i)$ of the ith sample of $\tilde{FS}$.

Third Extension. The third Extension of $\bar{\tilde{FS}}$ is given as

$$\text{Ext}^3[\bar{\tilde{FS}}] = (\bar{\zeta}_1 \bar{\zeta}_2 \bar{\zeta}_3) * (\bar{\zeta}_4 \bar{\zeta}_5 \bar{\zeta}_6) * \cdots (\bar{\zeta}_{F_0}\, 0\, 0) \tag{36}$$

where the samples of $\bar{\tilde{FS}}$ have been grouped into binary 3-tuples (the last 3-tuple is completed by adding dummy zeroes, as necessary). Thus, we have

$$a = \mathscr{L}(\text{Ext}^3[\bar{\tilde{FS}}]) = \left\lfloor \frac{F_0}{3} \right\rfloor \text{ 3-tuples} \tag{37}$$

and

$$b = \mathscr{L}(\text{Ext}^3[\bar{\tilde{FS}}]) = 3\left\lfloor \frac{F_0}{3} \right\rfloor = 3a \quad \text{bits} \tag{38}$$

We are interested in $\text{Ext}^3[\bar{\tilde{FS}}]$ as a source of binary 3-tuples, so we write

$$\text{Ext}^3[\bar{\tilde{FS}}] = \beta_1 * \beta_2 * \cdots \beta_a \tag{39}$$

where β_i are the individual 3-tuples that make up (36).

Coding $\mathrm{Ext}^3[\tilde{FS}]$. Code operator PSI0 is defined as the coding $\mathrm{Ext}^3[\tilde{FS}]$ in (39) by the variable-length code in Table 2.[4]

Table 2. 8-Word 3-Tuple Code, cfs[i]

3-tuple β_i	CODE WORD $cfs[\beta_i]$
000	1
001	001
010	010
100	011
011	00000
101	00001
110	00010
111	00011

Applying this code to the 3-tuples of $\mathrm{Ext}^3[\tilde{FS}]$ provides the result we are looking for

$$\Psi_0[\tilde{\delta}n] = cfs[\beta_1] * cfs[\beta_2] * \ldots cfs[\beta_a] \qquad (40)$$

Estimate. It can be shown that a bound and good estimate to $\mathscr{L}(\mathrm{PSI0}[\tilde{\delta}^n])$ is

$$\gamma_0(\tilde{\delta}^n) = \frac{F_0}{3} + 2(F_0 - J) \geq \mathscr{L}(\psi_0[\tilde{\delta}^n]) \qquad (41)$$

[4]This particular code is slightly different than listed in Refs. 1–4. It has the same code word lengths and thus results in the same performance. However, this arrangement offers a hardware implementation advantage [12].

Example. Let

$$\tilde{\delta}^n = \delta_1 \delta_2 \ldots \delta_{10} = 0\,0\,0\,1\,0\,0\,0\,2\,0\,0 \qquad (42)$$

Then, by using (33),

$$\tilde{FS} = 1\,1\,1\,0\,1\,1\,1\,1\,0\,0\,1\,1\,1 \qquad (43)$$

with length

$$F_0 = 13 \text{ bits} \qquad (44)$$

By complementing, we get

$$\bar{\tilde{FS}} = 0\,0\,0\,1\,0\,0\,0\,0\,1\,1\,0\,0\,0 \qquad (45)$$

and

$$\mathrm{Ext}^3[\bar{\tilde{FS}}] = (0\,0\,0)\,(1\,0\,0)\,(0\,0\,1)\,(1\,0\,0)\,(0\,0\,0) \qquad (46)$$

Then, by applying the cfs[·] code in Table 1, we get from (40)

$$\Psi_0[\tilde{\delta}^n] = 1\,0\,1\,1\,0\,0\,1\,0\,1\,1\,1 \qquad (47)$$

and

$$\mathcal{L}(\Psi_0[\tilde{\delta}^n]) = 11 \text{ bits} \qquad (48)$$

Observe that by using (41),

$$\gamma_0 = 5 + 2\,(13 - 10) = 11 = \mathcal{L}(\Psi_0[\tilde{\delta}^n]) \qquad (49)$$

Average Performance. The average measured performance for PSI0 is added to the Fig. 8 results in Fig. 9.

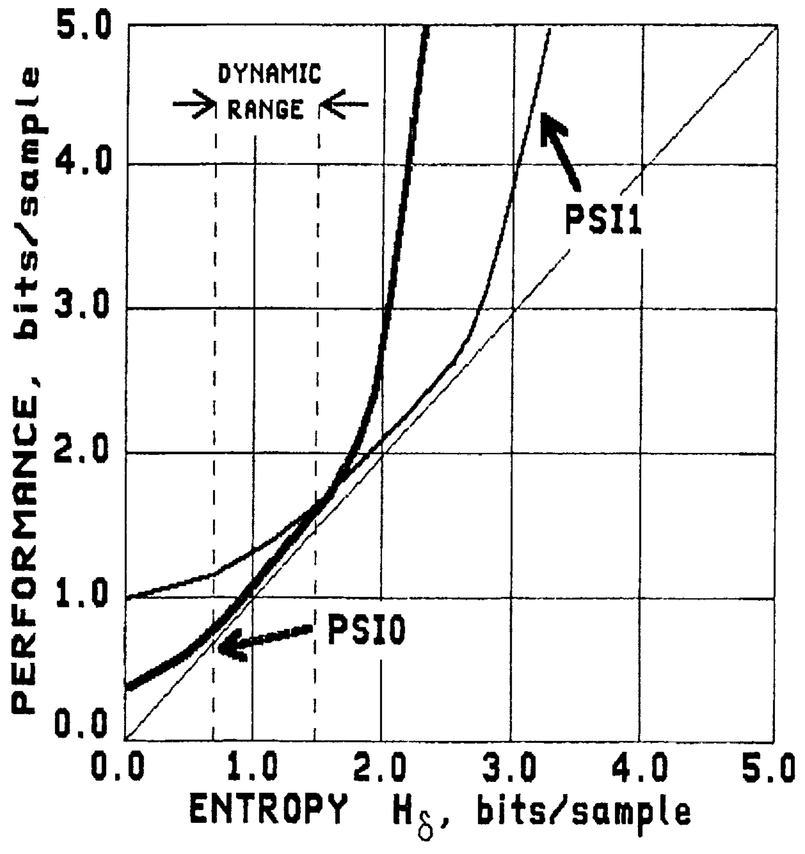


Fig. 9. Average PSI0 Performance

PSI0 performance becomes efficient at the lower entropies precisely where PSI1 performance begins to pull away from the entropy line. This should not be surprising; we are coding $\tilde{FS}$ = PSI1[$\tilde{\delta}$n] as a data source and using a Huffman (3-tuple) code, cfs[·]. When PSI1 starts becoming inefficient, $\tilde{FS}$ must have some redundancy left in it.

It is debatable how "efficiency" should be defined as entropies drop below 1 bit/sample. After all, at $\bar{H}_\delta$ = 1 bit/sample, 0.1 bit/sample represents a 10% error, whereas, at $\bar{H}_\delta$ = 5 bits/sample, 0.1 bit/sample represents only a 2% error. However, we will make the practical assumption that PSI0 average performance can be called efficient down to 0.75 bit/sample. We note also that PSI9 from Refs. 3 and 4 can significantly improve performance as entropies become very low.

Thus, we take the Dynamic Range for PSI0 to be $0.75 \le \bar{H}_\delta \le 1.5$ bits/sample.

The Unity Code Operator, PSI3

A trivial code operator is obtained by defining PSI3[$\tilde{\delta}^n$] as any fixed-length binary representation of $\tilde{\delta}^n$. In the simplest case, we can take PSI3[$\tilde{\delta}^n$] as $\tilde{\delta}^n$ itself so that

$$\psi_3[\tilde{\delta}^n] = \tilde{\delta}^n \tag{50}$$

Operators PSI2 and PSI4

We will only make brief note of these two code operators for historical purposes. They do not form a part of PSI14.

PSI2 is very similar to PSI0. PSI2[$\tilde{\delta}^n$] is obtained by directly using the code in Table 2 to code its 3-tuples [1]–[4]. That is, perform all the same operations as for PSI0 but don't complement the Fundamental Sequence. The Dynamic Range for PSI2 is $3 \leq \bar{H}_\delta \leq 4$ bits/sample.

PSI4, also called the "Basic Compressor," is an adaptive coder, as defined by PSI11 in Fig. 7 and (21), which uses the options PSI0, PSI1, PSI2 and PSI3. Its Dynamic Range is $0.75 \leq \bar{H}_\delta \leq 4.0$ bits/sample, as illustrated in Fig. 10.

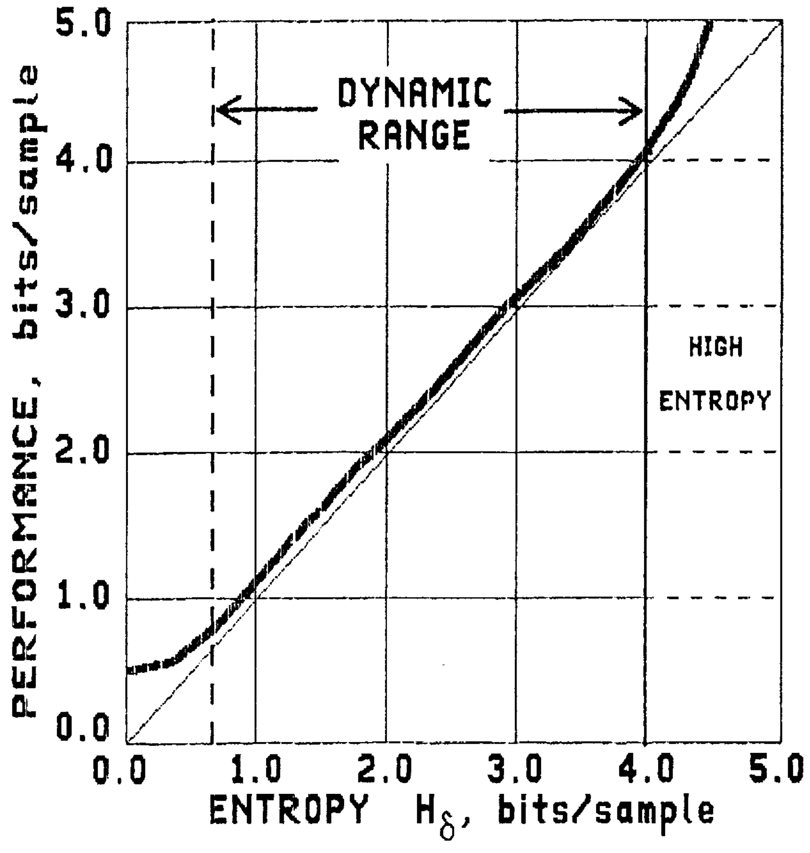


Fig. 10. Average Performance, PSI4

Split-Sample Modes

We can limit some of the generality provided in the notation of Ref. 5 since we are looking for a more specific result here.[5]

Again, let $\tilde{\delta}^n$ be a sequence of J preprocessed samples where now the "n" means that these samples are quantized to n bits. Define the basic "Split-Sample" Operator $SS^{n,k}$ by

$$SS^{n,k}[\tilde{\delta}^n] = \{\tilde{M}^{n,k}, \tilde{L}_k\} \tag{51}$$

[5]For additional generality, Ref. 5 used $SS_0^{n,k}[\cdot]$ for $SS^{n,k}[\cdot]$, $\tilde{L}_k^0$ for $\tilde{L}_k$, $\tilde{M}_0^{n,k}$ for $\tilde{M}^{n,k}$, etc.

where

$$\tilde{L}_k = S_A^{n,k}[\tilde{\delta}^n] = \left\{ \begin{array}{l} \text{J-sample sequence} \\ \text{of all the k least-} \\ \text{significant bits of} \\ \text{each } \tilde{\delta}^n \text{ sample} \end{array} \right. \tag{52}$$

and $\tilde{M}^{n,k}$ is simply the J-sample sequence of the n-k most-significant bits of each $\tilde{\delta}^n$ sample after removing the least-signficant k. That is

$$\tilde{M}^{n,k} = S_B^{n,k}[\tilde{\delta}^n] = \left\{ \begin{array}{l} \text{J-sample sequence} \\ \text{of all the n-k most-} \\ \text{significant bits of} \\ \text{each sample of } \tilde{\delta}^n \end{array} \right. \tag{53}$$

and where we note that

$$SS^{n,0}[\tilde{\delta}^n] = \tilde{M}^{n,0} = \tilde{\delta}^n \text{ and } SS^{n,n}[\tilde{\delta}^n] = \tilde{L}_n = \tilde{\delta}^n$$

so that

$$0 \leq k \leq n \tag{54}$$

These operations are illustrated in Fig. 11. For future simplicity, we will call $SS^{n,k}$ simply "SPLIT."

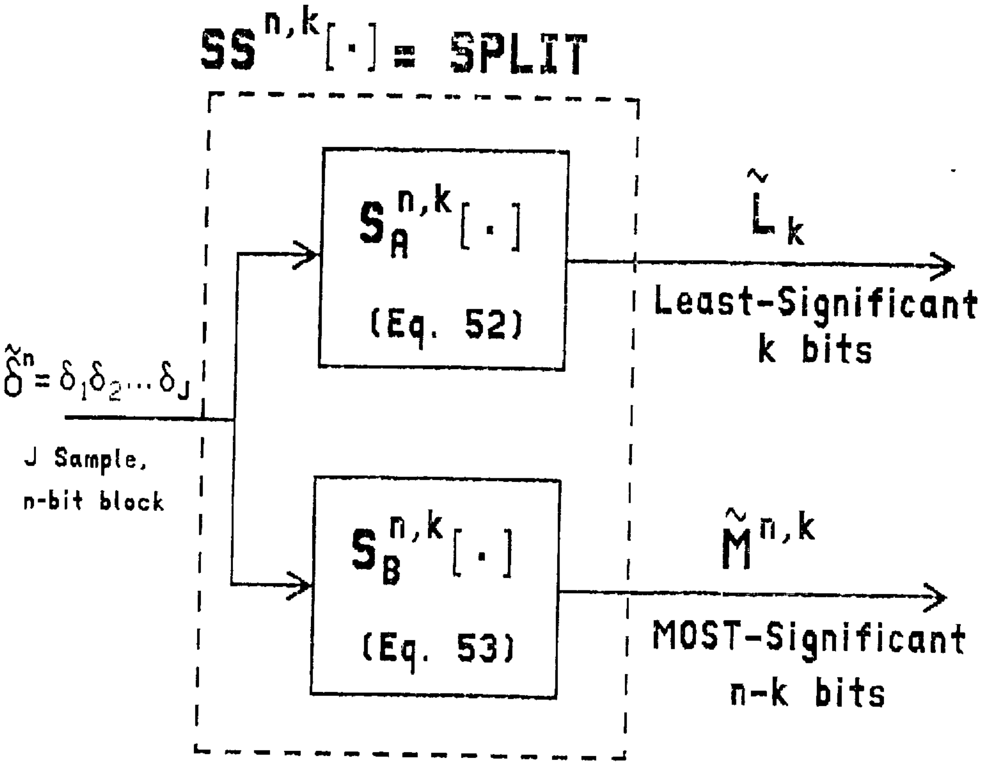


Fig. 11. Basic Split-Sample Operator

Motivation. Let $H(\tilde{\delta}^n)$, $H(\tilde{M}^{n,k})$ and $H(\tilde{L}_k)$ denote the average entropies associated with $\tilde{\delta}^n$, $\tilde{M}^{n,k}$ and $\tilde{L}_k$ sequences, respectively. It has been observed that

$$H(\tilde{M}^{n,k}) = H(\tilde{\delta}^n) - k \tag{55}$$

and

$$H(\tilde{L}_k) \approx k \tag{56}$$

provided that (approximately) $H(\tilde{M}^{n,k}) > 3$ bits/sample.

Equation 56 says that the least-significant bits are totally random, and Eq. 55 says that removing k least-significant bits drops the entropy of what remains by k bits/sample. More important, Eqs. 55 and 56 imply

$$H(\tilde{M}^{n,k}) + H(\tilde{L}_k) \approx H(\tilde{\delta}^n) \tag{57}$$

Thus, if both $\tilde{M}^{n,k}$ and $\tilde{L}_k$ sequences can be separately coded efficiently (close to the entropy), we will succeed in coding the original sequence $\tilde{\delta}^n$ efficiently also.

Coding $\tilde{L}_k$ sequences efficiently is trivial; since they are random, their uncoded form is already an efficient representation. For the $\tilde{M}^{n,k}$ sequences, we note that they continue to retain the desired probability ordering of (8) as k is increased (remember that $\tilde{\delta}^n$ is preprocessed by assumption). Thus, at some value of k, the entropy of $\tilde{M}^{n,k}$ sequences will drop low enough to lie within the Dynamic Range of code operators such as PSI0, PSI1, PSI4, etc.

Operator PSIi,k. Split-Sample "code" operator PSIi,k is defined by[6]

$$\psi_{i,k}[\tilde{\delta}^n] = \psi_i[\tilde{M}^{n,k}] * \tilde{L}_k \tag{58}$$

The structure of this operator is illustrated in Fig. 12.

Example for PSI1,k. The following should give a practical feel for the Split-Sample Mode concept.

Let

$$\tilde{\delta}^5 = 10, 4, 3, 7, 5, 0, 2 \tag{59}$$

[6]The order of PSIi[$\tilde{M}^{n,k}$] and $\tilde{L}_k$ in (58) is reversed from the definition in Ref. 5. Statistically the order doesn't matter, but the arrangement in (58) appears to hold a hardware implementation advantage [8]–[9]. It also provides a format consistent with Split-Sample modes generated before preprocessing, as discussed later, where there are additional advantages to this ordering.

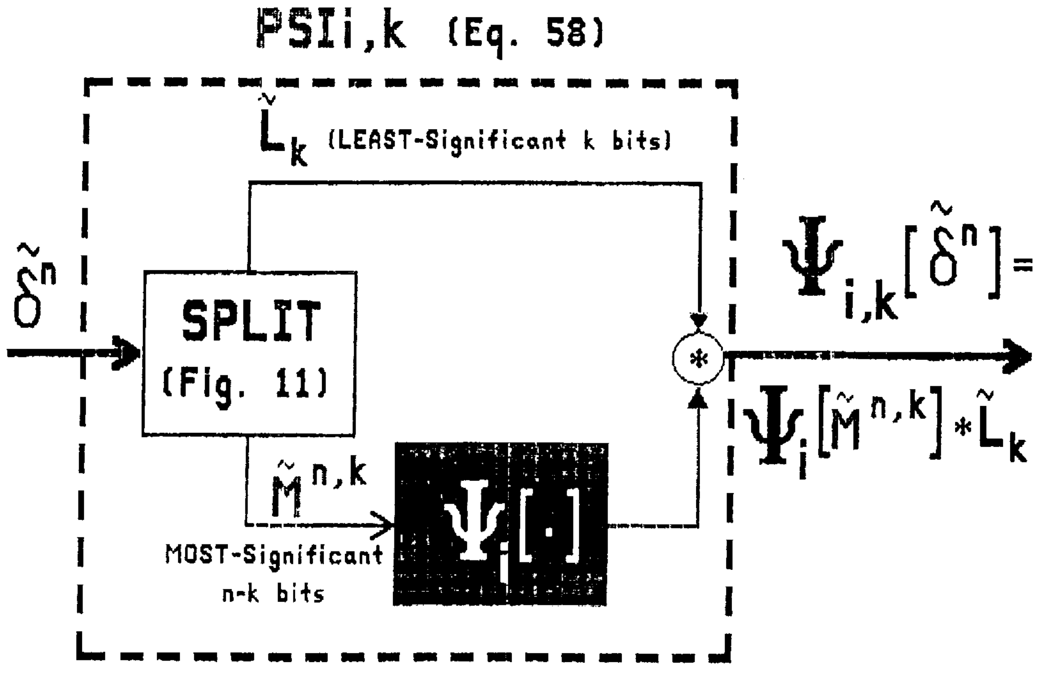

PSIi,k (Eq. 58)
$\tilde{L}_k$ (LEAST-Significant k bits)
$\tilde{\delta}^n$
SPLIT
(Fig. 11)
$\tilde{M}^{n,k}$
MOST-Significant n-k bits
$\Psi_i[\cdot]$
$*$
$\Psi_{i,k}[\tilde{\delta}^n] =$
$\Psi_i[\tilde{M}^{n,k}] * \tilde{L}_k$

Fig. 12. Split-Sample Coder, PSIi,k

be a J = 7 sample sequence of 5-bit samples. In binary, $\tilde{\delta}^5$ becomes

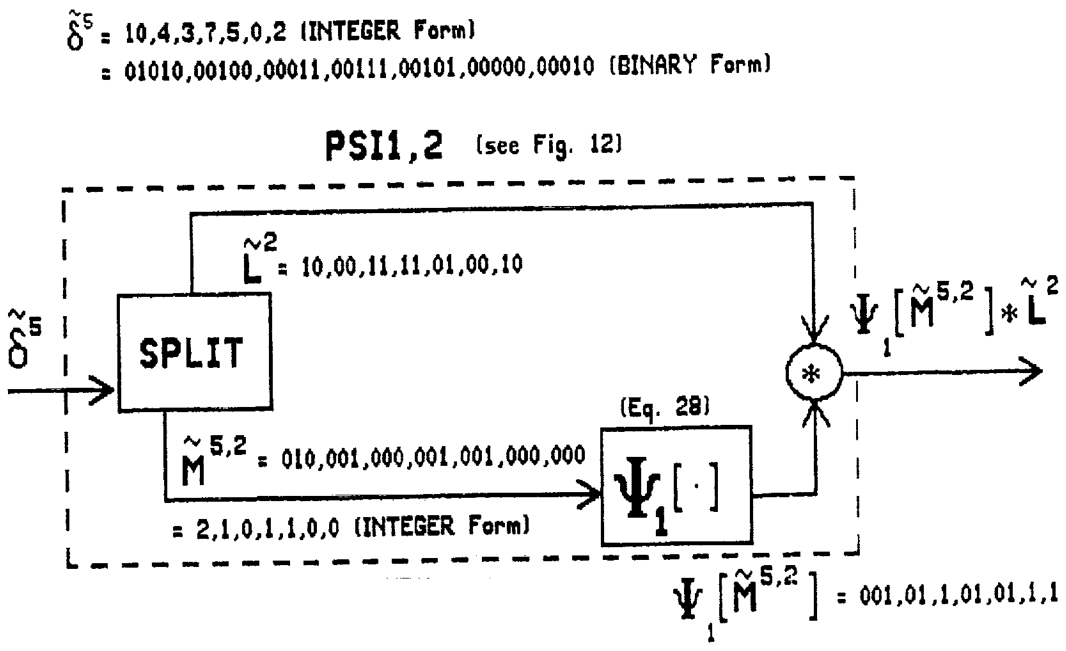
$$\tilde{\delta}^5 = 010\boxed{10}, 001\boxed{00}, 000\boxed{11}, 001\boxed{11},$$
$$001\boxed{01}, 000\boxed{00}, 000\boxed{10} \qquad (60)$$

where for future reference we have placed boxes around the two least-significant bits of each sample. Now let's try coding with PSI1,2. A block diagram showing the steps is provided in Fig. 13.

From the figure we have

$$\mathscr{L}(\text{PSI1,2}[\tilde{\delta}^5]) = 26 \text{ bits} \qquad (61)$$

whereas a direct coding of $\tilde{\delta}^5$ using PSI1 would require



$\tilde{\delta}^5$ = 10,4,3,7,5,0,2 (INTEGER Form)
= 01010,00100,00011,00111,00101,00000,00010 (BINARY Form)
PSI1,2 (see Fig. 12)
$\tilde{L}^2$ = 10,00,11,11,01,00,10
$\tilde{\delta}^5$
SPLIT
$\tilde{M}^{5,2}$ = 010,001,000,001,001,000,000
= 2,1,0,1,1,0,0 (INTEGER Form)
(Eq. 28)
$\Psi_1[\cdot]$
$\Psi_1[\tilde{M}^{5,2}] * \tilde{L}^2$
$\Psi_1[\tilde{M}^{5,2}]$ = 001,01,1,01,01,1,1

Fig. 13. PSI1,2 Example

$$\mathcal{L}(\text{PSI1}[\tilde{\delta}5]) = \sum_j \delta j + 7 = 39 \text{ bits} \qquad (62)$$

Performance for PSI4,k. Again for historical purposes, the performance for Split-Sample modes using PSI4 (as the coder of most-significant bits) is illustrated in Fig. 14 [1]–[4].

The figure shows that each increase in k produces a code option PSI4,k whose individual Dynamic Range has been shifted upward by 1 bit/sample. Further, there is a PSI4,k Split-Sample option to fit each 1-bit entropy range above 4 bits/sample. The breakpoints occur near integer entropy values.

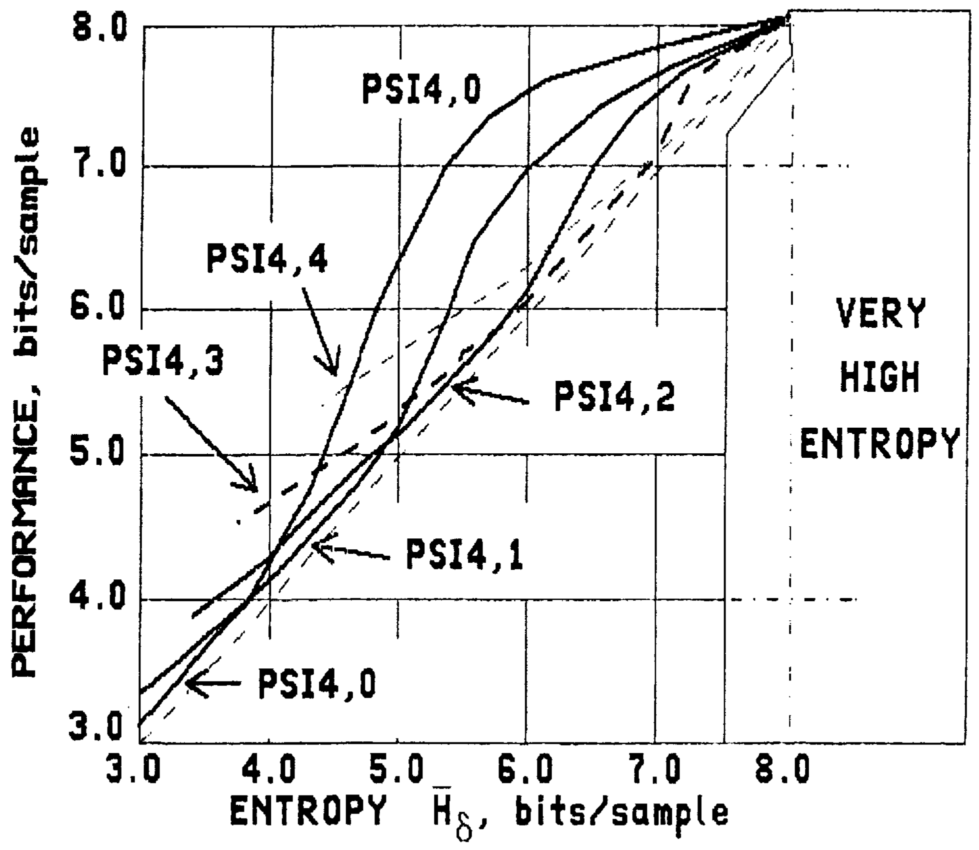


Fig. 14. Average Performance, PSI4,k

Performance for PSI1,k. It should be no surprise that the same type of curves result when only the simple code operator PSI1 is used to represent the most-significant bits. Of course, the performance curves start at lower entropies since the efficient operating range for PSI1 alone is between 1.5 and 2.5 bits/sample (versus about 0.75 to 4 bits/sample for PSI4).

We will not study the individual characteristics of each PSI1,k. Instead, the next section investigates the composite performance obtained by an adaptive coder which has many PSI1,k options from which to choose. This, in fact, is the basis for code operator PSI14. For detailed performance curves of the individual PSI1,k, consult Yeh [11]. In fact, **she has shown an equivalence between PSI1,k Split-Sample Modes and Huffman codes.** This is discussed further in Appendix B.

CODE OPERATOR PSI14

Definition

PSI14 is defined as a version of adaptive coder PSI11 in (19), which is repeated here as

$$\Psi_{14}[\tilde{\delta}^n] = ID(id) * \Psi_{\alpha_{id}}[\tilde{\delta}^n] \tag{63}$$

and which uses three or more "adjacent" code options from the list in (64) and code option PSI3 (or PSIF).[7]

$$\left\{ \begin{matrix} PSI0\,, & PSI1\,, & PSI1,1\,, & PSI1,2,\ldots \\ \lambda = 0 & \lambda = 1 & \lambda = 2 & \lambda = 3 \end{matrix} \right\} \tag{64}$$

Thus, a particular N-option PSI14 configuration is completely specified by identifying the first option in this list (from the left) to be included. And given this list, this first option can be specified parametrically by its position in the list

$$\lambda \geq 0 \tag{65}$$

as shown in (64). For example, $\lambda = 2$ means the first code option to be used is PSI1,1.

Starting Option. The specification of this starting option can be made quite compact if we note that both PSI0 and PSI1 are really limiting forms of Split-Sample modes PSIi,k with k = 0. That is,

$$PSI0 \equiv PSI0,0 \tag{66}$$

and

[7]PSIF is the "Fast Compressor" discussed in Ref. 5. In this application, PSIF would replace PSI3 as an adaptive "backup coder" (see Eq. 50). However, most typical applications would receive no statistical benefit.

$$\mathrm{PSI1} \equiv \mathrm{PSI1,0} \tag{67}$$

We can quite generally define a PSI14 Starting Option as

$$\Psi_{i(\lambda),k(\lambda)} \text{ for } 0 \le \lambda \le n+1 \tag{68}$$

where

$$i(\lambda) = \begin{cases} 0 & \text{if } \lambda = 0 \\ 1 & \text{Otherwise} \end{cases} \tag{69}$$

$$k(\lambda) = \begin{cases} 0 & \text{if } \lambda = 0 \\ \lambda - 1 & \text{Otherwise} \end{cases} \tag{70}$$

and n is the input bits/sample.

Numbering Each Option. We must assign an identifier number to each of the N codes used in PSI14. By definition, we take

$$id = N - 1 \text{ for PSI3} \tag{71}$$

Then we take

$$id = 0 \tag{72}$$

for the starting option at position λ in the list of (64). All other options (to the right of this starting option) are assigned increasing id values as the list is traversed to the right. Parametrically, this is very straightforward. The idth option is

$$\Psi_{i(\lambda + id),\, k(\lambda + id)} \tag{73}$$

for

$$0 \le \lambda \le n + 1 \text{ and } 0 \le id \le N - 2$$

Note that λ can have $n + 2$ possible values (0 through $n + 1$); this is the maximum number of coders available. Then

$$N \le (\text{max no. of coders}) - (\text{starting coder no.}) = (n + 2) - \lambda \qquad (74)$$

Looking closely at (54), (71) and (73), **we see that parameters N, λ, and n completely define any PSI14, including the identifier to use in (63).** We will later see that these parameters also define the corresponding PSI14 Dynamic Range.

Example. Let $N = 6$ and $\lambda = 3$. Then a complete PSI14 code specification can be obtained from (68)–(74), as shown in Table 3.

When $\lambda \ge 1$. If we exclude PSI0 from the possible options by restricting $\lambda \ge 1$, a considerable simplification in notation results since

$$i(\lambda) = 1 \text{ for all } \lambda$$

and

$$k(\lambda) = \lambda - 1$$

Then, a PSI14 coder with parameters N, λ and n has code options

$$\text{PSI1}, \lambda - 1 + id \quad \text{for } 0 \le id \le N - 2$$

and

$$\text{PSI3} \equiv \text{PSI1},n \quad \text{for } id = N - 1$$

where id is the identifier associated with each option. Under these restrictions, PSI14 reduces to PSIss, as discussed in Ref. 7. Further restricting λ to be fixed at $\lambda = 1$ yields the basis of VLSI implementations [8]–[9].

Table 3. PSI14 Specification Example
for $N = 6, \lambda = 3, n \geq 7$

Code Identifier Number, id	Binary Identifier, ID(id)	Code Used, $\Psi_{\alpha_{id}}$
0	000	PSI1,2
1	001	PSI1,3
2	010	PSI1,4
3	011	PSI1,5
4	100	PSI1,6
5	101	PSI3

More Generality. We will henceforth refer to the above definitions. However, it is worth noting that we can further extend the generality of PSI14 by allowing $\lambda < 0$ to specify additional undefined code options PSI-1, PSI-2, ... These can be incorporated into the definitions here by noting that PSI-$\ell \equiv$ PSI-ℓ,0 as in Eq. 66. Equations 69 and 70 become

$$i(\lambda) = \begin{cases} \lambda & \lambda \leq 0 \\ 1 & \text{Otherwise} \end{cases}$$

and

$$k(\lambda) = \begin{cases} 0 & \text{if } \lambda \leq 0 \\ \lambda - 1 & \text{Otherwise} \end{cases}$$

PSI-ℓ might designate legitimate low-entropy options or simply act as escape mechanisms to alert a decoder that subsequent coding will be using completely different algorithms. Such escape mechanisms could also be provided by using code identifiers id $\geq$ N.

Decision Criteria

Optimum. The optimum criterion for selecting the best option to represent $\tilde{\delta}^n$ is to simply choose the one that produces the shortest coded sequence. Using (4) we have

Choose coder option id = id+ if

$$\mathcal{L}(\Psi_{\alpha_{id+}}[\tilde{\delta}^n]) = \min_{id} \{\mathcal{L}(\Psi_{\alpha_{id}}[\tilde{\delta}^n])\} \tag{75}$$

Simplified. The latter approach implies that each coded result must be generated to determine which option to use. The computation requirements can be drastically reduced by using estimates instead.

Let

$$\gamma_{\alpha_{id+}}(\tilde{\delta}^n) \approx \mathcal{L}(\psi_{\alpha_{id}}[\tilde{\delta}^n])$$

be the estimated coded length by using code option PSIα_{id}. Then the simplified decision criterion becomes:

Choose coder option id = id+ if

$$\gamma_{\alpha_{id+}}(\tilde{\delta}^n) = \min_{id} \{\gamma_{\alpha_{id}}(\tilde{\delta}^n)\} \tag{76}$$

Now consider estimates for the specific PSI14 options in (64).

Trivially we have

$$\mathcal{L}(\Psi_3[\tilde{\delta}^n] = \gamma(\tilde{\delta}^n) = nJ \quad (77)$$

For convenience, we repeat (43) here as

$$\mathcal{L}(\Psi_0[\tilde{\delta}^n]) \approx \Psi_0(\tilde{\delta}^n) = \left\lfloor \frac{F_0}{3} \right\rfloor + 2(F_0 - J) \quad (78)$$

Now consider the PSI1,k options in (58). First we have

$$\mathcal{L}(\Psi_{1,k}[\tilde{\delta}^n]) = \mathcal{L}(\Psi_1[\tilde{M}^{n,k}]) + Jk \quad (79)$$

since the length of the separate least-significant bit sequence is fixed.

Extending notation, we let

$$F_k = \mathcal{L}(\Psi_1[\tilde{M}^{n,k}]) \quad (80)$$

be the length of the Fundamental Sequence for the n-k most-significant bits $\tilde{M}^{n,k}$. The special case for F_0 is shown in (30) to be simply the sum of the samples of $\tilde{M}^{n,0} = \tilde{\delta}^n$ plus the block size J. This same calculation applies more generally to the samples of $\tilde{M}^{n,k}$.

Taking advantage of the randomness in the least-significant bits, the expected value for each F_k can be related to F_0 by[5]

$$E\{F_k|F_0\} = 2^{-k} F_0 + \frac{J}{2}(1 - 2^{-k}) \quad (81)$$

By using (81) we can then estimate the overall length for each PSI1,k option as, from (79)

$$\mathcal{L}(\Psi_{1,k}[\tilde{\delta}^n]) \approx \Psi_{1,k}(\tilde{\delta}^n) = 2^{-k} F_0 + \frac{J}{2}(1 - 2^{-k}) + Jk \quad (82)$$

Using the simplified decision criteria in (76) with the estimates in (77), (78) and (82) leads to clear-cut decision regions based on F_0. An example for an 8-option PSI14 with starting option PSI0($\lambda = 0$) is shown in Table 4.

Table 4. PSI14 Decision Regions, $N = 8, \lambda = 0$

Code Option	Decision Region $N = 8, \lambda = 0$
PSI0	$F_0 \leq 3J/2$
PSI1	$3J/2 < F_0 \leq 5J/2$
PSI1,1	$5J/2 < F_0 \leq 9J/2$
PSI1,2	$9J/2 < F_0 \leq 17J/2$
PSI1,3	$17J/2 < F_0 \leq 33J/2$
PSI1,4	$33J/2 < F_0 \leq 65J/2$
PSI1,5	$65J/2 < F_0 \leq (64n-351)J/2$
PSI3	$(64n-351)J/2 < F_0$

It is a simple matter to create equivalent tables for other PSI14 configurations.

Modified Simplified? The basic assumptions in determining the decision regions for the various PSI1,k options in Table 4 are based on the inherent randomness in the least-significant bits being split. This is a very good assumption for all but the $F_0 \geq 5J/2$ decision point that determines whether PSI1 or PSI1,1 should be chosen.

At low entropies, below 3 bits/sample, the least-significant bits start becoming less random. Consider how this affects the optimum decision point given by Ref. 5.

Choose PSI1,1 if

$$F_0 \geq 3J - \Sigma \text{ (least-significant bits)} \tag{83}$$

If half of the least-significant bits are ones (corresponding to random), PSI1,1 is better if $F_0 \geq 5J/2$, the same result as in Table 4.

However, as entropy drops there is a greater tendency for more zeroes to occur than ones, which culminates in all zeroes at an entropy of zero. With more zeroes, the decision region specified in (83) moves upward. This leaves open the question of whether the fixed simplified decision point of 5J/2 in Table 4 should be adjusted upward.

One can expect only minor average performance gains, if any. Actual comparisons between the simplified rule in Table 4 and the optimum rule indicate very little advantage for the optimum rule.

Baseline PSI14 Performance

PSI14 coders which use code options PSI0 or PSI1 as their starting option (λ = 0 or λ = 1) are called "Baseline" PSI14 coders.

The average measured performance for N = 8 Baseline PSI14 coders is shown in Fig. 15.

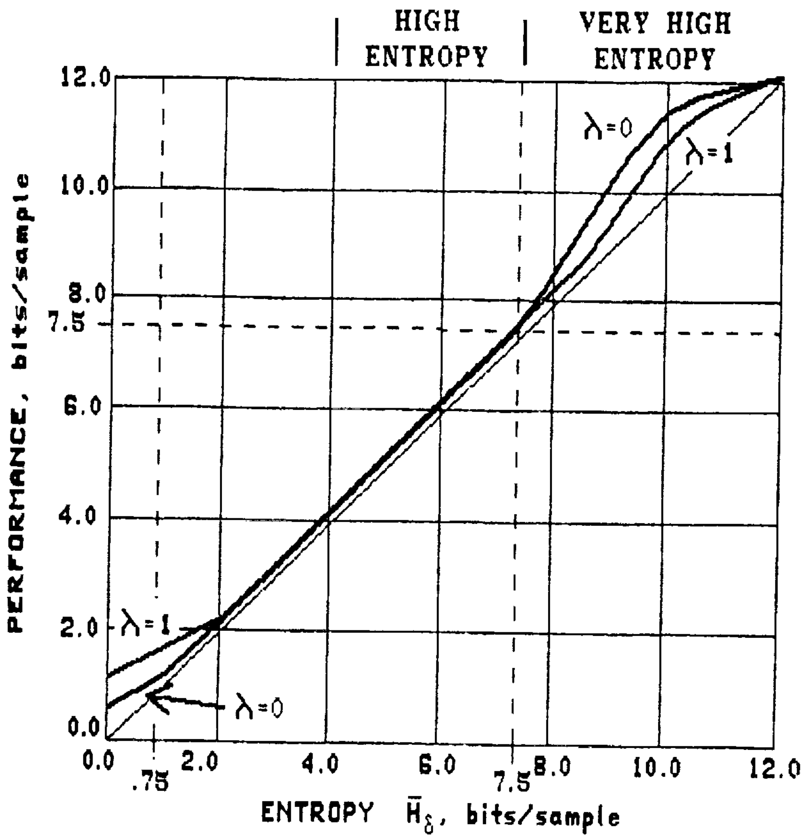


Fig. 15. Baseline PSI14 Performance, N = 8, λ = 0, 1, n = 12

These graphs were primarily derived from 12-bits/sample imaging spectrometer data and assumed a block size of J = 16. Some areas above 8 bits/sample were approximated since very few samples were available. But exact precision is not the issue here. These curves are approximately correct for almost any real problem. The **major observation is** that efficient performance from ~0.75 bit/sample to 7.5 bits/sample can be achieved with an 8-option PSI14 coder with starting option PSI0. This roughly 7-bits/sample Dynamic Range can be pushed upward by about 1 bit/sample by starting with PSI1 as the first option. The additional top-end performance is obtained at the expense of some low-end performance.

The corresponding graphs for Baseline PSI14 coders with N = 4 options are shown in Fig. 16. Voyager images were the source of the data [5]. With only four options, the Dynamic Range is, of course, much narrower.

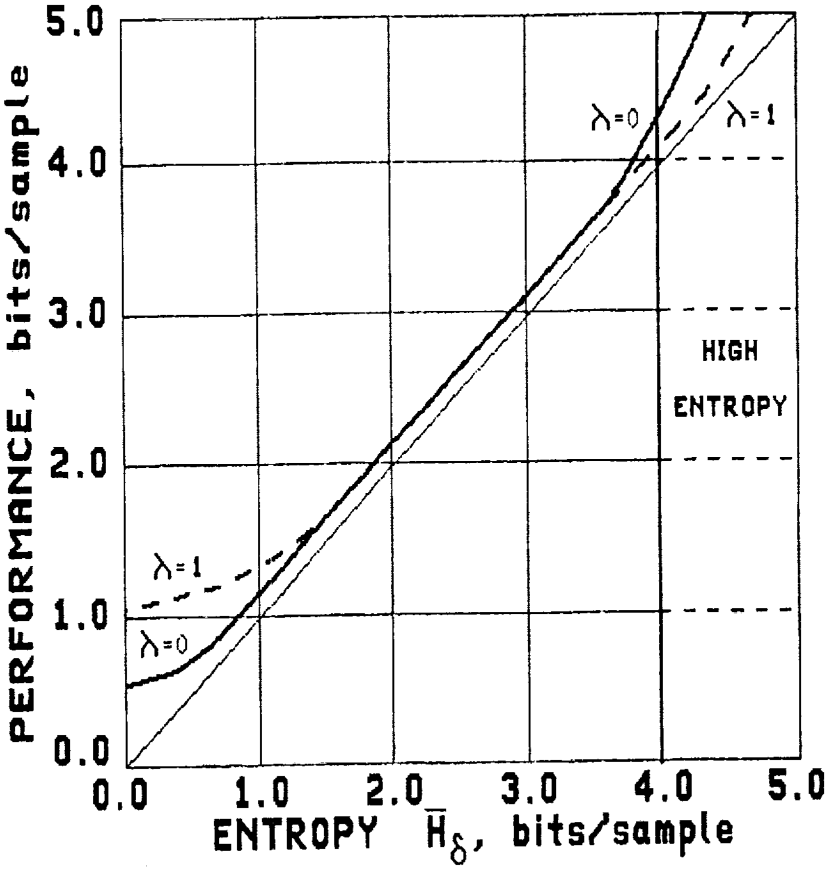


Fig. 16. Baseline PSI14 Performance, N = 4, $\lambda = 0, 1$, $n \geq 6$

Coding for Very High Entropies

The performance curves for the 8-option Baseline PSI14 coders in Fig. 15 provide a broad Dynamic Range, which should be adequate for many applications. However, many newer scientific instruments have pushed the quantization requirements to much higher levels, some to as much as 16 bits/sample. The consequence is that entropies above 8 bits/sample (viewed as "Very High Entropies") may be present much of the time. This can lead to inefficiencies if restricted to the Baseline PSI14 performance curves in Fig. 15.

Adding Options. Each additional PSI1,k option added to a Baseline PSI14 coder will extend the top end of the Dynamic Range upward by 1 bit/sample. Thus, one solution is simply to add more options until the expected "very high" entropy range is covered.

Assuming the simple fixed-length representation for the code identifier as specified by (25), to increase the number of options beyond $N = 8$ to the $9 \leq N \leq 16$ range will require an additional identifier bit. The potential performance impact is, at most, $1/J$ bits/sample or 0.0625 bit/sample for a typical block size of $J = 16$. Performance can still be considered "efficient."

Such a penalty is probably insignificant considering the almost 15-bits/sample Dynamic Range provided by an $N = 16$, $J = 16$ Baseline PSI14 coder. Moreover, if the data source causes frequent and significant variations in data entropy, the $1/J$ cost will be more than compensated by the ability to choose from a greater number of options.

Moving the Operating Range. After looking more closely at the class of real problems which generate very high entropies, it may not be necessary to add more options. We note that:

- As the highest expected data entropies increase, so usually do the lowest. Thus, the complete range of expected entropies tends to move upward, not just the top end. (84)

- Efficient coding performance is really only necessary over the expected range of data entropies. (85)

- The maximum expected range of data entropy (max entropy – min entropy) is unlikely to be as large as the Dynamic Range of efficient performance exhibited by an 8-option Baseline PSI14, as shown in Fig. 14 (approximately 7 bits/sample). (86)

These observations point to two other approaches to very-high entropy coding without extending the number of options.

The first approach is inherent in the general definition of PSI14 in (63)–(73). Just pick a starting code option for a non-baseline 8-option coder (i.e., $\lambda > 1$).

That first code option determines the lower end of efficient performance for your PSI14 coder. Pick that option so that the full 7-bits/sample Dynamic Range is centered over the expected entropy range of your problem. Table 5 should help guide you in that decision; it is derived from the performance runs in Fig. 15 and simplified decision rules, as in Table 4.

Table 5. PSI14 Dynamic Ranges for 8-option Coders, J = 16

	λ	Starting Option from (68)-(70)	DYNAMIC (Entropy) RANGE of Efficient Performance (bits/sample)
BASE-LINE	0	PSI0,0	$0.75 \le \overline{H}_\delta \le 7.5$
	1	PSI1,0	$1.5 \le \overline{H}_\delta \le 8.5$
Non-BASELINE	2	PSI1,1	$2.5 \le \overline{H}_\delta \le 9.5$
	3	PSI1,2	$3.5 \le \overline{H}_\delta \le 10.5$
	4	PSI1,3	$4.5 \le \overline{H}_\delta \le 11.5$
	5	PSI1,4	$5.5 \le \overline{H}_\delta \le 12.5$
	6	PSI1,5	$6.5 \le \overline{H}_\delta \le 13.5$
	etc.	etc.	etc.

Parametric Dynamic Range. In fact, we can extend these results to specify the Dynamic Range for any PSI14 with parameters N, n and λ.[8] By using (69) and (70), we have[9]

$$\lambda + 0.75 - 0.25\, i(\lambda) \leq \bar{H}_\delta \leq \min \begin{cases} n \\ \lambda + N - 0.5 \end{cases} \tag{87}$$

Note that the implementation of a PSI14 coding module with a shiftable Dynamic Range would require external control of the parameter λ (starting option).

We will later return to the issue of Dynamic Range after discussing another approach.

CODE OPERATOR PSI14,K

Another approach that is statistically equivalent to the Non-Baseline PSI14 coders suggested in Table 5 is obtained by returning to the basic definition of Split-Sample code option PSIi,k, which is specified in (58) and Fig. 12.

PSIi specifies the coder that is used to represent the most-significant n-k bits after splitting off k least-significant bits. The original Split-Sample coders used PSI4 for PSIi. In looking for implementation simplification, PSI1 was investigated and that led to PSI14, a coder which (except for PSI0 and PSI3—which is actually PSI1,n) uses various PSI1,k as code options. Now replace PSIi with PSI14 itself.

From Fig. 12, and using a capital K for the number of split least-significant bits, the code "option" PSI14,K becomes that shown in Fig. 17. The internal PSI14 is shown with parameters N and λ to indicate the number of options and the starting option, respectively.

8Some caution should be used in the interpretation of (87) for small block sizes ($J < 12$), where the impact of identifier bits becomes increasingly significant. The subject of identifier coding is addressed in Chapter 4.

9Where the first term would simplify to $\lambda + 0.5$ when $\lambda \geq 1$ [7].

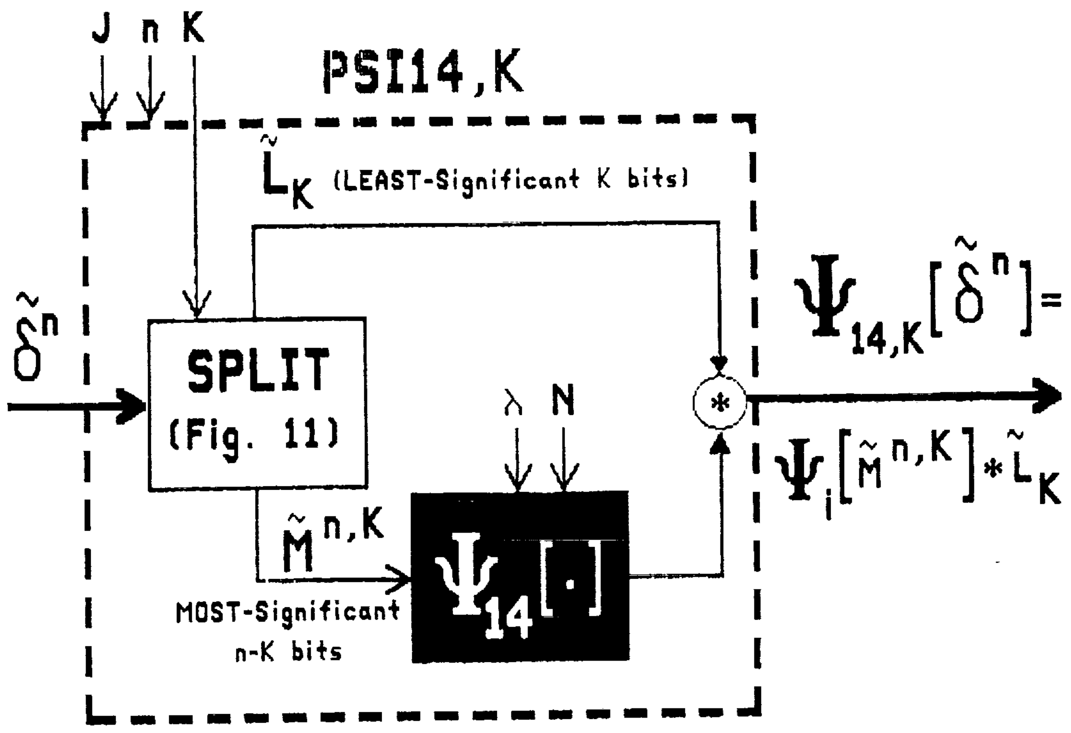

J
n
K
PSI14,K
$\tilde{L}_K$ (LEAST-Significant K bits)
$\tilde{\delta}^n$
SPLIT (Fig. 11)
λ
N
$\tilde{M}^{n,K}$
MOST-Significant n-K bits
$\Psi_{14}[\cdot]$
$*$
$\Psi_{14,K}[\tilde{\delta}^n] =$
$\Psi_i[\tilde{M}^{n,K}] * \tilde{L}_K$

Fig. 17. Split-Sample Coder PSI14,K

Performance for K = 0

With K = 0, all data pass directly to the internal PSI14, so that

$$\text{PSI14,0} \equiv \text{PSI14} \tag{88}$$

Thus, PSI14 is really a special case of PSI14,K. If the internal PSI14 in Fig. 17 is an N = 8 option Baseline PSI14, the Dynamic Range is given in Fig. 15 and (87). We will henceforth use this new notation for PSI14 to emphasize the identity when K = 0.

Performance for K > 0

To see what happens when K ≠ 0, note again that by (66) and (67), ALL the options specified for PSI14,0 are of the form PSIi,k. Returning to Fig. 17, it is easy to see that the net effect of a split of K bits is effectively to transform each PSIi,k in the internal PSI14 into another Split-Sample option with K additional splits. That is, equivalently for $\lambda \geq 0$ and $0 \leq id \leq N-2$,

$$\Psi_{i(\lambda+id),k(\lambda+id)} \rightarrow \Psi_{i(\lambda+id),k(\lambda+id)+K} \tag{89}$$

With each code option affected in the same manner, it should be clear that **the entire Dynamic Range for any PSI14,K will be shifted upward by 1 bit/sample for each increase in K.**[10]

Then we can specify the Dynamic Range for any PSI14,K configuration for $K \geq 0$ as[11]

$$K + 0.75 + \lambda - 0.25i(\lambda) \leq \bar{H}_\delta \leq \min \begin{cases} n \\ K+\lambda+N-0.5 \end{cases} \tag{90}$$

Remember, efficient means average performance close to the entropy. It doesn't mean that slightly better performance isn't possible under certain conditions. That is, one "efficient" coder option may still be better than another "efficient" coder option over a particular range. With this in mind, we look next at the comparison of an N-option PSI14,K and its closest equivalent N-option PSI14,0.

INTERNAL $\lambda \geq 1$. If we momentarily exclude the possibility of using PSI0 as a PSI14,0 option, we can say that for any N-option PSI14,K there is an equivalent PSI14,0 with exactly the same Dynamic Range. That is:

If an N-option internal PSI14,0 of PSI14,K starts with option PSI1,k(λ), $\lambda > 0$, then an N-option PSI14,0 that starts with PSI1,k(λ) + K will exhibit the same Dynamic Range for $K \geq 0$.

INTERNAL $\lambda = 0$. When we include the possibility of PSI0 as the starting option we cannot make as precise a statement.

Consider the case where K = 0, 1, 2. The effective options for PSI14,K are

---

[10]Of course, this is not precise when the top end of the performance range approaches n, the number of quantization bits. The same is true at the very low end where the LSBs are not totally random, as discussed earlier.

[11]Subject to the same caution for small values of $J < 12$.

$$\left.\begin{array}{lllll} K = 0 & PSI0,0 & PSI1,0 & PSI1,1 & PSI1,2... \\ K = 1 & PSI0,1 & PSI1,1 & PSI1,2 & PSI1,3... \\ K = 2 & PSI0,2 & PSI1,2 & PSI1,3 & PSI1,4... \end{array}\right\} \tag{91}$$

Indeed, the Dynamic Range will shift upward by approximately 1 bit/sample for each increase in K. But are these configurations identical to some PSI14,0 with a different starting option?

For this example, and in general, there is no PSI14,0 configuration that includes starting option PSI0,k' for k' > 0 as an option.

If we replace starting option PSI0,k' with PSI1,k'−1, we do get a legitimate PSI14,0 defintion since all the code options after the first already fit. So the only question is, How does PSI0,k' compare with PSI1,k'−1? This comparison is statistically the same for every $k' \geq 1$, it just happens at different overall entropy values. So we only need to compare PSI0,1 with PSI1 = PSI1,0 at low entropies; at higher entropies, other options will be used.

We provide this comparison in Appendix C and conclude that over the low entropy range of interest, the average performance of PSI1 is probably always slightly better than that of PSI0,1. Using this observation and (90), we can say that for K > 0; over the entropy range of the first option

$$0.5 + K \leq \bar{H}_\delta \leq K + 1.5 \tag{92}$$

a PSI14,K with internal N-option PSI14,0 with $\lambda = 0$ will not perform quite as well as one with internal N-option PSI14,0 which starts with PSI1,K-1. Performance over the remainder of the Dynamic Range should be identical.

From Yeh's result this is not surprising [11]. She showed that the PSI1,k options are equivalent to Huffman codes over the range of interest.

PSI14,K vs PSI14,0 for K > 0

Basically, the difference in performance between any PSI14,K configuration and its nearest equivalent PSI14,0 is either nonexistent or too minor to exclude one approach or the other. The choice should be based on implementation considerations. Let us now discuss some specific implementation examples.

Single Dynamic Range. Here we must design a coder which works efficiently over a single prescribed entropy range. If this coder is being designed from scratch, there is no point in including PSI0 as an option since it offers no advantage when $K > 0$ (higher entropies). Under these conditions, the performance of either approach is identical. On the surface at least, it's a toss-up.

However, if the design of a fixed PSI14,0 with starting option PSI0 already exists, it is clearly simpler to turn it into a fixed PSI14,K than to redesign a new PSI14,0 with a new λ to meet a specific Dynamic Range goal.

Multiple Dynamic Ranges. In this case the implemented coder must include an external parameter to adjust the Dynamic Range: Either K for PSI14,K or λ for PSI14,0. Now $K \geq 0$ and $\lambda \geq 0$ are possible all in the same coder. External control of these parameters would be essentially identical to a user.

A PSI14,K has one major advantage: The adaptive part of the coder is fixed (i.e., the internal PSI14,0). All the code-option decision making and the assignment of identifiers would not change as K was adjusted. This is not true for a PSI14,0 with adjustable λ.

On the other hand, PSI14,0 with adjustable λ maintains a slightly simpler output format. But overall, the edge in simplicity for this situation seems to be with PSI14,K.

Stretch PSI14 (sPSI14)

Another way to obtain a broad Dynamic Range, at some penalty in performance, is to build an adaptive coder that only uses every other standard PSI14 code option; we denote this by sPSI14.

By using the notation developed in (63)–(73), we can define all the code options of an N-option sPSI14 by

$$\Psi_{i(\lambda+2id),\, k(\lambda+2id)} \tag{93}$$

$$\text{for } \lambda \geq 0 \text{ and } 0 \leq id \leq N-2$$

and, as for PSI14 itself,

$$\Psi_{\alpha_{N-1}} \equiv \Psi_3 \tag{94}$$

Such an sPSI14 will have a "potential" Dynamic Range given by

$$\lambda + 0.75 - 0.25i(\lambda) \leq \bar{H}_\delta \leq \min \begin{cases} n \\ \lambda+2(N-1)-0.5 \end{cases} \tag{95}$$

for n-bit input data. For large N, this is roughly double the range given in (87) for a PSI14 using the same number of options.

We say "potential" Dynamic Range because there is a performance penalty in skipping options.

The consequence should be apparent from Fig. 18, which shows how the performance of three adjacent PSI1,k are related. These graphs are not precise and are only intended to make a point. Greater accuracy under various conditions is provided by Yeh [11].

Referring to Fig. 18, an sPSI14 would have options PSI1,m and PSI1,m+2 but not PSI1,m+1. Under ideal stationary conditions, the average performance given up by not including PSI1,m+1 is indicated by the crosshatched region (adjusted slightly because generally one less identifier bit would be needed). Under most real conditions, it is speculated that this average penalty will be spread over a broader entropy range. An sPSI14 performance curve will look like a slightly inefficient and "bumpy" PSI14 performance curve over the same entropy range. Specific details must await simulations.

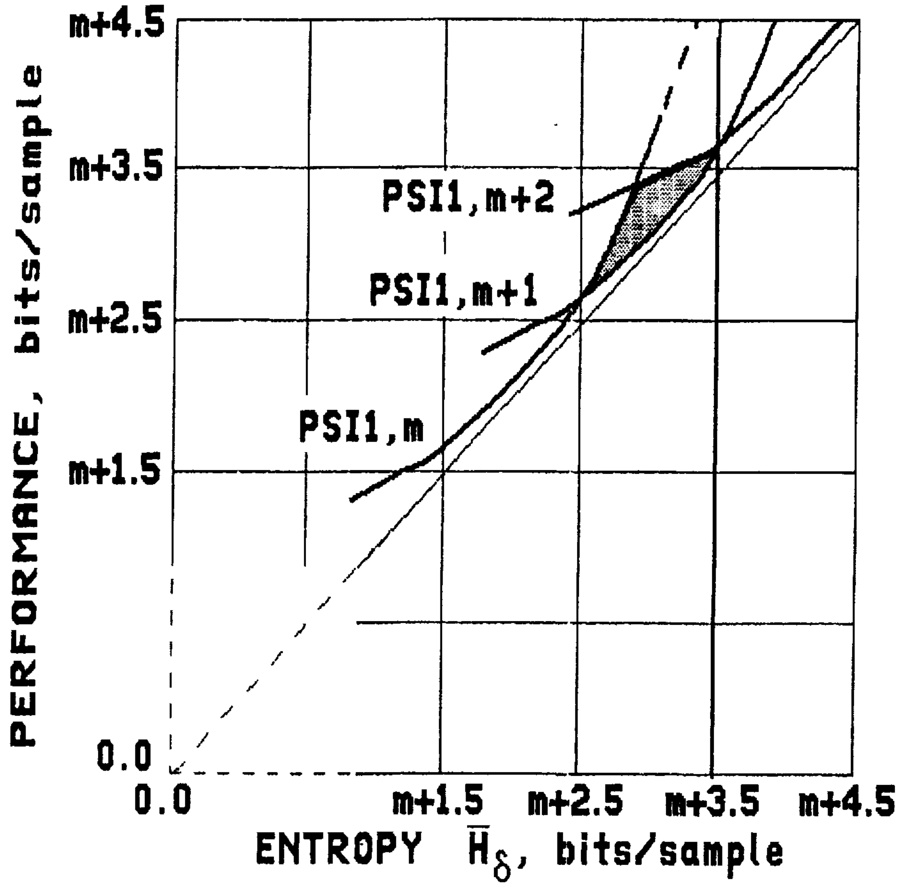


Fig. 18. Skipping an Option

CURRENT VLSI IMPLEMENTATIONS

GSFC and the University of Idaho have implemented a custom 1.0-µm CMOS VLSI version of the Coding Module in Fig. 4, as well as a compatible "decoding" module [8]. The coder incorporates most of the general-purpose functionality in that diagram. However, the PSI14,K coder included is actually PSI14,0 with N = 12 options, $\lambda = 1$ and J = 16. Input quantization can vary from n = 4 to n = 14. The coder and the decoder were recently tested successfully under laboratory conditions. The coder was operated at input data rates of up to 700 Mbits/s and the decoder at half that rate.

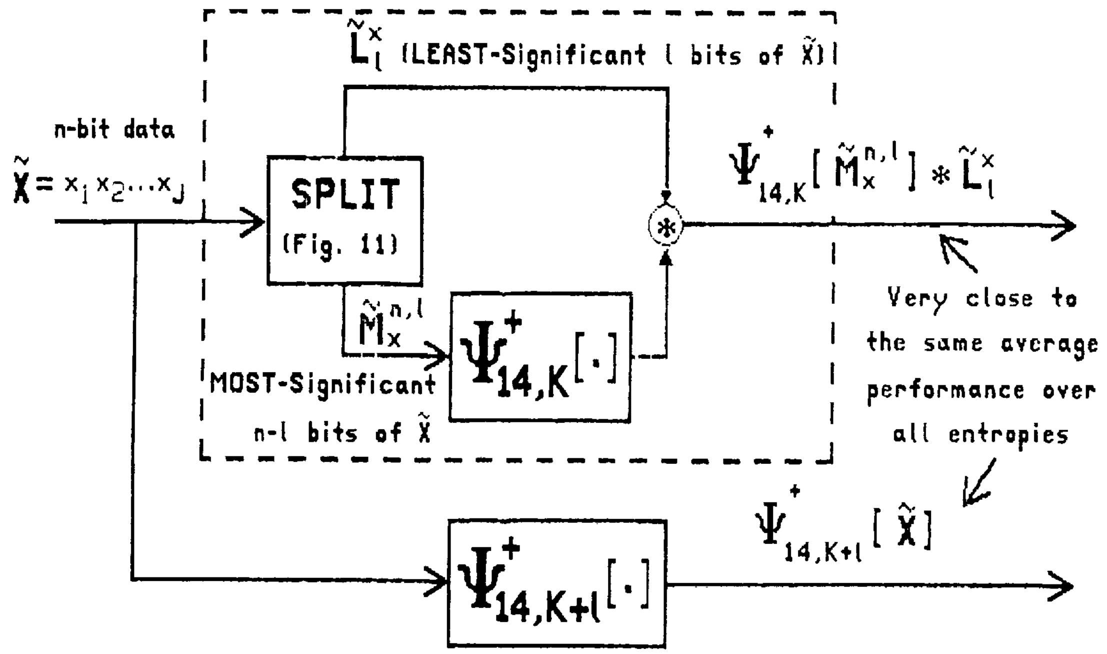
JPL has used different technologies to implement two custom 1.6-µm CMOS VLSI versions of the Coding Module in Fig. 4. Because of certain instrument-specific constraints, these modules include fewer of the more general functions of Fig. 4. The internal coders are in each case PSI14,0 with N = 11 options, $\lambda = 1$, J = 16 and allowance for input quantization of n = 12 bits/sample. Both versions have been tested in the laboratory at up to 180 Mbits/s [9]. An enhanced version of this coder is being implemented for the CRAF/Cassini project.

IV. OUTSIDE THE MODULE

Chapters II and III have filled in the details of coding module PSI14,K+ shown in Fig. 4. This chapter briefly addresses some issues relating to the use of PSI14,K+ within a "compression system."

EXTERNAL SPLIT-SAMPLES

One interesting observation that can affect a compression system's implementation is that the concept of Split-Sample Modes can also be accomplished outside the module for many applications, as illustrated in Fig. 19.



$\tilde{L}_l^x$ (LEAST-Significant l bits of $\tilde{X}$)
n-bit data
$\tilde{X} = x_1 x_2 \ldots x_J$
SPLIT (Fig. 11)
$\tilde{M}_x^{n,l}$
MOST-Significant n-l bits of $\tilde{X}$
$\Psi^+_{14,K}[\cdot]$
$*$
$\Psi^+_{14,K}[\tilde{M}_x^{n,l}] * \tilde{L}_l^x$
Very close to the same average performance over all entropies
$\Psi^+_{14,K+l}[\tilde{X}]$
$\Psi^+_{14,K+l}[\cdot]$

Fig. 19. Internal vs External Splits

In this illustration:

- $\tilde{X}$ is a "raw" n-bit data sequence (e.g., imaging or magnetometer data).
- The indicated PSI14,K+ uses the built-in preprocessor directly or with an external predictor.

The **upper coding process**, within the dashed lines, first performs an ℓ-bit split directly on $\tilde{X}$. By using the Split-Sample coding structure shown in Fig. 12, the sequence of most-significant n-ℓ bit samples of $\tilde{X}$ (i.e., $\tilde{M}_X^{n,\ell}$) are coded by PSI14,K+, and the least-significant ℓ-bit samples (i.e., $\tilde{L}_\ell^X$) are passed on unchanged.

In the **lower coding process**, all $\tilde{X}$ is passed directly to a PSI14,K+ module where the parameter K has been increased by ℓ to $K' = K + \ell$.

The message from this figure is that the upper and lower coding processes perform almost identically and exhibit the same Dynamic Range.[12] Computer simulations indicate that performing Split-Sample operations all with PSI14,K+ (the lower path) gives a slight edge in performance of less than .05 bit/sample [5].

Some observations:

- The effective Dynamic Range of a given PSI14,K+ module can be shifted upward externally to the module (e.g., to entropy values not supported by the largest value of K or λ). The Galileo image compressor uses this approach to shift the performance for a complete image line [13].
- A PSI14,K+ module that can only accommodate data quantized up to n bits/sample might be usable on n+ℓ bit data if the upward ℓ bit/sample shift in Dynamic Range is acceptable. For example, a PSI14,K coder with a Dynamic Range $2.5 \leq \bar{H}_\delta \leq 8.5$, which was originally designed to work only on 12 bits/sample data, could now be used on 14 bits/sample data, but with a new Dynamic Range of $4.5 \leq \bar{H}_\delta \leq 10.5$.
- Another advantage is provided in systems employing Priority Driven Rate Control, as discussed in a later section.

[12]This would also be true for PSI14,0 with starting option parameter λ increased by ℓ.

DECODING PREPROCESSED DATA

PSI14,K+ allows for input data to be passed directly to the adaptive variable-length coding section, PSI14,K without preprocessing. An input block $\tilde{\delta}^n$ is coded as PSI14,K[$\tilde{\delta}^n$]. Given this coded sequence, $\tilde{\delta}^n$ can be reconstructed precisely; no other information is required. This is not necessarily the case when preprocessing is involved.

Let us review the preprocessing and coding process at the sample level. The error between a sample x_i and its predicted value $\hat{x}_i$ produces the error value

$$\Delta_i = x_i - \hat{x}_i \qquad (96)$$

which is converted to

$$\delta_i \qquad (97)$$

by the reversible process in (17), or by (18) and (19). Blocks of δ_i are then coded. During reconstruction, each Δ_i can clearly be retrieved by reversing these steps. Then the original sample x_i can be reconstructed as

$$x_i = \hat{x}_i + \Delta_i \qquad (98)$$

Thus, reconstruction of any individual sample can be accomplished, provided that its original prediction is known or can be recomputed from available information.

To see how this fits various applications, we must define some terms to handle sequences of blocks. Let

$$\tilde{X} = \tilde{X}_1 * \tilde{X}_2 * \ldots * \tilde{X}_m \qquad (99)$$

denote a sequence of blocks, where the individual $\tilde{X}_i$ are known a priori to be J_i samples long, so that

$$\tilde{X}_i = x_{i,1}\, x_{i,2} \cdots x_{i,J_i} \qquad (100)$$

The corresponding vector of prediction values used during the coding process is then

$$\hat{\tilde{X}}_i = \hat{x}_{i,1}\ \hat{x}_{i,2} \ldots \hat{x}_{i,J_i} \tag{101}$$

and the resulting error sequence is then

$$\Delta_i = \Delta_{i,1}\ \Delta_{i,2} \ldots \Delta_{i,J_i} \tag{102}$$

where

$$\Delta_{i,\ell} = x_{i,\ell} - \hat{x}_{i,\ell} \tag{103}$$

The equivalent prediction and error sequences for multiple blocks are then

$$\hat{\tilde{X}} = \hat{\tilde{X}}_1 * \hat{\tilde{X}}_2 * \ldots \hat{\tilde{X}}_m \tag{104}$$

and

$$\tilde{\Delta} = \tilde{\Delta}_1 * \tilde{\Delta}_2 * \ldots \Delta_m \tag{105}$$

We can then say that all $\tilde{X}$ can be recovered precisely from $\tilde{\Delta}$, provided that $\hat{\tilde{X}}$ is known or can be computed.

Thus, the coding of each $\tilde{X}_i$ by PSI14,K+ is really a function of the prediction used, as

$$\text{PSI14,K} + [\tilde{X}_i, \hat{\tilde{X}}_i] \tag{106}$$

rather than the short form

$$\text{PSI14,K+}[\tilde{X}_i] \tag{107}$$

which we have been using (and will continue to use); (106) merely emphasizes what we have implicitly assumed in (107).

One-Dimensional Predictor

For this case we assume that the $\tilde{X}$ in (99) really represents a partitioning of a single long sequence of sampled data from a single source into several smaller blocks. That is, the first sample of $\tilde{X}_i$ follows the last sample of block $\tilde{X}_{i-1}$. All the samples are adjacent in time. For example, $\tilde{X}$ could represent a single image line.

Using the built-in predictor of PSI14,K+ means predicting that the next sample is the same as the last, so that for block $\tilde{X}_i$

$$\hat{x}_{i,\ell} = x_{i,\ell-1} \text{ for } \ell \geq 2 \tag{108}$$

and

$$\hat{x}_{i,\ell} = x_{i-1,J_{i-1}} \text{ for } \ell = 1 \tag{109}$$

This means that the prediction is always known, provided that a "last sample" exists. This is true except for the first sample of the first block $x_{1,1}$. Prediction for the first sample must be supplied to "initialize" both a PSI14,K+ coder and decoder. The coder must have $\hat{x}_{1,1}$ to generate the first error, $\Delta_{1,1}$, and a decoder needs it to recompute the first sample $x_{1,1} = \hat{x}_{1,1} + \Delta_{1,1}$. This initial prediction value is usually called a Reference Sample.

The overall coding of $\tilde{X}$ can be described by

$$\Psi^{+}_{15,K}[\tilde{X}] = \text{REF} * \Psi^{+}_{14,K}[\tilde{X}_1] * \Psi^{+}_{14,K}[\tilde{X}_2] * \ldots * \Psi^{+}_{14,K}[\tilde{X}_m] \tag{110}$$

where it is presumed that the PSI14,K+ module is initialized by "REF."

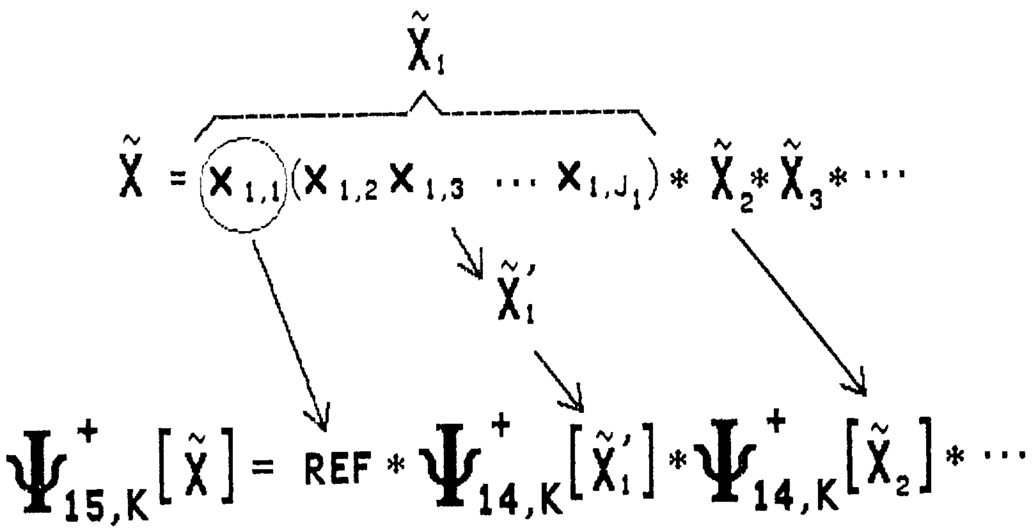
In practice, REF can be obtained in several ways:

a) If REF is known, it can be omitted altogether.

b) If REF was set to some arbitrary known constant, it could also be omitted. However, the initial $\Delta_{1,1}$ could be quite large, causing the wrong code option to be used for the remainder of the first block, $\tilde{X}_1$. This could be expensive in bits.

c) REF could be set to the first sample in $\tilde{X}_1$, causing the initial difference to be a zero. Keep in mind that the contribution of a zero to any Fundamental Sequence is only 1 bit (see 27).

d) The first block can be split into two parts: A Reference Sample and a new $J_1 - 1$ sample block $\tilde{X}_1'$ that begins with the second sample of $\tilde{X}_1$, as illustrated in Fig. 20.



$\tilde{X}_1$
$\tilde{X} = (x_{1,1})(x_{1,2}\, x_{1,3} \cdots x_{1,J_1}) * \tilde{X}_2 * \tilde{X}_3 * \cdots$
$\tilde{X}_1'$
$\Psi^{+}_{15,K}[\tilde{X}] = \mathrm{REF} * \Psi^{+}_{14,K}[\tilde{X}_1'] * \Psi^{+}_{14,K}[\tilde{X}_2] * \cdots$

Fig. 20. Reference Sample Extraction

This approach can be better than (c) by only 1 bit, but there seems to be a hardware advantage [8]. Consequently, the University of Idaho VLSI team has incorporated the capability to extract a Reference Sample in this way into their current VLSI module implementation.

External Predictor

The PSI14,K+ module allows for input of an external prediction on a sample-by-sample basis. We will consider a few practical applications here. First, note that the form of coding specified in (109) does not really depend on the type of predictor used. Of course, the actual prediction must be known for each sample by both coder and decoder. PSI15,K+ defines the form for coding a sequence of blocks using PSI14,K+, regardless of the prediction method used. The REF sample may or may not actually be present.

Two-dimensional (2-D) Arrays. Fig. 6 introduced the concept of using 2-D prediction for image applications. We can now be more specific about formats and initialization problems.

To aid in the discussion we again extend our notation to include a parameter to specify the jth line of a 2-D array. Then

$$\{\tilde{X},\ \tilde{X}_i,\ \hat{\tilde{X}}_i, \ldots\} \rightarrow \{\tilde{X}(j),\ \tilde{X}_i(j),\ \hat{\tilde{X}}_i(j), \ldots\} \qquad (111)$$

where now $\tilde{X}(j)$ represents the jth line, etc. For simplicity we presume that the length of any block in one line is the same as the corresponding block in another line, as illustrated in Fig. 21.

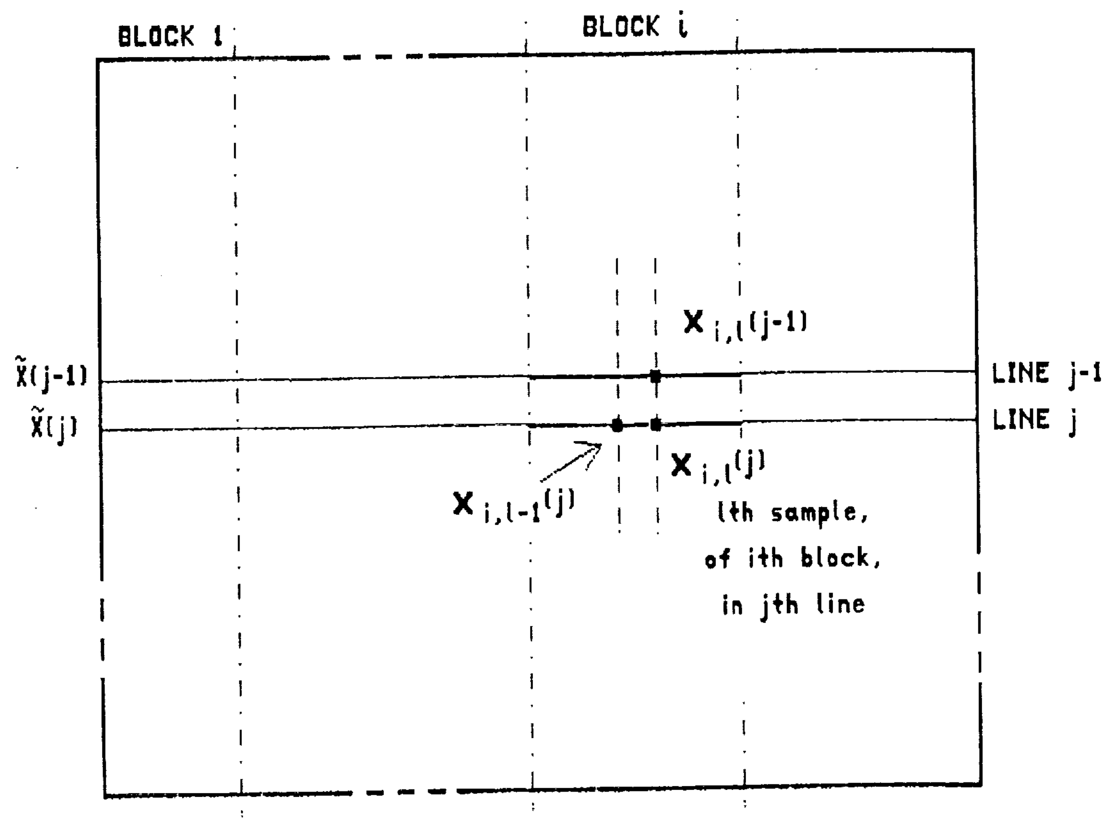


Fig. 21. Two-Dimensional Array

Using Fig. 6, we have the equation for a 2-D prediction of the ℓth sample in the ith block of line j, as[13]

$$\hat{x}_{i\ell}(j) = \frac{x_{i,\ell-1}(j) + x_{i,\ell}(j-1)}{2} \tag{112}$$

where $x_{i,-1}(j)$ is interpreted to be the last sample of the preceding block, $x_{i-1,J}(j)$. This is a legitimate prediction, provided that each term exists. Unfortunately, such is not the case for all i,j,ℓ. We need to investigate how the predictions should be modified to ensure the reconstruction of $x_{i,\ell}(j)$ as

$$x_{i,\ell}(j) = \hat{x}_{i,\ell}(j) + \Delta_{i,\ell}(j) \tag{113}$$

[13]There are other possible versions which could be substituted for Eq. 112. In most cases they are unlikely to alter performance significantly.

Consider the most common arrangement for coding and decoding: Line j follows Line j–1, and each line is coded from left to right.

When $\tilde{X}(1)$ is coded, there is no previous line, so the term $x_{i,\ell}(j-1)$ is not valid. Thus, $\tilde{X}(1)$ should be coded with a one-dimensional (1-D) predictor, as described in the previous section.

On subsequent lines, only the first term of the first block is missing, $x_{1,1}(j)$ in (112). This could be handled in a number of ways. From a performance point of view, the best method is to employ a 1-D prediction by using the first sample from the line above, as

$$\hat{x}_{1,1}(j) = x_{1,1}(j-1) \tag{114}$$

This is generally as good as a 1-D prediction in the same line.

Fig. 22 reviews the prediction strategy described above.

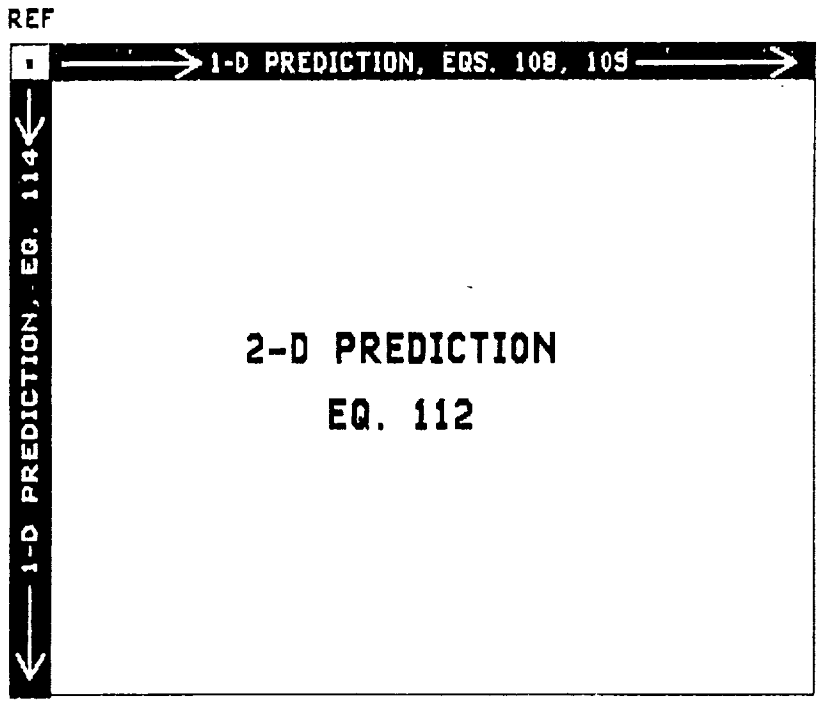


Fig. 22. 2-D Prediction Regions, Common Format: Top-to-Bottom, Left-to-Right Coding

The array coding format for the jth line takes the form from (110)

$$\Psi^{+}_{15,K}[\tilde{X}(j)] = \mathrm{REF} * \Psi^{+}_{14,K}[\tilde{X}_1(j)] * \ldots * \Psi^{+}_{14,K}[\tilde{X}_m(j)] \tag{115}$$

If the prediction of (114) is used, then REF will be present only for j = 1. In either case, the coding of the complete array takes the form

$$\Psi^{+}_{15,K}[\tilde{X}(1)] * \Psi^{+}_{15,K}[\tilde{X}(2)] * \ldots \tag{116}$$

that is, coded line 1 followed by coded line 2, etc.

Sensor Noise. Some scientific instruments generate a two-dimensional image by "sweeping" a single line of individual one-picture element sensors across a target. The sweeping action is usually supplied by spacecraft motion. For example, the High-Resolution Imaging Spectrometer (HIRIS) instrument [14] will simultaneously sweep 192 such line sensors, each representing different spectral bands (and producing separate two-dimensional images). The direction along the swept line is usually called "Cross-track" and the direction being swept is called "Down-track." Figure 23 is provided to help keep these distinctions in mind.

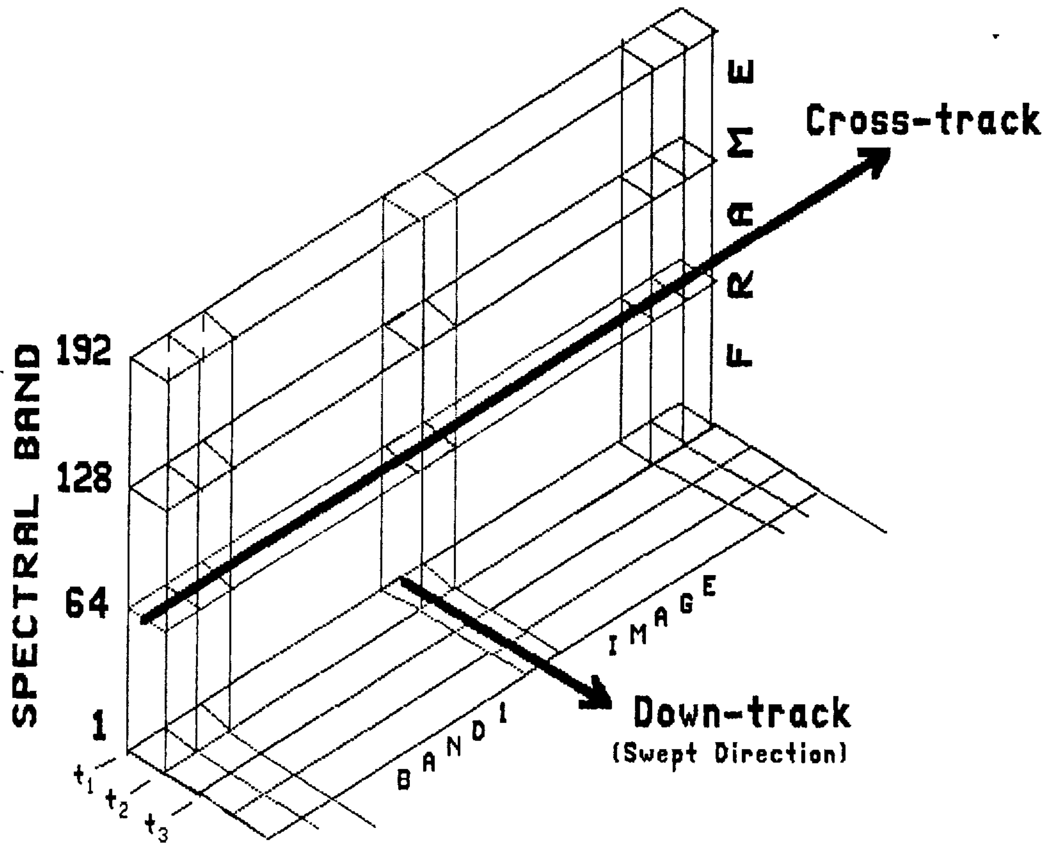


Fig. 23. Images Formed by Swept-Line Sensors

The figure shows a single line of sensor elements (running cross-track) from each spectral band. At time t_1, each line of sensor elements is exposed to a corresponding line of reflections from earth, which creates a single digital line of a familiar image (see below).

The composite of all such lines is called a "frame." At time t_2, the spacecraft has moved the "view" of each line sensor to the next "Down-track" position, which creates a second frame. At time t_3, the view has again moved, and so on. The composite of the digital lines created by each swept spectral line forms a two-dimensional image, as illustrated for the first spectral band.

Unfortunately, the individual uncalibrated sensor elements of today's instruments tend to have different gain and offset characteristics. This means that each sensor element will not necessarily generate the same output for a given input. The result appears as an additional randomness (which we can call SENSOR NOISE) when trying to predict the values of one sensor element by using the known values of another along this line (cross-track). That is, for example, the use of the built-in predictor of a PSI14,K+ module. The consequence of this additional randomness is an increase in prediction entropy, and hence code rate. The potential impact on code rate caused by this Sensor Noise is discussed in Appendix D. Although there are really no hard numbers available at this time, the potential increase to Cross-track prediction entropy for some of the latest high-performance instruments could be as high as 1 bit/sample.

But note that a prediction based on the same sample in the previous line (e.g., $\hat{x}_{1,1}(j)$ in Eq. 114) would not be affected since both samples originate from the same sensor element, as a result of sweeping the line of sensor elements (see Fig. 23). Then, by assuming symmetric data characteristics, a lower prediction entropy would be achieved by predicting using only the "previously generated" line. By extending Eq. 114 to all the samples of a line, we have

$$\hat{x}_{i,\ell}(j) = x_{i,\ell}(j-1) \tag{117}$$

Note that using this kind of "Down-track" predictor does not mean that a large multi-line buffer is necessary. Only the previous line is needed to produce the predictions in (117). The coding still takes the form in (116). Only the predictor has changed, as illustrated in Fig. 24.

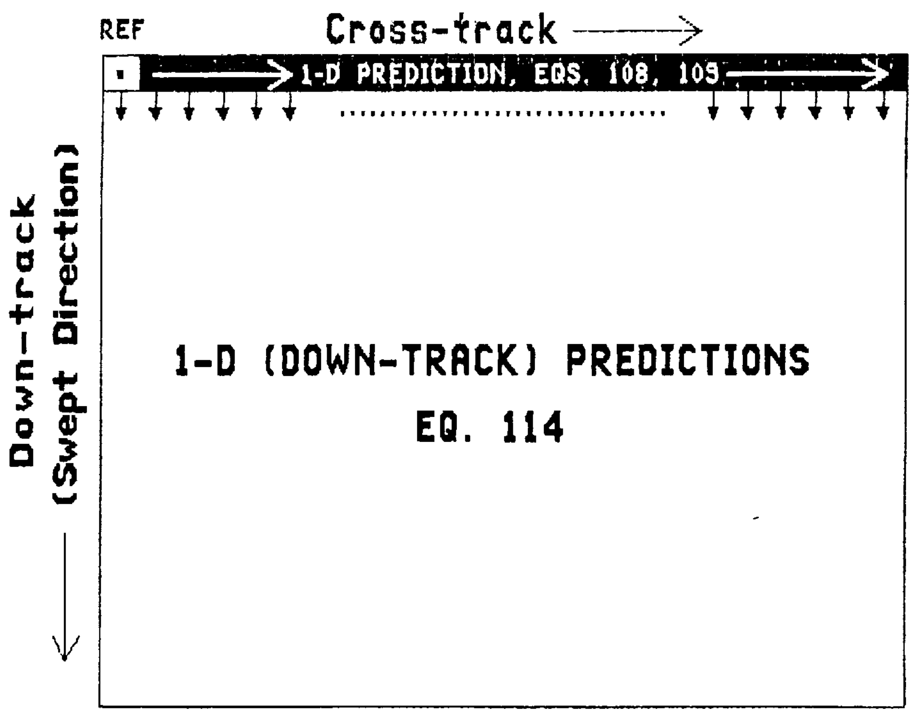


Fig. 24. 1-D Down-Track Prediction Strategy
for an Uncorrected Swept-Line Sensor

A similar look at a two-dimensional predictor reveals that it would be affected less than a left-to-right 1-D prediction because only one of the terms in (112) has noise riding on it. But typically the most one can expect to gain from a two-dimensional prediction is about 0.5 bit/sample reduction in entropy. This advantage might easily be canceled by the above-mentioned noise effects for some current modern instruments.

More General 2-D. It should be noted that the presumed line-by-line, left-to-right ordering in the examples described above is really a convention, not a constraint. The real constraint is spelled out in (96) to (98): The coder and decoder must ultimately use the same prediction for each sample that is coded and decoded. There is no reason that lines could not be coded in opposite directions, or in any order, provided that appropriate modifications are made to the prediction strategies to assure compatibility between coder and decoder. In fact, we will need to take some liberty in such assumptions in later discussions to avoid getting bogged down in notation.

In particular, to aid discussion of higher level issues, we will use

$$\text{PSI2D} \tag{118}$$

to denote a general purpose 2-D array coder, of which Figs. (22) and (23) and Eqs. 115 and 116 provide good examples. When we specify PSI2D, we will generally ignore coding details and focus on prediction strategies and the relationship of one array to another.

Extreme Reordering Example. An extreme example of reordering the coding and communication of a 2-D array is shown in Fig. 25. Here it is presumed that:

1) A large 2-D array, $\tilde{A}$, is partitioned into 16 equal-size sub-arrays, $\tilde{A}_1, \tilde{A}_2, \ldots \tilde{A}_{16}$. (119)

2) All the data (each $\tilde{A}_i$) are generated in a buffer before coding begins. (120)

3) A prescribed order of coding and communications of the $\tilde{A}_i$ is

$$\begin{gathered}\tilde{A}_7, \tilde{A}_2, \tilde{A}_5, \tilde{A}_3, \tilde{A}_6, \tilde{A}_{15}, \\ \tilde{A}_8, \tilde{A}_{13}, \tilde{A}_{10}, \tilde{A}_{11}, \tilde{A}_1, \tilde{A}_9, \\ \tilde{A}_{14}, \tilde{A}_4, \tilde{A}_{12}, \tilde{A}_{16}\end{gathered} \tag{121}$$

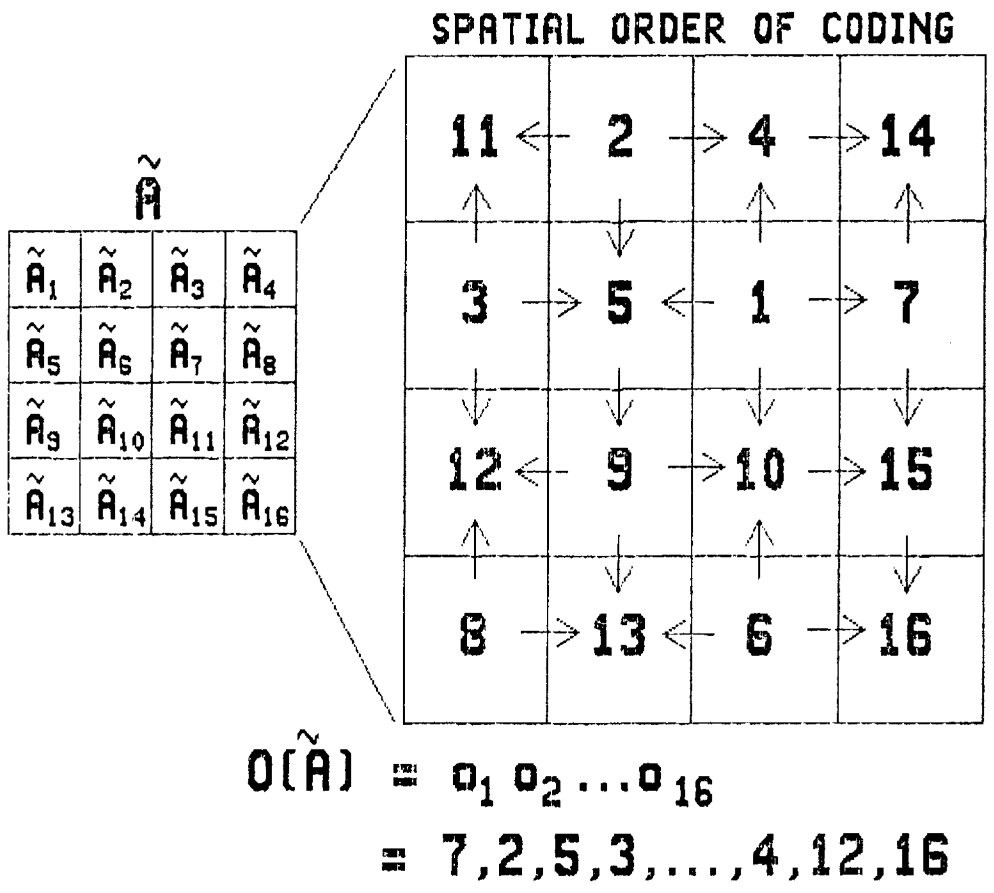

SPATIAL ORDER OF CODING
$\tilde{A}$
$\tilde{A}_1$ $\tilde{A}_2$ $\tilde{A}_3$ $\tilde{A}_4$
$\tilde{A}_5$ $\tilde{A}_6$ $\tilde{A}_7$ $\tilde{A}_8$
$\tilde{A}_9$ $\tilde{A}_{10}$ $\tilde{A}_{11}$ $\tilde{A}_{12}$
$\tilde{A}_{13}$ $\tilde{A}_{14}$ $\tilde{A}_{15}$ $\tilde{A}_{16}$
11 2 4 14
3 5 1 7
12 9 10 15
8 13 6 16
$O[\tilde{A}] = o_1\, o_2 \ldots o_{16}$
$= 7,2,5,3,\ldots,4,12,16$

Fig. 25. Example: Changing the Order of Coding 2-D Sub-Arrays

More generally, we see that (121) is a special case for an ordering specified by

$$O(\tilde{A}) = O_1\, O_2 \ldots O_{16} \tag{122}$$

where O_j = array number for the jth coded sub-array. For the specific example in (121) and Fig. 25

$$O(\tilde{A}) = 7, 2, 5, 3, 6, 15, 8, 13, \\ 10, 11, 1, 9, 14, 4, 12, 16 \tag{123}$$

Clearly, the coding of each $\tilde{A}_i$ can take the form in (115) and (116), or more generally PSI2D in (118). The only significant unanswered questions are, How are the prediction of $\tilde{A}_i$ samples specified? and Do we need to communicate additional information on the transmission ordering of the $\tilde{A}_i$?

Case I: The $\tilde{A}_i$ are independently coded (no information from surrounding sub-arrays is used), and the ordering of sub-array transmission is known a priori at the coder and decoder.

No other information is needed, so the coding of $\tilde{A}$ will appear as

$$\mathrm{PSI2D}[\tilde{A}_7] * \mathrm{PSI2D}[\tilde{A}_2] * \mathrm{PSI2D}[\tilde{A}_5] * \ldots \tag{124}$$

for the example specified in (119)–(121).

Case II is the same as Case I, except the ordering of sub-array transmission (coding) is unknown a priori to the decoder. Then the coding of $\tilde{A}$ can be described in general by the format

$$O(\tilde{A}) * \mathrm{PSI2D}[\tilde{A}_{0_1}] * \mathrm{PSI2D}[\tilde{A}_{0_2}] * \ldots \tag{125}$$

or

$$O_1 * \mathrm{PSI2D}[\tilde{A}_{0_1}] * O_2 * \mathrm{PSI2D}[\tilde{A}_{0_2}] * \ldots \tag{126}$$

where the order numbers (see Eqs. 122, 123), would, of course, be converted to binary numbers for use in (125) and (126).

For the specific ordering in (121), (122), and Fig. 25, (126) becomes

$$7 * \mathrm{PSI2D}[\tilde{A}_7] * 2 * \mathrm{PSI2D}[\tilde{A}_2] * \ldots \tag{127}$$

Case III is the same as Case II, except that the predicted boundary samples of any $\tilde{A}_j$ can make use of adjacent samples from surrounding $\tilde{A}_j$.

The additional rule to follow is simply that the boundary samples from adjacent sub-arrays can be used in a prediction IF that sub-array has already been coded and transmitted (so a decoder would be able to use the same data). To illustrate, Fig. 25 exhibits arrows pointing across the boundaries between sub-arrays. Arrows from $\tilde{A}_k$ into $\tilde{A}_j$ mean that the corresponding adjacent samples of $\tilde{A}_k$ can be used to improve the prediction of $\tilde{A}_j$. In the Fig. 25 example, the top row of $\tilde{A}_6$ can be used to aid the prediction of the bottom row of $\tilde{A}_{10}$, because $\tilde{A}_6$ is transmitted before $\tilde{A}_{10}$. In (117), $\hat{x}_{i,\ell}(j)$ would become $\hat{x}_{i,\ell}(j) = x_{i,\ell}(j+1)$ instead.

Of course we are ignoring details, but it is important to note that the algorithm which specifies how (and if) the coding of a particular $\tilde{A}_j$ will use the samples from (the boundaries of) adjacent sub-arrays can be determined solely by the order of transmission. No other information is necessary to tell a decoder what the coder did.

Whether the additional coding gains obtained by improved prediction on the sub-array boundaries are worth the additional complexity is another question altogether.

An Archive. The manner in which an array $\tilde{A}$ is used can produce other constraints which may override the quest for maximum coding efficiency. While the arrangement suggested in Case III above might provide a slight improvement in efficiency over Case I and II (a lot depends on the size of the $\tilde{A}_j$), full knowledge of all $\tilde{A}_{O_j}$, $i < j$ is required to decode $\tilde{A}_{O_j}$. In the worst case, all $\tilde{A}$ must be decoded in order to decode the last $\tilde{A}_j$ ($\tilde{A}_{16}$ for the specific order in (123)).

In an image archive, $\tilde{A}$ might represent a complete image, and the $\tilde{A}_i$ might represent spatial subimages (with typically many more than 16 subdivisions). In this case it is clearly desirable to be able to directly access and decode selected $\tilde{A}_i$ without the need to decode a complete image.

All $\tilde{A}$ could be represented as in Case I and (124), repeated here for convenience, and now assuming a standard ordering $\tilde{A}_1, \tilde{A}_2, \tilde{A}_3, \ldots\ldots\ldots \tilde{A}_{16}$.

$$\underset{\substack{\uparrow \\ \text{START} \\ \text{OF IMAGE}}}{}\mathrm{PSI2D}[\tilde{A}_1]\underset{\substack{\uparrow \\ \text{START} \\ \text{OF } \tilde{A}_2}}{\cdot}\mathrm{PSI2D}[\tilde{A}_2]\underset{\substack{\uparrow \\ \text{START} \\ \text{OF } \tilde{A}_3}}{\cdot}\mathrm{PSI2D}[\tilde{A}_3]\cdot\ldots \qquad (128)$$

Now, each PSI2D[$\tilde{A}_i$] is variable length and embedded within the longer coded string of (128). The problem is that a decoder can only extract $\tilde{A}_j$ without the need to decode those $\tilde{A}_i$ in front of it, provided that it knows where PSI2D[$\tilde{A}_j$] begins (in the overall coded string of binary data in (128)). That is, without this specific knowledge, $\tilde{A}_1$, $\tilde{A}_2$, $\tilde{A}_{j-1}$ would have to be decoded first.

So, let[14]

$$\ell_j = \mathscr{L}(\mathrm{PSI2D}[\tilde{A}_j]) \tag{129}$$

and

$$\tilde{L}(\tilde{A}) = \ell_1\, \ell_2\, \ell_3 \ldots \tag{130}$$

Then, $\tilde{A}$ can be coded and stored serially as

$$\begin{tikzcd} L(\tilde{A}) \arrow[rrr, bend right] \arrow[rrrrr, bend right] \arrow[rrrrrrr, bend right] & \ast & \mathrm{PSI2D}[\tilde{A}_1] & \ast & \mathrm{PSI2D}[\tilde{A}_2] & \ast & \mathrm{PSI2D}[\tilde{A}_3] & \ast & \ldots \end{tikzcd} \tag{131}$$

or

$$\begin{tikzcd} \ell_1 \arrow[rrrr, bend left] & \ast & \mathrm{PSI2D}[\tilde{A}_1] & \ast & \ell_2 \arrow[rrrr, bend left] & \ast & \mathrm{PSI2D}[\tilde{A}_2] & \ast & \ell_3 \arrow[rrr, bend left] & \ast & \mathrm{PSI2D}[\tilde{A}_3] & \ast & \ldots \end{tikzcd} \tag{132}$$

In (131), $\tilde{L}(\tilde{A})$ is maintained separately as a table of pointers to the relative positions in memory where each $\tilde{A}_i$ begins. In (132), each ℓ_i is used to jump over each coded $\tilde{A}_i$ until the desired one is reached.[15] Any individual $\tilde{A}_i$ can now be quickly found and decoded without the need to decode any other $\tilde{A}_i$. Keep in mind that this example illustrates an approach that speeds the determination of the location of a variable-length string in a serial data stream. For the archive problem, the $\tilde{A}_i$ could just as easily be a one- or three-dimensional array. Additionally, exactly the same problem results when communication errors threaten to disrupt the decoding process.

[14]This ℓ_j has no relationship to the ℓ_j in Eq. 28.

[15]Note that in both cases, the ℓ_i would typically be reduced to word, or byte, pointers rather than bit pointers.

ID CODING

Consider again the coding of a sequence of data blocks, this time from a more general point of view. We'll get more specific later. As before, we take

$$\tilde{X} = \tilde{X}_1 * \tilde{X}_2 * \ldots \tilde{X}_m \qquad (133)$$

as the sequence of m blocks to be coded. Each block is to be coded by an N-option adaptive PSI11 coder, as defined in (21)–(24). We need not be too specific here.

The N code options for this PSI11 are given as

$$\text{PSI}\alpha_0, \text{PSI}\alpha_1, \ldots \text{PSI}\alpha_{N-1} \qquad (134)$$

so that the form of a coded $\tilde{X}_j$ is

$$\Psi_{11}[\tilde{X}_j] = ID(id_j) * \Psi_{\alpha_{id_j}}[\tilde{X}_j] \qquad (135)$$

where

$$0 \le id_j \le N-1 \qquad (136)$$

is the code identifier for block $\tilde{X}_j$ and

$$ID(id_j) \qquad (137)$$

is its standard $\lfloor \log_2 N \rfloor$ bit binary representation.

Now, define

$$\tilde{id} = id_1 * id_2 * \ldots id_m \qquad (138)$$

as a block of all the identifiers and

$$ID(\tilde{id}) = ID(id_1) * ID(id_2) * \ldots ID(id_m) \tag{139}$$

as the corresponding sequence of standard binary representations.

Rearranging the Coding of $\tilde{X}$

The basic form for coding all $\tilde{X}$ is simply[16]

$$\Psi_{11}[\tilde{X}_1] * \Psi_{11}[\tilde{X}_2] * \ldots \Psi_{11}[\tilde{X}_m] \tag{140}$$

or, expanding from (135),

$$ID(id_1) * \Psi_{\alpha_{id_1}}[\tilde{X}_1] * ID(id_2) * \Psi_{\alpha_{id_2}}[\tilde{X}_2] * \ldots ID(id_m) * \Psi_{\alpha_{id_m}}[\tilde{X}_m] \tag{141}$$

But this can be rearranged without jeopardizing the decoding process, and without any difference in performance as

$$ID(\tilde{id}) * \Psi_{\alpha_{id_1}}[\tilde{X}_1] * \Psi_{\alpha_{id_2}}[\tilde{X}_2] * \ldots \tag{142}$$

What have we gained? Thus far, we have only incurred an additional buffering requirement. However, it should now be clear that $ID(\tilde{id})$ represents the fixed-length coding of a sequence of symbols from a new data source – "the code identifiers." If the entropy of this new source is less than a fixed-length representation requires, there may be room for improvement. This suggests the coding structure shown in Fig. 26, where we have used SPLIT' to indicate the operation of separating a coded block's id_j and $PSI\alpha_{id_j}[\tilde{X}_j]$. We have named this structure PSI16 so that[17]

16From previous discussions, additional header information might be required (e.g., REF, $O(\tilde{A})$, etc.), depending on the actual coding process.

17Again, note that, for the sake of simplicity, we have left out the possible additional header information that may be required, such as REF, $O(\tilde{A})$, etc. In this case, information could be required for both the original data $\tilde{X}$ and the $\tilde{id}$.

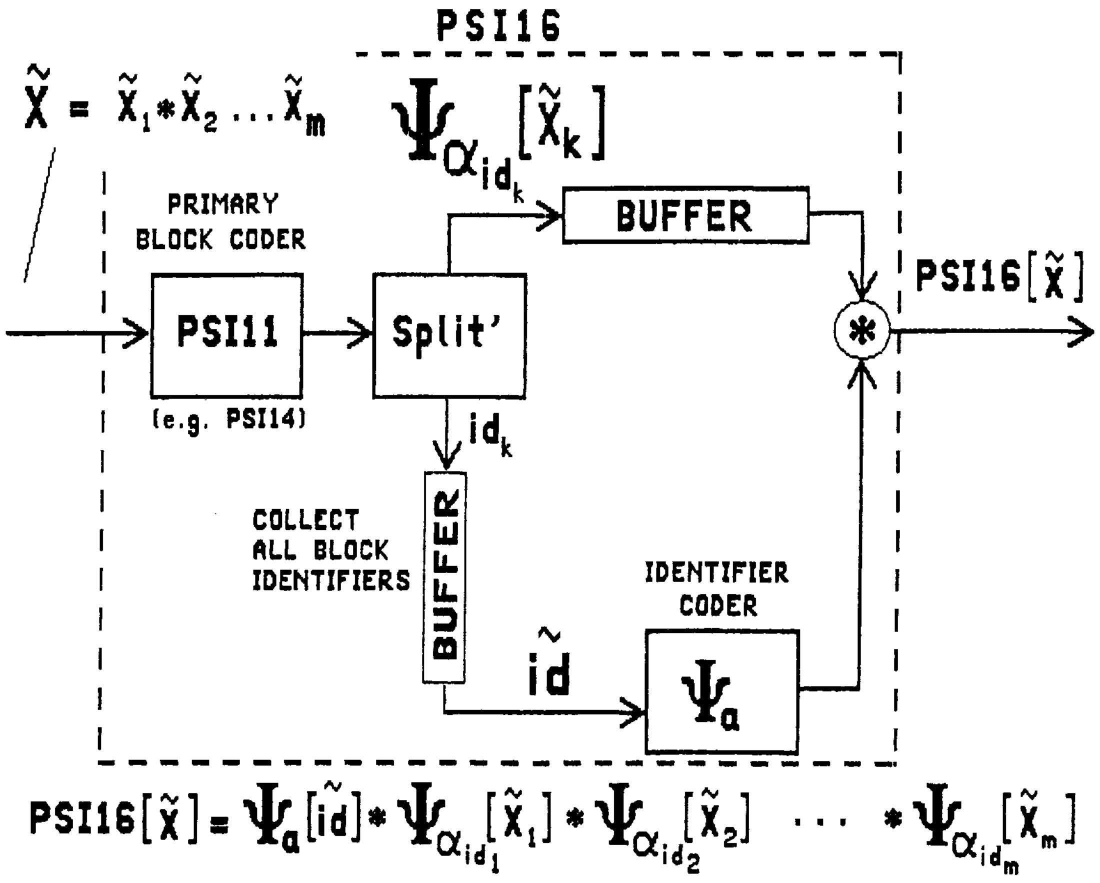

PSI16
$\tilde{X} = \tilde{X}_1 * \tilde{X}_2 \ldots \tilde{X}_m$
$\Psi_{\alpha_{id_k}}[\tilde{X}_k]$
PRIMARY BLOCK CODER
PSI11
(e.g. PSI14)
Split'
BUFFER
id_k
COLLECT ALL BLOCK IDENTIFIERS
BUFFER
$\tilde{id}$
IDENTIFIER CODER
Ψ_a
$PSI16[\tilde{X}]$
$PSI16[\tilde{X}] = \Psi_a[\tilde{id}] * \Psi_{\alpha_{id_1}}[\tilde{X}_1] * \Psi_{\alpha_{id_2}}[\tilde{X}_2] \cdots * \Psi_{\alpha_{id_m}}[\tilde{X}_m]$

Fig. 26. PSI16, Coding the Identifiers

$$\Psi_{16}[\tilde{X}] = \Psi_a[\tilde{id}] * \Psi_{\alpha_{id_1}}[\tilde{X}_1] * \ldots \Psi_{\alpha_{id_m}}[\tilde{X}_m] \tag{143}$$

where PSIa denotes the "to-be-determined" coder to be used to represent the m-sample identifier sequence, $\tilde{id}$. In the basic representation of (140), PSIa ≡ $\tilde{ID}[\cdot]$.

The Code Rate

Let

$$A = \sum_{j=1}^{m} \mathcal{L}(\Psi_{\alpha_{id_j}}[\tilde{X}_j]) \text{ bits} \tag{144}$$

and

$$B = \mathcal{L}(\text{PSIa}[\tilde{\text{id}}]) \text{ bits} \tag{145}$$

Then, the average code rate for PSI16 is

$$\Re = \frac{E\{\mathcal{L}(\Psi_{16}[\tilde{X}])\}}{\mathcal{L}(\tilde{X})} = \frac{E\{A\}}{\mathcal{L}(\tilde{X})} + \frac{E\{B\}}{\mathcal{L}(\tilde{X})} \text{ bits/sample} \tag{146}$$

Without loss of generality we can assume that for each block, both block size and number of code options used for each block are the same and set to J and N, respectively. Then we have

$$\mathcal{L}(\tilde{X}) = mJ = \text{fixed} \tag{147}$$

Now, in considering what happens as N and J are varied, we will assume that m (the number of blocks in $\tilde{X}$) is adjusted to maintain the length of $\tilde{X}$ at a fixed value.

Contributions From (A). The statistical term (denominator fixed)

$$\frac{E\{A\}}{mJ} \tag{148}$$

is a function of N, J, and the characteristics of the data. Clearly, we can say that

1) Larger N means that there are more code options to choose from over a given block size J. Thus, this term would tend to decrease with increasing N. Of course, real improvements will only occur if the added options are actually used (i.e., that they are chosen to be used because they provide better performance than other options).

2) As J is decreased, the decision to use one of N code options is made more often. The coder is thus able to respond more rapidly to changes in data entropy and can thus code more efficiently. Since the penalty of added code identifier bits is not included in E{A}, this term will tend to decrease with decreasing J.

We cannot be more precise in evaluating E{A} without getting considerably more specific.

Code Identifiers. Now, consider the second term, which corresponds to the contribution from code identifiers

$$\frac{E\{B\}}{mJ} \tag{149}$$

If we assume a standard fixed-length binary representation, as in (140), we have

$$B = E\{B\} = m\lfloor \log_2 N \rfloor \text{ bits}$$

so that (147) is reduced to the nonstatistical

$$\alpha_F(N,J) = \frac{\lfloor \log_2 N \rfloor}{J} \text{ bits/sample} \tag{150}$$

α_F is plotted in Fig. 27 for various N and J of interest. Because of a potential reduction in the code-rate penalty implied by α_F, the structure in Fig. 26 was devised.

The total code rate in (146) becomes

$$\Re_F(N,J) = \frac{E\{A\}}{\mathcal{L}(\tilde{X})} + \alpha_F(N,J) \tag{151}$$

Note that α_F increases with N and decreases with J, just the opposite of the "tendencies" exhibited by the first term in (146) and (151). To investigate this any further we must be more specific.

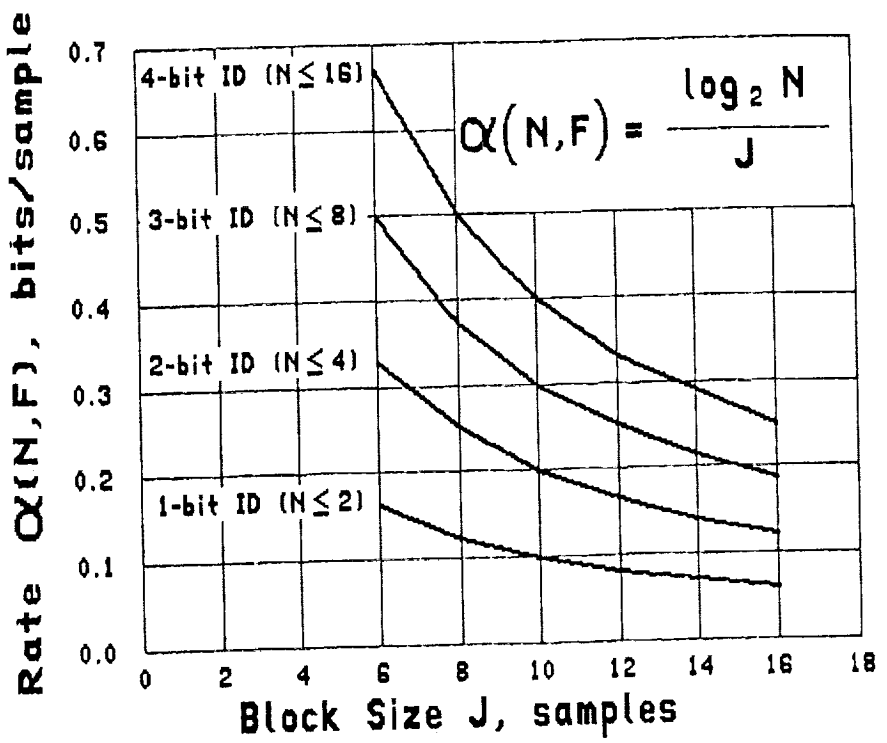


Fig. 27. Code Identifier Code Rate, Fixed-Length Representation

Application to PSI14,K+

Now, let the primary block coder in Fig. 26 be PSI14,K+.

Fixed-Length Identifiers. For most practical applications (e.g., imaging), simulations will show that $\Re_F(N,J)$ in Eq. 151 behaves something like the graph in Fig. 28. That is, from an overall performance point of view, the choice of block size J is not critical, provided that it is not too small and not too large. Since there is generally an implementation advantage to smaller block sizes, most applications to date have used a convenient J = 16 (power of 2) near the lower end of this range of "best" performance. Thus, in many situations, variations in the two terms in Eq. 151 cancel each other out.

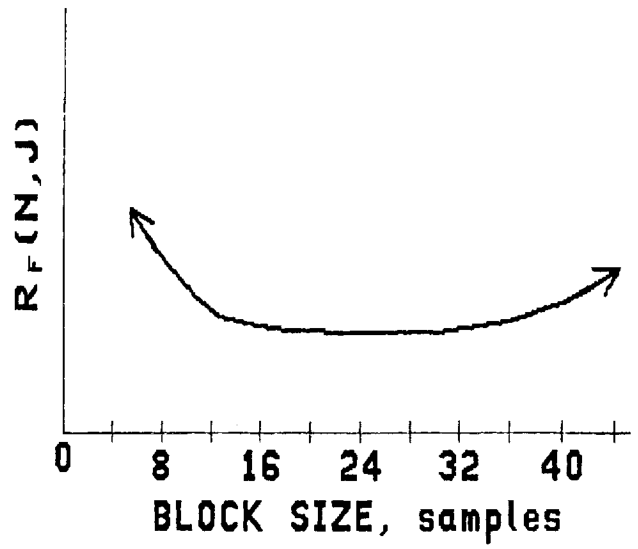


Fig. 28. $\Re_F(N,J)$ for a PSI14,K+ Primary Coder

Coding the identifiers. Now, fix the PSI14,K+ structure in Fig. 26 and consider situations where the coding of identifiers might be beneficial and for which we can identify PSIa.

For illustration, let

$$\begin{aligned} m &= 16 \\ J &= 16 \end{aligned} \tag{152}$$

(a realistic assumption). This means that each $\tilde{X}$ in (131) is a sequence of 16 blocks of 16 samples each. Additionally, assume that the number of available code options is

$$9 \leq N \leq 16 \tag{153}$$

By Fig. 27 or Eq. 150, the penalty for a fixed-length code identifier representation is 0.25 bit/sample (or 4 bits per identifier). For our example, this is the most we can expect to reduce the overall code rate by prescribing a different PSIa.

The desired form for PSIa in Eq. 143 and Fig. 26 becomes readily apparent by drawing an analogy from the plots of entropy in Figs. 29 and 30.

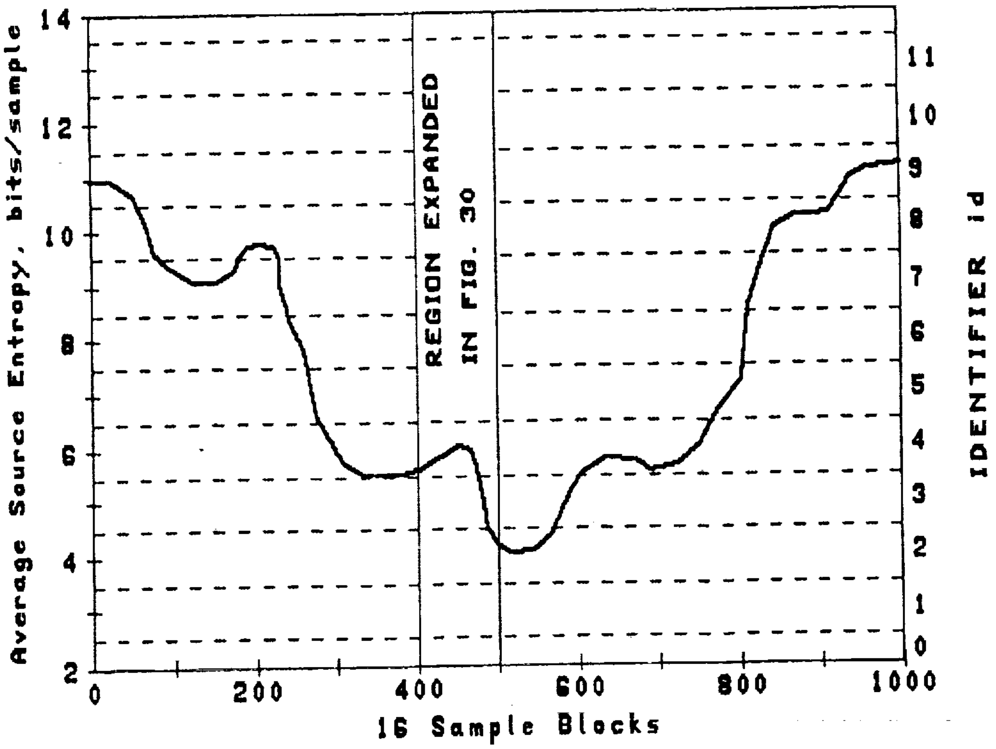


Fig. 29. Hypothetical Entropy Graph

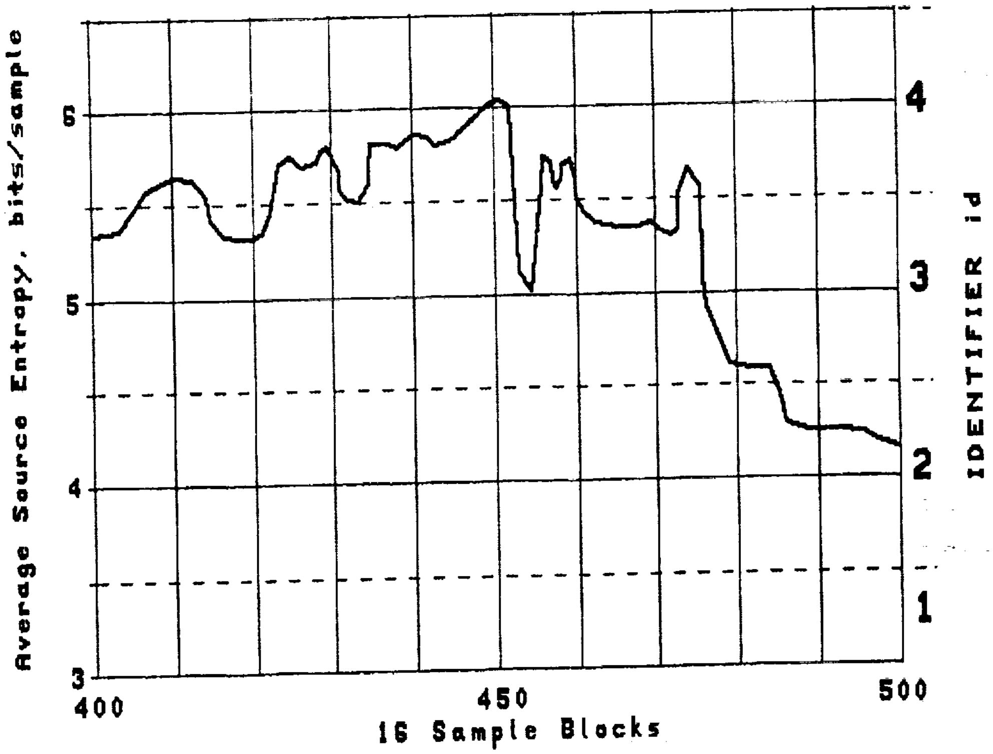


Fig. 30. Expanded Hypothetical Entropy Graph

The figure shows a hypothetical, but not unrealistic, graph of average source entropy plotted for a series of J = 16 sample blocks. The slowly varying entropy function is shown for 1,000 such blocks in Fig. 29, and then Fig. 30 expands the region for blocks 400–500.

Note that these graphs look much like the waveforms generated by typical data sources to which PSI14,K+ would generally apply. In fact, this is the key to specifying an appropriate practical form for PSIa.

Recall how PSI14K+ codes a block of sampled data. It compares the performance of a set of individual code options (repeated here from (73) where, without loss of generality we'll assume K = 0, since we're focusing on codes with many options):

$$\Psi_{i(\lambda+id),\ k(\lambda+id)} \tag{154}$$

for $\lambda \geq 0$ and $0 \leq id \leq N - 2$ and

$$\Psi_3 \text{ for } id = N - 1 \tag{155}$$

(where λ sets the starting option and id is the code identifier for a given option) and chooses to use the best one, say with code identifier[18]

$$id = id+ \tag{156}$$

It is now important to recall the performance characteristics for each option as the identifier number is increased. Except for the first option and those with very large id values, each individual option exhibits a Dynamic Range of efficient performance of about 1 bit/sample. These 1-bit Dynamic Ranges are both adjacent and non-overlapping [11]. Thus, the Dynamic Range for the option with id = ℓ sits 1 bit below the Dynamic Range of the option corresponding to id = ℓ + 1. This is shown more

[18]For example, with λ = 1 and N = 16 the codes in order of increasing id are PSI1, PSI1,1, PSI1,2 . . . PSI1,15, PSI3 for a 16-option coder, and corresponding to PSIss in Ref [7].

explicitly in Fig. 30 where, with $\lambda = 1$ in (154), the code identifiers for code options PSI1,id+1 are shown adjacent to their corresponding Dynamic Ranges.

Code option PSI1,id+1 will be chosen most frequently when the source entropy lies within its 1-bit Dynamic Range ($id+1.5 \leq \overline{H}_\delta \leq id+2.5$). For example, in the graph of Fig. 30, the identifier would tend to switch between id = 3 and id = 4 over blocks 400–425. Then, id = 4 would predominate for the next 25 blocks as average entropy rises. And so on.

Thus, we can interpret the generation of code identifiers as a "sampling" of the entropy waveform with a quantization accuracy of 1 bit of entropy per quantization interval (i.e., each code option). Hence, the data source represented by a sequence of code identifiers behaves much like the data sources that PSI14,K+ itself was designed to code efficiently. We should be able to follow a similar approach here.

In fact, for the example in (152) and (153) we should be able to use a properly configured PSI14,K+.

Recall that the "+" in "PSI14,K+" refers to those steps that precede the actual coding of $\tilde{\delta}^n$ sequences by adaptive variable-length coding. In general, those steps could be arbitrary. In fact, the definition of a PSI14,K+ "module" presumes that pre-processing can be external and, therefore, arbitrary. However, we can be more specific for the identifier coding problem. The same kind of predictive preprocessing considerations apply to the coding of identifiers as to more typical forms of data. For now it suffices to note that the simple PSI14,K+ built-in one-dimensional predictor (or external predictor) and mapper from Chapter II directly apply here. We will return to this issue momentarily.

Then, assuming that a preprocessor has been specified, let[19]

$$\text{PSIa} = \text{PSI14,0+}$$

with parameters n_a, J_a, λ_a, and N_a given as

[19]Or the original PSI4 would also cover the required range of "id entropy."

n_a = Number of input quantization bits = $\lfloor \log_2 N \rfloor$ = 4

J_a = Block length = m = 16

λ_a = Starting Option Parameter for PSI0 = 0

N_a = Number of options = 4

Thus, the code options are from (154) to (155)

$$\text{PSI0, PSI1, PSI1,1, and PSI3} \tag{157}$$

That is, more simply,

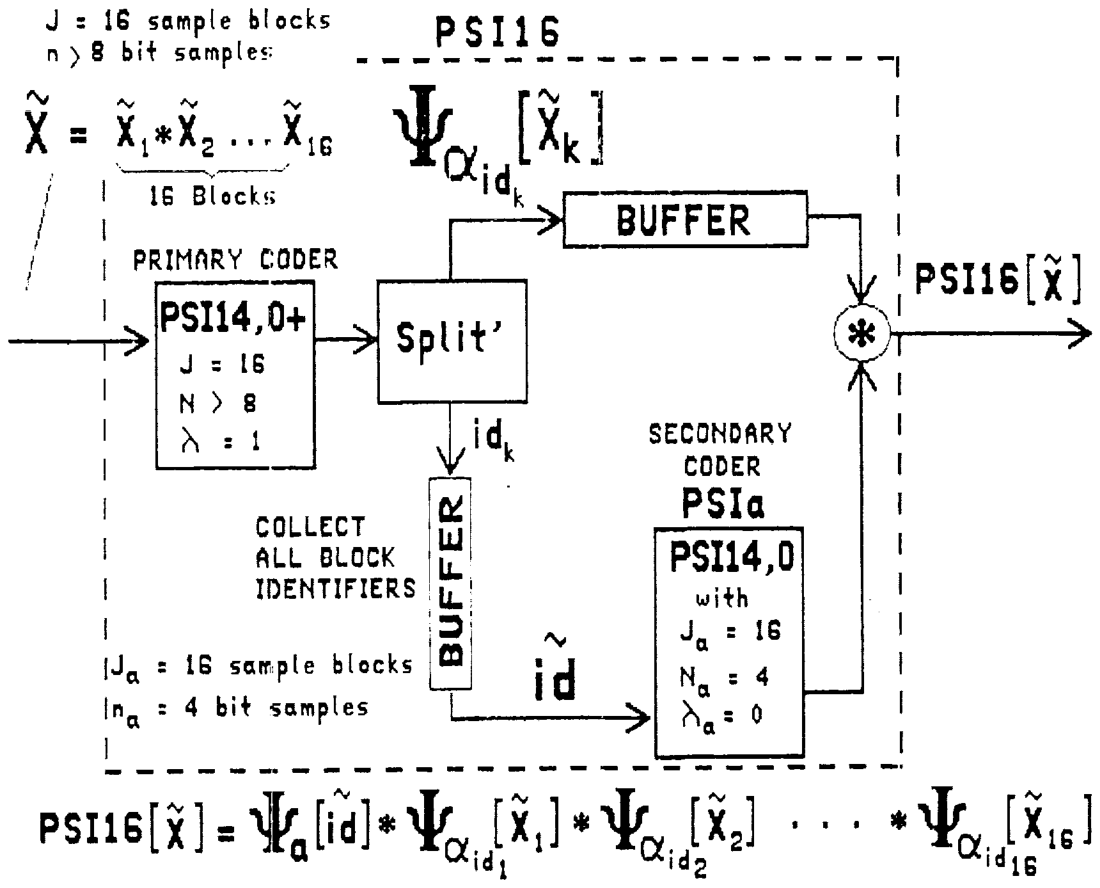
$$\text{PSIa} \equiv \text{PSI14+ with } \dot{\lambda} = 0 \tag{158}$$

The source of identifiers that we are now coding takes on values

$$id = 0, 1, 2, \ldots, 14, 15 \tag{159}$$

with the value of one sample distributed about the value of an adjacent sample (with adjustments near the boundary values of 0 and N–1). We have seen this before. The one-dimensional predictor in (14) and the specialized mapping of (18)–(19) directly apply without modification.

The typical performance for the 4-option coder should adequately cover the 1–4 bit entropy range, as shown in Fig. 16 (although the latter graph was obtained from data with a much larger number of quantization levels). If much lower identifier entropies can be anticipated, other algorithms could be applied (e.g., PSI9 in Ref. 3, or run-length coding). For this example, the coder for PSI16 in Fig. 26 reduces to that shown in Fig. 31.



$$PSI16[\tilde{X}] = \Psi_a[\tilde{id}] * \Psi_{\alpha_{id_1}}[\tilde{X}_1] * \Psi_{\alpha_{id_2}}[\tilde{X}_2] \quad \cdots \quad * \Psi_{\alpha_{id_{16}}}[\tilde{X}_{16}]$$

Fig. 31. PSI16 Coder of Identifiers Example

As already noted, the difference in performance between this coder and that provided by a standard PSI14,K+ representation is totally in the penalty imposed by the identifier bits. For the fixed-length representation, this penalty is $\alpha_F(N,J)$ in Eq. 150 and Fig. 27, or 0.25 bit/sample with N > 8 and J = 16, as in this example.

Under the worst-case conditions, the PSI14,0 coding of $\tilde{id}$ in Fig. 31 doesn't work at all, so that PSI3 is used on all identifier blocks. But PSI3 is equivalent to the fixed-length representation. Thus, the total penalty for identifier representation is at most $\alpha_F(N,J)$ plus the penalty to specify which code option was used to represent the identifier blocks themselves.

Assuming the 4-option PSI14,0 in Fig. 31, this penalty cannot be larger than 2 bits per $\tilde{X}$ sequence (from which a block of identifiers is extracted). This computes to $1/128 \leq 0.01$ bit/sample.

But note that actual conditions may be quite different from these worst-case conditions. If the entropy is gradually changing, as in the hypothesized plots of Figs. 29 and 30, the identifier predictions would produce small errors so that code rates between 1 and 2 bits/identifier might be a reasonable expectation. This would reduce the identifier penalty (and hence the overall average code rate) to 0.0625 to 0.125 bit/sample. In the limiting case (all PSI0 occurrences – same code used for all primary blocks), the identifier cost would drop to 0.03 bit/sample. Overall, this says that such identifier coding **might pick up as much as 0.2 bit/sample when stationary conditions prevail, but would not lose anything significant when data characteristics are rapidly changing.**

These gains would double if a primary block size of $J = 8$ were used so that the fixed identifier penalty for an $N > 8$ option PSI14,K+ is 0.5 bit/sample. Thus, identifier coding might be most appropriate for situations in which the source is sometimes rapidly changing (where smaller block sizes can be advantageous) and at other times quite stationary.

How General? It should be clear that the primary and secondary coding steps in Figs. 26 and 31 are independent of each other. That is, a primary PSI11 (or PSI14,K+) coder that operates on an input $\tilde{X}_j$ does not need to know about the later coding of identifiers by PSIa (or the reduced PSI14) and vice versa. Thus, for example, identifier coding can be done external to a VLSI module that performs the primary coding.

Because of this independence, one can look at the identifier coding problem as more than a one-dimensional problem, as we did earlier for the primary data stream. For example, the identifiers can be viewed as a two-dimensional array, just as in Figs. 2, 22 and 24, where two-dimensional predictors were considered. Certainly for imaging applications, a prediction of the identifier (entropy) used when coding a particular block could be positively influenced by knowledge of the identifier (entropy) used by the block above it. In fact, the block above offers more pertinent prediction information than an adjacent block along the same line (the block above is located in the same spatial region).

Iterative PSI16. While not of practical value for the real data sources normally encountered, it is worthwhile to note that PSIa in Fig. 26 could be interpreted to be another PSI16 (with different parameters). By doing so, it means that the identifiers of the coded primary identifiers would be coded also. And so on. A similar iterative structure results from the binary coding algorithms in Ref. 3.

RATE CONTROL

Not all applications have the luxury of a data system that can absorb the naturally occurring variations in output rate from a noiseless coder. Often there is a physical constraint that sets a fixed number of bits that can be used to represent blocks of data. This may come in the form of memory restrictions and/or transfer-rate limitations.

The case of primary interest here is one in which the number of bits available to represent data blocks is close to what is necessary to represent those blocks efficiently (close to the entropy) by using noiseless coding. If the number of bits available is close, but not sufficient, some loss in data fidelity must occur. But this loss should be minor since almost enough bits are available (for a perfect reconstruction). The resulting data reconstruction should be imperfect but "near-noiseless."

In this section, we will first look at typical rate or buffer restrictions placed on imaging science applications. We will then introduce simple modifications to the coding process that should provide a practical measure of control on how this near-noiseless error will occur. We later expand this approach to the "rate control" of the multispectral HIRIS instrument.

The FIFO Buffer Constraint

The general rate-constraint problem we are addressing is ultimately one in which a fixed, and presumably limited, number of bits can be used to represent a certain span of data. Such a constraint can arise from many factors but often materializes in the form of a buffer that cannot hold all the data generated for the span of data. We will tie much of our future discussion to, and obtain motivation from, the constraints imposed by the assumption of "First-In First-Out" (FIFO) buffers, as described below.

The concept of a FIFO Buffer as the vehicle for a rate constraint is shown in Fig. 32.

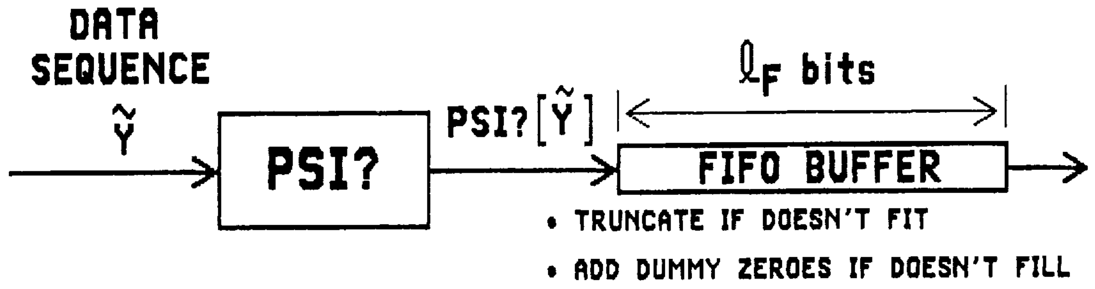


Fig. 32. The FIFO Buffer

Here, a data sequence $\tilde{Y}$ (which could be an image line, a single data block, a complete image, etc.) is coded by a noiseless coder

$$\text{PSI?} \tag{160}$$

which we have intentionally left arbitrary since the details of coding are immaterial at this point. The coded result PSI?[$\tilde{Y}$] is passed into a FIFO Buffer of length ℓ_F bits. Then, the basic assumption is:

$$\text{Exactly } \ell_F \text{ bits will be communicated for } \tilde{Y} \tag{161}$$

This means that:

1) If $\mathcal{L}$(PSI?[$\tilde{Y}$]) > ℓ_F, all bits of PSI?[$\tilde{Y}$] after the first ℓ_F will be truncated. (162)

2) If $\mathcal{L}$(PSI?[$\tilde{Y}$]) < ℓ_F, dummy bits will be concatenated to the end of PSI?[$\tilde{Y}$] to make the total length equal to ℓ_F. (163)

Standard Image Format. Recall the familiar standard format for the encoding of images illustrated in Fig. 33. Lines are coded top-to-bottom and left-to-right. Recall that earlier notation – $\tilde{X}(j)$ represents the jth line of a T-line image, with samples running left-to-right.

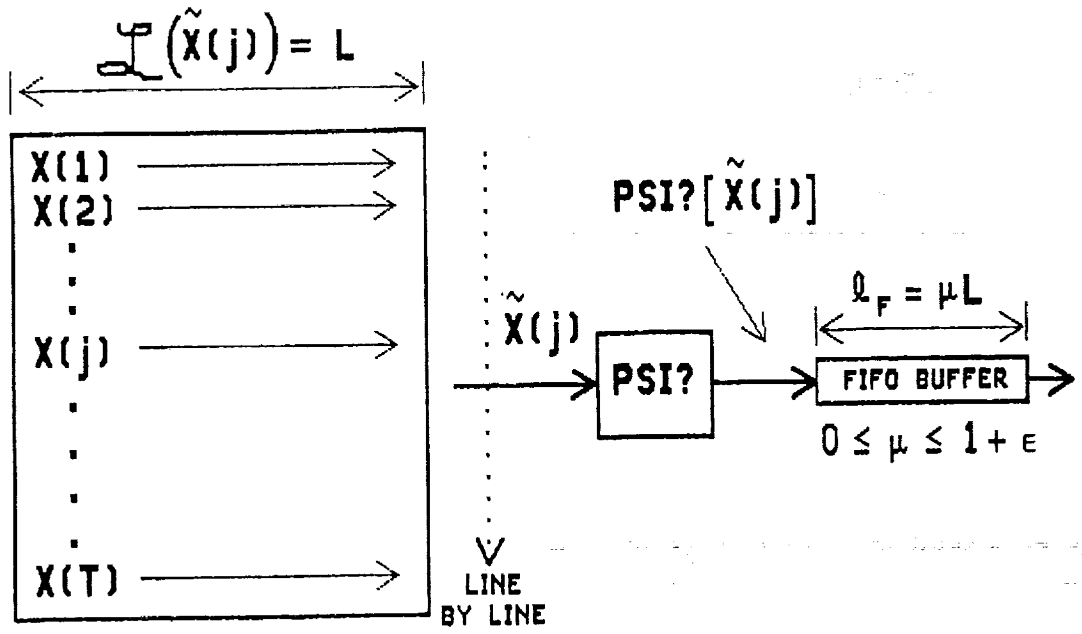


Fig. 33. Standard Format Line Coding and Buffering

Lines are coded one at a time, again by using arbitrary noiseless coder

PSI?

as in Fig. 32 and (160).

The coded result, PSI?$[\tilde{X}(j)]$ for the jth line, is shown being placed in a FIFO Buffer of length

$$\ell_F = \mu L \text{ bits} \tag{164}$$

where

$$L = \mathcal{L}(\tilde{X}(j)) \text{ bits} \tag{165}$$

is the length of an uncoded line, and

$$0 \le \mu \le 1 + \varepsilon \tag{166}$$

Of course, the actual lengths of these coded lines will vary. If $\mu = 1 + \varepsilon$, the buffer can hold any line produced, including an uncoded line (we assume that PSI? incorporates a PSI3 option and that the ε is very small, but large enough to cover any identifier overhead).

By assumption, in (161)–(163), the buffer length defines the rate constraint on a "line." That is, the line must be communicated with precisely μL bits, regardless of the number required or used by PSI?.[20]

Lines that use less than this quantity must be supplemented with dummy zeroes, as in

$$\overbrace{\underbrace{\text{PSI?}[\tilde{X}(j)]}_{\substack{\text{CODED LINE}\\ \text{(SHORTER THAN BUFFER)}}} * \underbrace{0\,0\,0\ldots0\,0\,0}_{\text{DUMMY ZEROES}}}^{\ell_F = \mu L \text{ BITS}} \tag{167}$$

The penalty here is, of course, efficiency.

Longest Line. If μ is decreased (shortening the buffer), eventually it will decrease the buffer size to the length of the longest coded line.

Let

$$\mu_{max} \tag{168}$$

be the value when this happens. Then, at this point, all lines can still be communicated error-free. If ℓ_F is smaller than $\mu_{max}L$, the ends of some lines must be truncated to meet this constraint (see Fig. 32).

[20]Such a constraint could arise in many ways, but its effect is easiest to visualize with this buffer.

Average line. The average line length generated by PSI? for all lines in an image is given as

$$\Lambda = \frac{1}{T}\sum_{j} \mathcal{L}(\text{PSI?}[\tilde{X}(j)]) \text{ bits} \tag{169}$$

If PSI? incorporates PSI14,K+ we know that Λ would typically be close to a differential entropy measurement times the number of samples in a line.

Suppose that instead of a single line buffer, we used a full-image FIFO buffer to store all coded lines before communication. Then TΛ gives the minimum-length buffer that would allow error-free coding (see 161–163). But if we apply this same rate constraint over a single line, some lines will be truncated since, in general,

$$\Lambda < \mu_{max} L \tag{170}$$

Thus, we immediately see the general advantage of larger buffers.

Visualizing Truncation. To visualize what happens, let the buffer size be the average line length, Λ. Coded line lengths of a single image will tend to be distributed about Λ, although the shape may not be as perfect as illustrated in Fig. 34. But this depiction will suffice here. Lines that are longer than Λ will be truncated by an amount

$$\mathcal{L}(\text{PSI?}[\tilde{X}(j)]) - \Lambda \text{ bits} \tag{171}$$

and shorter lines will be increased by the addition of

$$\Lambda - \mathcal{L}(\text{PSI?}[\tilde{X}(j)]) \tag{172}$$

dummy zeroes as in (167).

The bit truncation that occurs will ultimately truncate samples at the ends of lines, as shown in Fig. 35.

Actual image examples of this phenomenon can be found in Ref. 5. Note that because the characteristics of adjacent lines tend to be correlated, so too is the magnitude of truncation that occurs from line to line.

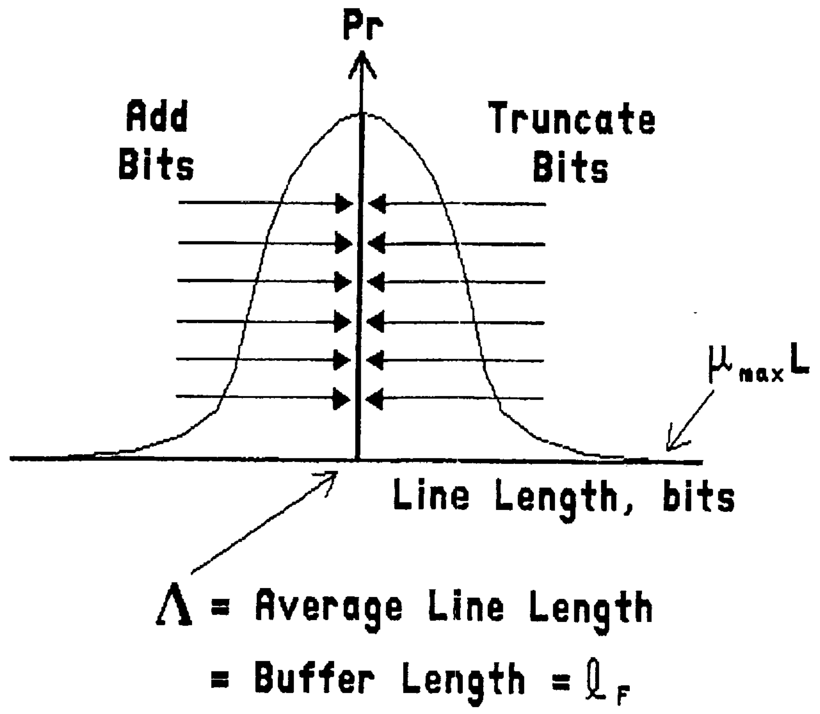


Fig. 34. Example: Line-Length Distributions for Single Image, Buffer Length = Average Line Length

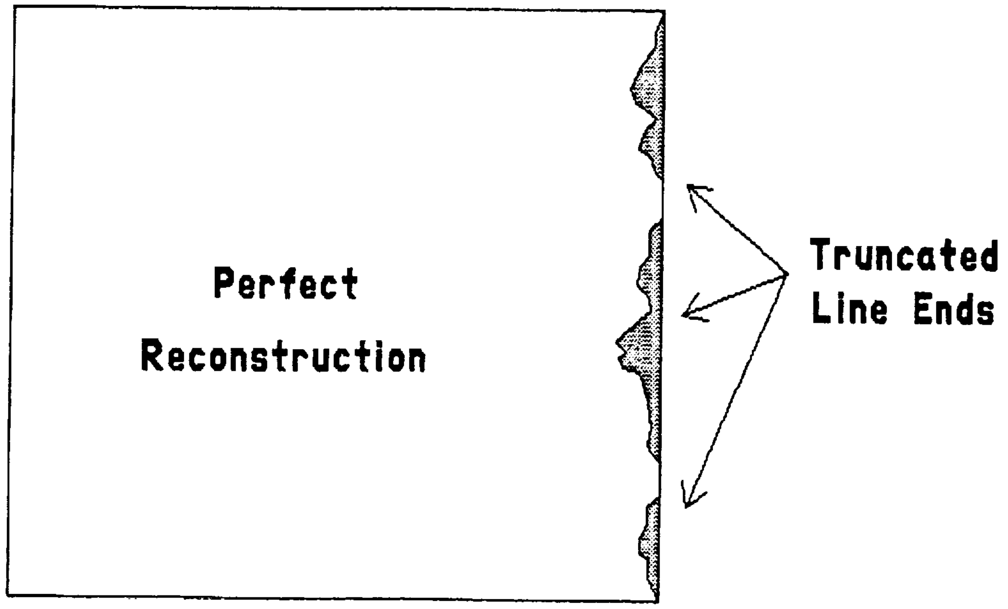


Fig. 35. Truncated Image Lines

Buffer Size and Error.The previous example was only intended to indicate approximately how truncation errors might occur. In reality, depending on the application, Λ and the distribution of line lengths about it are often not well known and may vary significantly from image to image as different types of scenes are encountered.

As an example, the HIRIS is a new experimental instrument with image-like characteristics that has not actually been built yet [14]. Only estimates of instrument source characteristics are available to guide the choice of buffer sizes and estimate the impact of rate control. The instrument has 192 spectral bands, 12 bits/sample quantization, with each band acting like a separate imaging system. Data derived from similar instruments have shown that band-to-band entropies could vary by as much as 6 bits/sample, depending on the prediction method used. This could mean a lot of fill bits for the lower entropy bands if a single fixed-line buffer length must apply to all bands (in order to minimize the truncations of higher entropy bands). The HIRIS instrument data problem is looked at more closely in a later section.

FIFO Buffer Priority. While adjustments to line-buffer sizes can affect the amount of truncation that occurs (at the expense of efficiency), we have so far ignored the possibility of altering the way truncation errors (e.g., Fig. 35) occur to a more acceptable form.

Another way to look at the truncation error that occurs in Fig. 35 is to note the following: Since damage occurs on the right side of the picture, the picture elements (samples) on the left side have "more priority" for the limited bits allowed per line than do the samples on the right side. When the bits run out, the samples with the least priority are deleted.

Alternate Lines. Now, alternate this priority from right-to-left on odd lines to left-to-right on even lines. By this we mean that alternate lines are coded and loaded into the FIFO buffer in opposite directions. The result is shown in Fig. 36. ODD lines are complete on the left side, and EVEN lines are complete on the right side. But truncations occur on BOTH sides of the image.

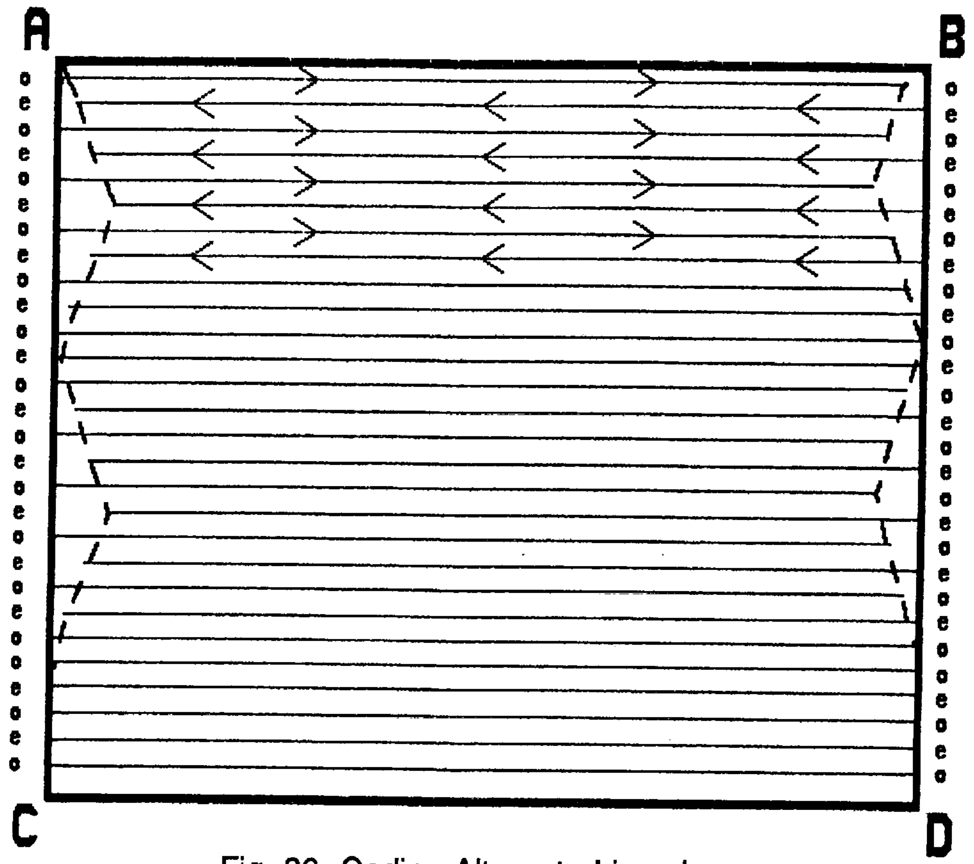


Fig. 36 Coding Alternate Lines in Opposite Directions

The figure shows even and odd lines being coded in opposite directions. (The scale does not allow all lines to be visible.) As shown, the region of line truncation on the right side is the same as in Fig. 35, but every other line (the even) is now without error. Similarly, the region of line truncation on the left side is (approximately) a mirror image of the region on the right side, where now the odd lines are without error. A blowup of the upper left-hand corner of Fig. 36 is shown in Fig. 37.

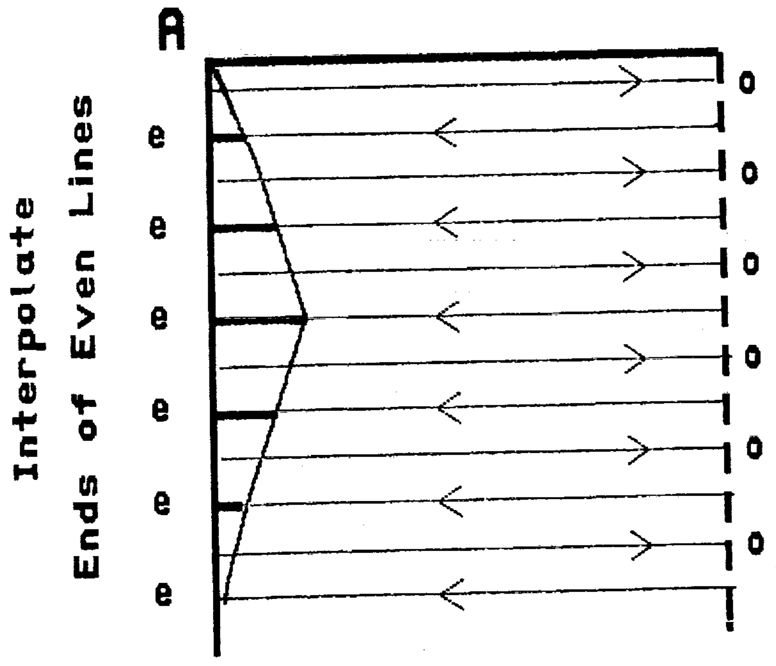


Fig. 37. A Blowup of the Upper Left-Hand Corner of Fig. 36

Since alternate lines are without error (on either side of the image), the missing pixels of truncated lines can be "filled in" by interpolation. The net result of this approach is that the missing data in Fig. 34 are traded for a loss of resolution on both edges of the image. This procedure was implemented for the Voyager 2 encounters of Uranus and Neptune.

Note that this prioritizing does not prevent the use of two-dimensional prediction discussed in an earlier section because when the coding of one line begins, the reproducible length of the previous line is known. This defines those samples that can be used to aid in prediction since a decoder will also be able to reconstruct them.

Prioritized LSBS. We will supplement the alternating left-right priorities described above by adding new conditions involving the least-significant bits of each sample in a line.

The concept we present here is quite simple, though the notation could easily grow into unmanageable proportions. To minimize this possibility, we will use a more specific limited example and extend earlier notation.

To do this we modify and add to Fig. 19, as shown in Fig. 38. Here, as we have been assuming, $\tilde{X}(j)$ is now a complete line of n-bit data instead of a single block. The Split-Sample parameter ℓ is set to $\ell = 2$ so that $\tilde{L}_2^x$ is a full line of the two least-significant bits of each sample of $\tilde{X}(j)$, and $\tilde{M}_x^{n,2}$ is a full line of the n-2 most-significant bit samples of $\tilde{X}(j)$.

For the coding of $\tilde{M}_x^{n,2}$, we use a line coder of the form PSI15,K+, as described in Eq. 110.

Next, we perform another reversible operation on the 2-bit lsb samples of $\tilde{L}_2^x$. First, note that even these two-bit samples exhibit significance. The operation $SS^{2,1}$ from (51)–(54) strips off the least-significant bits of each two-bit sample and places them in the sequence $\tilde{A}_0$ (note that these are also all the lsbs of $\tilde{X}(j)$). The remaining bits of each $\tilde{L}_2^x$ (which are the bits of next significance in samples of $\tilde{X}(j)$) are placed in $\tilde{A}_1$.

The sequences PSI15,K+[$\tilde{M}_x^{n,2}$], $\tilde{A}_1$, and $\tilde{A}_2$ are concatenated to produce the coded line $\tilde{X}(j)$ as

$$PSI?[\tilde{X}(j)] = PSI15,K+[\tilde{M}_x^{n,2}] * \tilde{A}_1 * \tilde{A}_0 \tag{173}$$

By the arguments describing Fig. 19, we know that this coder will perform almost identically to one that codes all the samples of $\tilde{X}(j)$ directly as

$$PSI15,(K+2)+[\tilde{X}(j)] \tag{174}$$

provided that the entropies of the $\tilde{M}_x^{n,2}$ sequences are high enough.

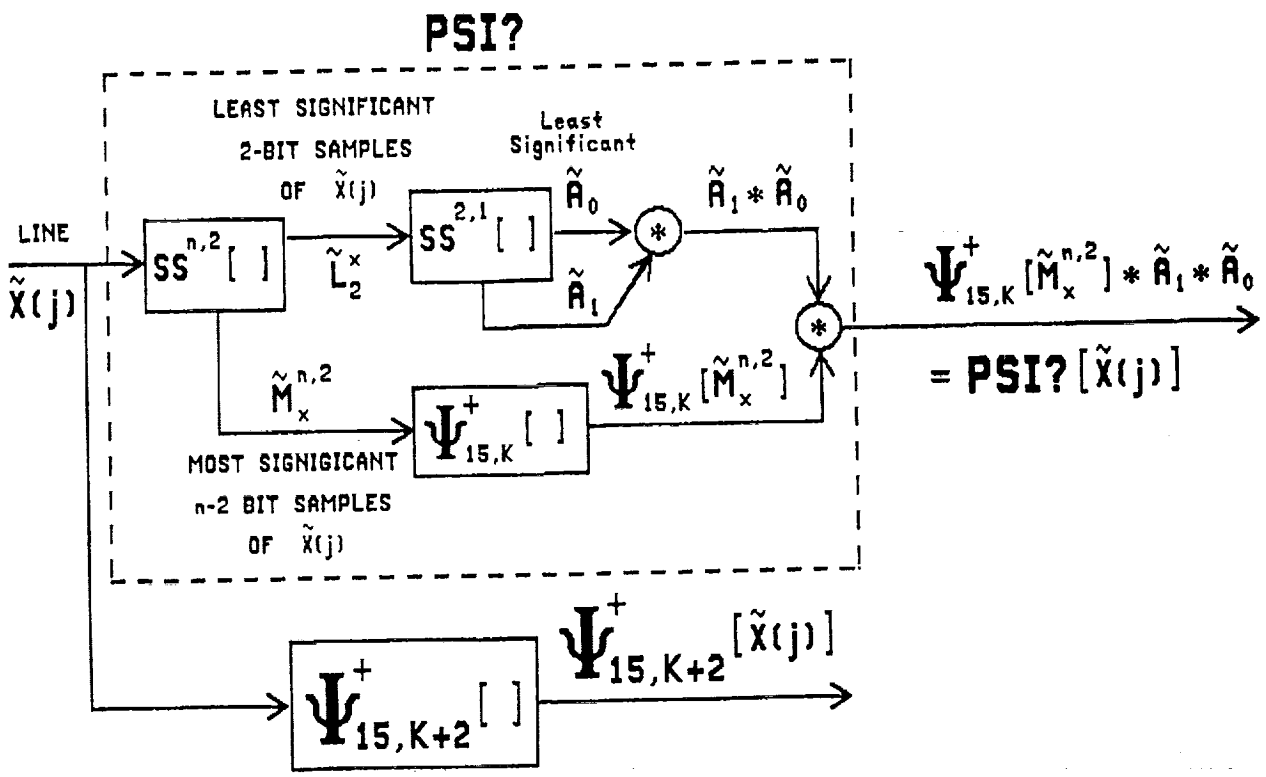

PSI?
LEAST SIGNIFICANT 2-BIT SAMPLES OF $\tilde{X}(j)$
Least Significant
LINE
$\tilde{X}(j)$
$SS^{n,2}[\ ]$
$\tilde{L}_2^x$
$SS^{2,1}[\ ]$
$\tilde{A}_0$
$\tilde{A}_1$
$\tilde{A}_1 * \tilde{A}_0$
$\tilde{M}_x^{n,2}$
$\Psi^+_{15,K}[\ ]$
$\Psi^+_{15,K}[\tilde{M}_x^{n,2}]$
MOST SIGNIGICANT n-2 BIT SAMPLES OF $\tilde{X}(j)$
$\Psi^+_{15,K}[\tilde{M}_x^{n,2}] * \tilde{A}_1 * \tilde{A}_0$
$= PSI?[\tilde{X}(j)]$
$\Psi^+_{15,K+2}[\ ]$
$\Psi^+_{15,K+2}[\tilde{X}(j)]$

Fig. 38. Arranging for Bit Priority

As described earlier, the initial "split" of the two lsbs off the input data shifts the effective Dynamic Range for the coder used to represent $\tilde{M}_x^{n,2}$ upward by 2 bits/sample. That is, if the lower end of the Dynamic Range for the $\tilde{M}_x^{n,2}$ coder is at 2.5 bits/sample, the lower end of the Dynamic Range for PSI? in Fig. 38 and Eq. 173 will be at 4.5 bits/sample. Thus, if entropies lower than this shifted Dynamic Range are not expected for an application, there is no performance penalty. This can be expected to be the case for many new instruments which have upward of n =12 bit quantization. The low-end entropies can be 5 or 6 bits/sample. And if we take K = 0 (and λ = 1) for PSI15,K+ in (173), the lower end of efficient performance is at an entropy of only

$$\bar{H}_\delta = 0.5 + 1 + 2 = 3.5 \text{ bits/sample}$$

well below the minimum expected data entropies just noted.

Now, consider what this "pre-splitting" does for rate control, assuming the same FIFO Line Buffer constraints as depicted in Figs. 32 and 33. With the arrangement in Fig. 38, the input to the FIFO Buffer of length $\ell_F = \mu L$ in Fig. 33 is the coded sequence PSI?[$\tilde{X}(j)$] in Eq. 173.

Figure 34 is expanded and modified in Fig. 39 to help see what happens as the result of this new PSI? and the FIFO buffer limitations. The abscissa has now been normalized to represent average bits/sample instead of total bits. Again we assume a buffer length equal to the average coded line length, which has been normalized in the figure to $\Lambda' = \Lambda/(\#$ samples per line $= L/n)$ bits/sample. The source data are presumed to be n-bit data so that the maximum **normalized** line length is n μ_{max} bits/sample. For discussion purposes, the distribution of normalized line lengths around the average is shown.

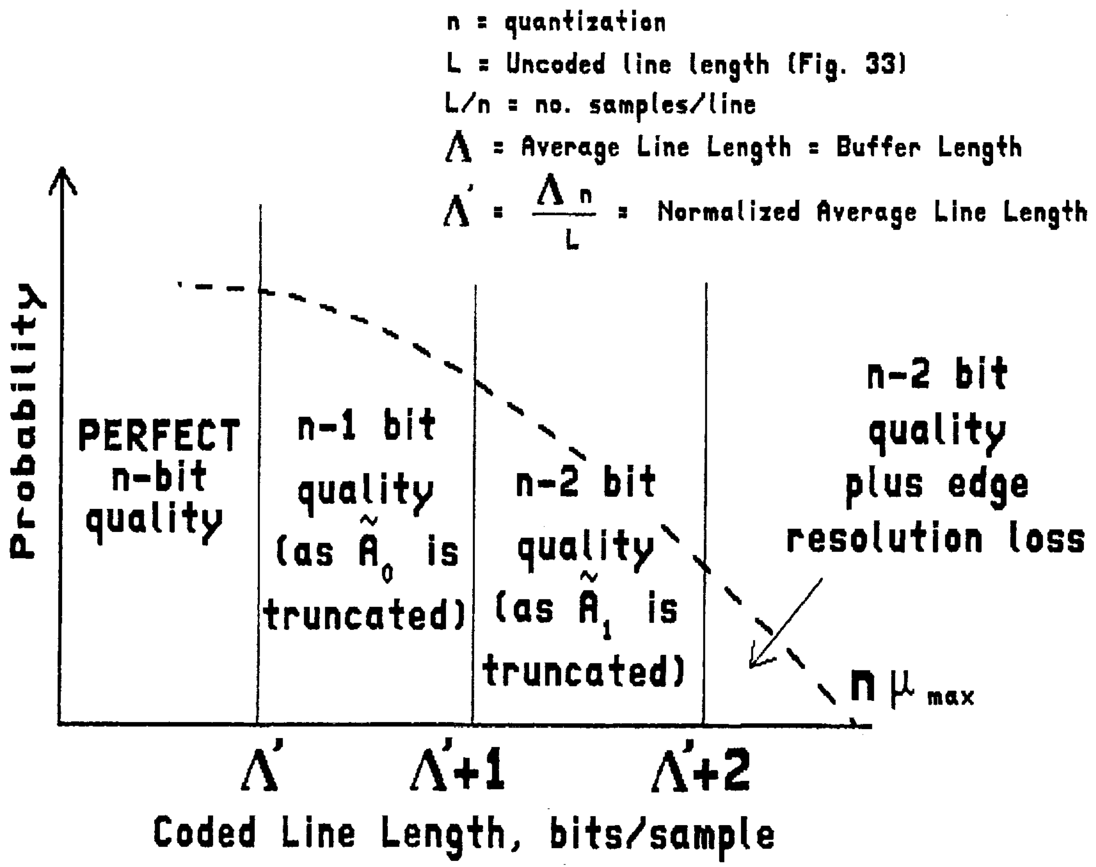

n = quantization
L = Uncoded line length (Fig. 33)
L/n = no. samples/line
Λ = Average Line Length = Buffer Length
$\Lambda' = \frac{\Lambda n}{L}$ = Normalized Average Line Length
Probability
PERFECT n-bit quality
n-1 bit quality (as $\tilde{A}_0$ is truncated)
n-2 bit quality (as $\tilde{A}_1$ is truncated)
n-2 bit quality plus edge resolution loss
n μ_{max}
Λ'
$\Lambda'+1$
$\Lambda'+2$
Coded Line Length, bits/sample

Fig. 39. Example: Normalized Line-Length Distribution for Bit Priority PSI? of Fig. 38

As in earlier examples, if a normalized coded line length (bits/sample) is shorter than the buffer length, Λ' here, there is no truncation.

As the normalized line length is increased past Λ', end bits of $\tilde{A}_0$ will be deleted. That is, the least-significant bits of one end of such a line will be deleted, allowing only n-1 bit reconstruction (keep in mind that this error represents only one quantization level out of 4,096 for n = 12 bit data). All $\tilde{A}_0$ will be lost when normalized line lengths reach $\Lambda'+1$ bits/sample.

Similarly, the second lsbs of some or all samples will be lost for normalized line lengths between $\Lambda' + 1$ and $\Lambda' + 2$ as sequence $\tilde{A}_1$ is effected. With two lsb's gone, only n-2 bit precision reconstruction can be accomplished.

As normalized line lengths exceed $\Lambda' + 2$, the coded sequence PSI15,K+$[\tilde{M}_x^{n,2}]$ in (173) will be affected, which causes truncation of samples at the ends of lines, as before. Those samples not truncated can be reconstructed with n-2 bit precision. By alternating the direction of coding, truncated samples can be filled in by interpolation, as before.

What we have done in these examples is to identify several different parts of a line's representation and order those parts by their relative importance. This ordering turns into a relative "priority" for the coded bits when they are loaded (in their order of importance) into a FIFO buffer. The priorities in this last example were:

(1) Samples have decreasing importance from one end of a line to the other.

(2) The priority direction in (1) above alternates from line to line.

(3) The next least-significant bits of all samples ($\tilde{A}_1$) are less important than all the n-2 most-significant bits.

(4) The least-significant bits of all samples in a line ($\tilde{A}_0$) are less important than all other bits.

The advantages of this "more graceful" rate control are obtained at the cost of the slightly greater implementation complexity required to reorder the coded data bits before loading the FIFO.

Priority Variations. Certainly there are many variations to this prioritization scheme. Among them:

- $\tilde{A}_0$, $\tilde{A}_1$, etc., could apply only to the edges, which leaves the central portion error-free.

- The error that results from truncation can be reduced without a performance penalty by rounding the most-significant bit samples that are coded by using PSI15K,+. (For a single lsb truncation, add a pseudorandom binary sequence during reconstruction. This will assure that the average error is zero.)

- Resolution can be more directly involved in the planned priorities. A line (or portions of a line) could be rearranged before coding into sequences of the even samples followed by the odd samples and vice versa, as shown in Fig. 40.

Figure 40 assumes that every line is arranged into odd then even samples. Once coded, the even samples will be deleted when the FIFO overflows. Assuming alternating left-right/right-left coding, it is easy to see that

- Provided that no more than 1/4 of a line's samples are truncated (i.e., half the even), then (175)

- All deleted samples (except on the image edge) will be completely surrounded on all sides by samples which were not deleted, allowing for an accurate interpolation reconstruction. (176)

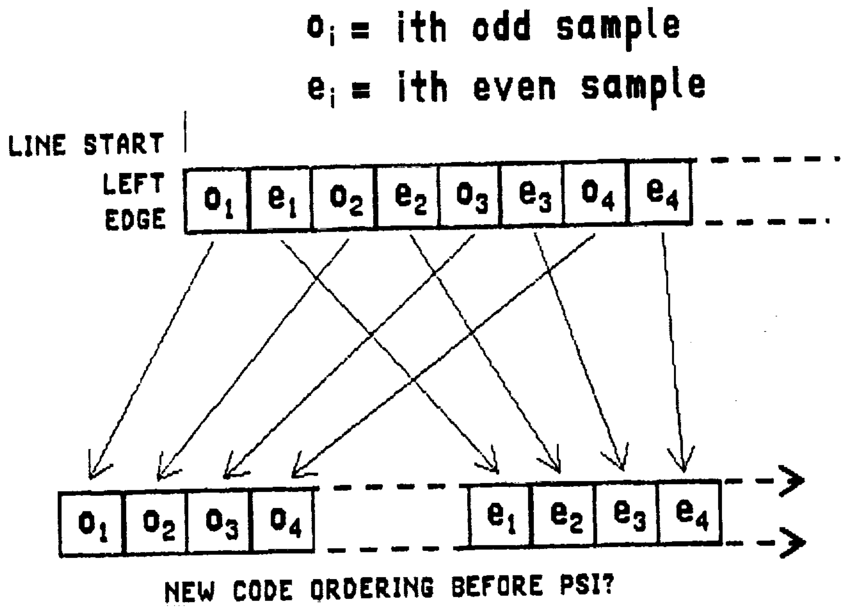


Fig. 40. Ordering Samples by Odd/Even

By alternating the ordering of odd and even samples as in Fig. 41, condition (175) above can be replaced by the condition

- Provided that no more than 1/2 of a line's samples are truncated (i.e., all the even or all the odd) (177)

Effects on Prediction. A slight penalty must be paid in prediction performance. Assuming the simpler case in (175) and (176), prediction of even samples is unaffected. But the prediction of odd samples cannot use an adjacent (possibly missing) even sample. It must use the previous odd sample and/or the corresponding odd sample in the line above.

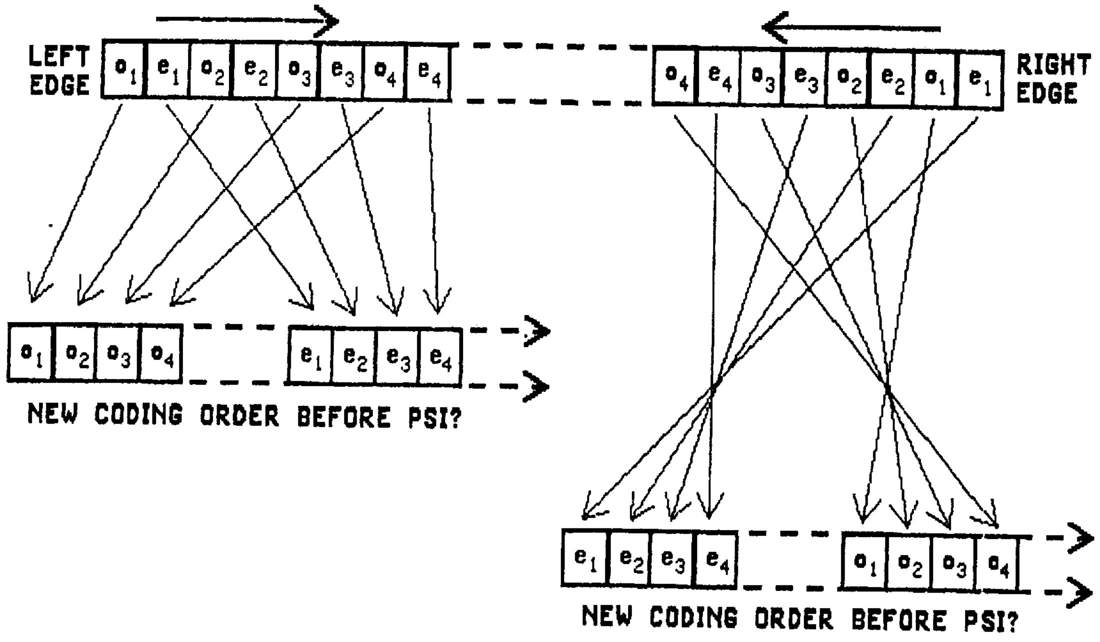


Fig. 41. Alternating the Ordering of Samples by Odd/Even

A Multispectral Frame Buffer Definition

A multispectral scanner will ultimately produce many two-dimensional images, one for each spectral band. A multispectral IMAGE is a collection of these 2-D images.

The HIRIS instrument currently being developed simultaneously produces one line at a time for all of its 192 spectral bands. The collection of one line from each band is called a FRAME. The collection of many FRAMES constitutes a multispectral IMAGE. See Fig. 23 for further clarification.

Sample Data. HIRIS could generate an extraordinary range of data entropies when comparisons are made over all of its spectral bands. Ultimately the actual variability will depend on the level of uncorrected sensor noise and/or the method of prediction used. Here we will focus on a worst-case entropy variability situation.

Figure 42 was derived from a sample multispectral image taken with an earlier 224-band instrument called AVIRIS [15]. The flat-field corrected image used a quantization of 10 bits/sample in each band. The figure shows a plot of observed entropies for each band obtained from the use of the 2-D predictor described in (112). The entropy of the average prediction error distribution (over all bands) was

$$\bar{H}_\delta = 4.9 \text{ bits/sample} \tag{178}$$

Performance using an appropriate PSI14,K+ followed this graph closely, which yielded an average performance for the complete image of

$$\Lambda' = 4.6 \text{ bits/sample} \tag{179}$$

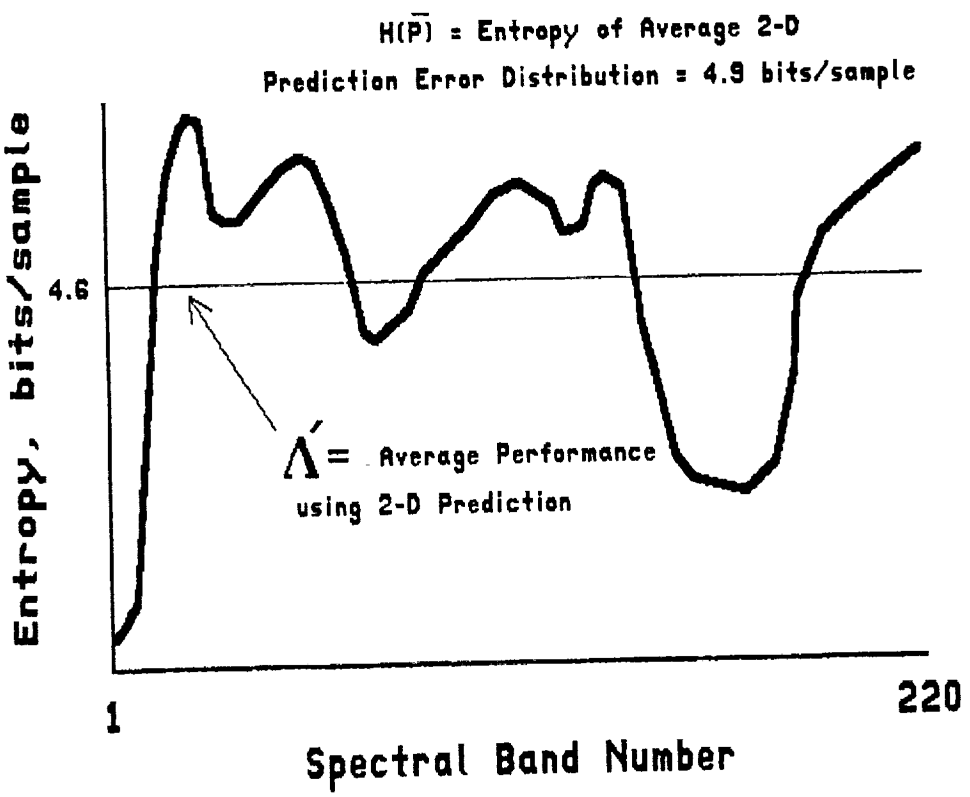


Fig. 42. HIRIS Entropy Variability–AVIRIS Image

Both $\bar{H}_\delta$ and Λ' in (178) and (179) are about 0.3 bit/sample lower than would be achieved by a 1-D Cross-track predictor.

The Buffering Problem. The graph in Fig. 42 represents an average over the many FRAMES that make up the sample image. FRAME to FRAME fluctuations of this graph would reflect the relatively smaller line-to-line activity variations within one spectral band.

Here we are interested in constraining rate over a single FRAME. For discussion purposes we will assume that the graphs for Λ' and $\bar{H}_{\delta}$ in Fig. 42 are representative of a particular FRAME.

The general impact caused by requiring individual FIFO line buffer rate control for each spectral band in a FRAME – all fixed to a single line length – should be evident from Fig. 42. Clearly, with a normalized line length equal to the average performance Λ', there would not only be a lot of truncation (higher activity bands) but also a lot of filler bits (lower activity bands). Of course, the line buffer length might be larger than Λ', at least reducing the truncation problem. But we won't count on that here.

The priority approaches just discussed are applicable here. In fact, individual bands could be assigned different buffer lengths to customize the effect of truncations caused by the FIFO buffers being too small. But truncations and other errors would still occur that could be avoided with a larger buffer.

As in earlier discussions, a FRAME Buffer of length

$$\Lambda' \times (\text{\# samples per FRAME}) \text{ bits} \tag{180}$$

could hold a complete coded FRAME without error – clearly a significant improvement over many line buffers of the same total length.

Remember, any buffer we talk about is really a convenient way of expressing a bit count constraint placed on a block of data or an interval of time. The buffer itself may not need to be physically present. For example, the concept of a FRAME Buffer may really mean that a data link allows the transfer of a certain number of bits over the time interval needed to generate a FRAME ("Frame Time"). As long as all data can be retrieved directly from the instrument, coded and transferred immediately with the allowed number of bits, no FRAME Buffer would be necessary.

A convenient way of achieving this is to constrain the number of bits to be fixed for each spectral line – with the consequences noted above. We will look for another way.

Note that the key advantage of a FRAME Buffer is that variable-length lines are allowed. The sum of those lengths must fit in the buffer.

If a FIFO FRAME Buffer were really available, a number of FRAME oriented priorities would be applicable. For example, reordering the spectral bands by priority would result in deletion of the least-important bands if the buffer weren't big enough. But this all-or-nothing approach might not be appropriate either. Instead let's consider a way to distribute any truncation and/or error losses more evenly over a FRAME.

Now we again need some additional notation. Let $\tilde{X}(j)$ denote a line from spectral band j, with uncoded length $\mathscr{L}(\tilde{X}(j)) = L$, where $j = 1, 2, \ldots T$. Here we have used the same notation as in Fig. 33, except that now each line is derived from a different spectral band.[21]

Let

$$L_j = \text{Coded Length of (a complete) } \tilde{X}(j) \qquad (181)$$

Then the average coded line length is

$$\Lambda = \Lambda' \frac{L}{n} = \frac{1}{T} \sum_{j=1}^{T} L_j \text{ bits/line} \qquad (182)$$

where Λ' is the same term as in Fig. 42, and $T\Lambda$ is the minimum-sized FRAME Buffer that could hold all of the coded $\tilde{X}(j)$.

Let

$$\theta_j = \frac{L_j}{T\Lambda} = \text{fraction of total bits generated by } \tilde{X}(j) \qquad (183)$$

$$B = \text{Actual Frame Buffer Length, in bits} \qquad (184)$$

[21] Note that T might be variable since some operational modes may delete some of the spectral bands from consideration.

and let

$$\gamma = \frac{B}{T\Lambda} = \left\{ \begin{array}{l} \text{Fractional Comparison} \\ \text{of Actual Frame Buffer} \\ \text{size with the minimum size} \\ \text{that can hold all coded} \\ \text{spectral bands} \end{array} \right. \qquad (185)$$

Now we can allocate the total number of bits available, B, to T FIFO line buffers according to the priority given in the $\{\theta_j\}$. The buffer sizes are

$$B_j = \theta_j B \text{ bits} \qquad (186)$$

Data for each coded individual spectral band will now be controlled by their individual FIFO buffers, instead of a FRAME Buffer.

And clearly, since $\Sigma\theta_j = 1$,

$$\sum_{j=1}^{T} B_j = B \text{ bits} \qquad (187)$$

By substituting from (181), we have

$$B_j = \theta_j B = \frac{L_j}{T\Lambda}(T\Lambda\gamma) = \gamma L_j \text{ bits} \qquad (188)$$

So the LINE Buffer size B_j is a fraction of the size needed to hold all of a coded $\tilde{X}(j)$. When $\gamma = 1$

$$B_j = L_j \text{ and } B = T\Lambda \text{ bits} \qquad (189)$$

By this process, we have established a set of variable-length FIFO line buffers whose individual lengths reflect their relative need for bits, based on the length of coded data (an "entropy" criterion).

When $\gamma \geq 1$, there is no loss. If γ is decreased below unity, each spectral band will begin to experience some truncation (and/or error) since each FIFO line buffer is no longer big enough to hold a complete coded line.

Thus, we have distributed the effect of FIFO buffer truncation over all the bands rather than only the very active. Of course, the priority scheme used for each spectral band could still be customized.

Note that if the variable buffer sizes could be set up a priori, the existence of real physical LINE Buffers would not be needed. Each spectral line, $\tilde{X}(j)$, would be coded on-the-fly with $\theta_j B$ bits. This is not unreasonable, since an entropy graph such as Fig. 42 is representative, even if it does vary.

But since the L_j for each spectral band can be expected to be highly correlated from FRAME to FRAME, the assignment of FIFO LINE Buffer lengths B_j could be made adaptive by using an L_j calculated from the previous FRAME for each spectral band.

Adding Other Priorities. The $\{\theta_j\}$ are priorities based on a criterion of entropy. Other criteria can also be integrated into the determination of FIFO LINE Buffer lengths, as we shall illustrate.

Let

$$\beta_j, j = 1, 2, \ldots T \tag{190}$$

be a set of spectral band priorities based on some other criteria – perhaps "scientific importance to the current investigation." Then we must have

$$0 \leq \beta_j \leq 1 \text{ and } \sum_{j=1}^{T} \beta_j = 1 \tag{191}$$

To weight the two criteria, we choose $0 \leq a \leq 1$ for the $\{\theta_j\}$ set and $(1-a)$ for the $\{\beta_j\}$ set.

We can now modify the FIFO LINE Buffer assignments in (186) to

$$B_j' = [a\theta_j + (1 - a)\, \beta_j]\, B_j \text{ bits} \qquad (192)$$

for spectral band j.

The addition of such priorities can cause the assignment of bits in (192) to exceed the amount needed to achieve noiseless coding (L_j in Eq. 181). But this can be accounted for by iterating the assignment of bits to redistribute the extra bits to other lines.

APPENDIX A

MORE ON THE MAPPING OF PREDICTION ERRORS

The function that maps prediction errors while placing priority on negative values over positive is transferred from Eq. 18 for convenience.

$$\mathcal{M}^{-+}(x_i, \hat{x}_i) = \delta_i = \begin{cases} 2\Delta_i & 0 \le \Delta_i \le \theta \\ 2|\Delta_i| - 1 & -\theta \le \Delta_i < 0 \\ \theta + |\Delta_i| & \text{Otherwise} \end{cases} \tag{A-1}$$

and θ is defined in Eq. 19.

When the positions of positive and negative differences in (16) are reversed, this mapping becomes

$$\mathcal{M}^{+-}(x_i, \hat{x}_i) = \begin{cases} 2\Delta_i - 1 & 0 < \Delta_i \le \theta \\ -2\Delta_i & -\theta \le \Delta_i \le 0 \\ \theta + |\Delta_i| & \text{Otherwise} \end{cases} \tag{A-2}$$

If $\tilde{X}$ and $\hat{\tilde{X}}$ are data and prediction sequences, then $\mathcal{M}^{-+}(\tilde{X},\hat{\tilde{X}})$ means the application of (A-1) to each sample of $\tilde{X}$, $\hat{\tilde{X}}$; and similarly for $\mathcal{M}^{+-}(\cdot,\cdot)$.

In most situations, the two mappings are statistically equivalent. However, certain kinds of images exhibiting many long brightness ramps might benefit from one or the other, or both (an adaptive preprocessor).

Relationship between $\mathcal{M}^{+-}$ and $\mathcal{M}^{-+}$

Define the Ones Complement of a sequence of one or more n-bit data samples $\tilde{y} = y_i\, y_2\, y_3 \ldots$ by

$$\overline{\text{ONE}}\,(\tilde{y}) = 2^n - 1 - y_1,\ 2^n - 1 - y_2,\ 2^n - 1 - y_3, \ldots \tag{A-3}$$

Then, with x_i and $\hat{x}_i$ as individual n-bit samples and their predictions

$$\mathcal{M}^{-+}(x_i, \hat{\tilde{x}}_i) = \mathcal{M}^{+-}(\overline{\text{ONE}}(x_i), \overline{\text{ONE}}(\hat{x}_i)) \tag{A-4}$$

and

$$\mathcal{M}^{+-}(x_i, \hat{\tilde{x}}_i) = \mathcal{M}^{-+}(\overline{\text{ONE}}(x_i), \overline{\text{ONE}}(\hat{x}_i)) \tag{A-5}$$

But more important, by extending this to data and prediction sequences $\tilde{X}$ and $\hat{\tilde{X}}$, we have

$$\mathcal{M}^{-+}(\tilde{X},\hat{\tilde{X}}) = \mathcal{M}^{+-}(\overline{\text{ONE}}(\tilde{X}), \overline{\text{ONE}}(\hat{\tilde{X}})) \tag{A-6}$$

and

$$\mathcal{M}^{+-}(\tilde{X},\hat{X}) = \mathcal{M}^{-+}(\overline{\text{ONE}}(\tilde{X}), \overline{\text{ONE}}(\hat{X})) \tag{A-7}$$

Test Vectors

Equations (A-6) and (A-7) tell us that a data sequence (and prediction) that produces a known sequence of δ's by one mapping can easily be converted to an alternate data sequence (and prediction) **that will generate the same sequence of δ's** by using the alternate mapping. It is, of course, the δ's to which adaptive variable length coding is applied (Fig. 4). The two complementary data sequences might be test vectors for this coder.

Decoding

$\tilde{X}$ and $\hat{\tilde{X}}$ can be retrieved by applying the inverse mapping operations of Eqs. (A-1), (A-2), (A-4)–(A-7) and any a priori information, and Eq. (115) so that

$$(\tilde{X},\hat{\tilde{X}}) = \text{inv}\,\mathcal{M}^{-+}\{\mathcal{M}^{-+}(\tilde{X},\hat{\tilde{X}})\} \tag{A-8}$$

and

$$(\tilde{X},\hat{\tilde{X}}) = \text{inv}\,\mathcal{M}^{+-}\{\mathcal{M}^{+-}(\tilde{X},\hat{\tilde{X}})\} \tag{A-9}$$

But suppose we apply the $+-$ inverse map to the result of applying the $-+$ map:

$$(\tilde{X}',\hat{\tilde{X}}') = \text{inv}\mathcal{M}^{+-}\{\mathcal{M}^{-+}(\tilde{X},\hat{\tilde{X}})\} \tag{A-10}$$

We don't get the original sequence back, but we know that by (A-7)

$$\mathcal{M}^{+-}(\tilde{X}',\hat{\tilde{X}}')) = \mathcal{M}^{-+}(\overline{\text{ONE}}(\tilde{X}'), \overline{\text{ONE}}(\hat{\tilde{X}}')) \tag{A-11}$$

so we must have

$$(\tilde{X},\hat{\tilde{X}}) = (\overline{\text{ONE}}(\tilde{X}'), \overline{\text{ONE}}(\hat{\tilde{X}}')) \tag{A-12}$$

Ultimately, all this means that a "decoder" using the $\mathcal{M}^{-+}(\quad)$ map could be used to decode data that had been generated using the $\mathcal{M}^{+-}(\quad)$ map, and vice versa.[22]

[22]Note that any Reference Sample indicated in Chapter IV must also be complemented before decoding begins.

APPENDIX B

MORE ON THE PSI1,k SPLIT-SAMPLE OPTIONS

Following the same notation as in Chapter III, Split-Sample Code "Option" PSI1,k is defined in (58) and Fig. 12 by

$$\Psi_{1,k}[\tilde{\delta}^n] = \Psi_1[\tilde{M}^{n,k}] * \tilde{L}_k \tag{B-1}$$

where $\tilde{\delta}^n$ is a J-sample block of n-bit samples, $\tilde{M}^{n,k}$ are all the most-significant n-k bit samples of $\tilde{\delta}^n$, and $\tilde{L}_k$ are all the least-significant k-bit samples of $\tilde{\delta}^n$.

PSI1,k constitutes one of several coding algorithms that are used with adaptive coder PSI14 or PSI14,K and others. Since decisions that determine which code option to be used are made over a complete J-sample data block, it is natural to arrange the individual pieces of the coded blocks PSI1$[\tilde{M}^{n,k}]$ and $\tilde{L}_k$ as in (A-1). There are subtle advantages of certain approaches to implementations and a performance advantage under certain rate-controlled situations, as discussed in Chapter IV. Consequently we have maintained the definition in (A-1) throughout. However, here we'll take a closer look at this definition.

PUTTING THE PIECES BACK TOGETHER

We need to expand our notation slightly. Let

$$\tilde{\delta}^n = \delta_1\, \delta_2 \ldots \delta_J \tag{B-2}$$

be the sequence of n-bit samples to be coded, and let

$$\tilde{M}^{n,k} = m_1\, m_2 \ldots m_J \tag{B-3}$$

denote the sequence of n-k bit samples of $\tilde{\delta}^n$, and

$$\tilde{L}_k = lsb_1 * lsb_2 * \ldots * lsb_J \tag{B-4}$$

denote the corresponding sequence of all the k-bit least-significant bit samples.

Then, by (27), (29) and (A-1), we have the expansion

$$\Psi_{1,k}[\tilde{\delta}^n] = fs[m_1] * fs[m_2] * \ldots fs[m_J] * lsb_1 * lsb_2 * \ldots lsb_J \quad \text{(B-5)}$$

we could instead rearrange (A-5) to

$$\Psi'_{1,k}[\tilde{\delta}^n] = fs[m_1] * lsb_1 * fs[m_2] * lsb_2 * \ldots * fs[m_J] * lsb_J$$

or

$$\Psi''_{1,k}[\tilde{\delta}^n] = lsb_1 * fs[m_1] * lsb_2 * fs[m_2] * \ldots lsb_J * fs[m_J] \quad \text{(B-6)}$$

With this arrangement it is more obvious that the coding of any individual sample for $\tilde{\delta}^n$, say δ_i, is the application of the variable-length code, say $C_{n,k}[\cdot]$, where

$$C_{n,k}[\delta_i] = lsb_i * fs[m_i] \quad \text{(B-7)}$$

We have merely collected the individual coded pieces of each sample in (A-5).

Optimality of the Code, $C_{n,k}[\ ]$

Yeh [11] has shown that the set of simple codes in (B-7) is equivalent to a class of Huffman codes under the Humblet condition.[23] This is a powerful result because it explains more directly why the measured performance of each PSI1,k code option is so good – that over **a limited entropy range, there is none better**.

Multiplexing

Now, subscript or superscript the n, $\tilde{\delta}^n$, δ_i, m_i, lsb_i, and k in the descriptions above by a, b and c to denote that each block (and its coding) is derived from different data sources.

[23]Assuming the "optimized fs code."

Clearly, we could **mix** the coded blocks from the three data sources as, from (B-6)

$$\Psi''_{1,k_a}\left[\tilde{\delta}_a^{n_a}\right] * \Psi''_{1,k_b}\left[\tilde{\delta}_b^{n_b}\right] * \Psi''_{1,k_c}\left[\tilde{\delta}_c^{n_c}\right] \tag{B-8}$$

But we could also go further than this and multiplex the individual codeword pieces from (B-7) for any n_a, n_b, n_c, k_a, k_b, k_c as

$$\left(lsb_1^a * fs\left[m_1^a\right]\right) * \left(lsb_1^b * fs\left[m_1^b\right]\right) * \left(lsb_1^c * fs\left[m_1^c\right]\right) * \ldots *$$
$$\left(lsb_J^a * fs\left[m_J^a\right]\right) * \left(lsb_J^b * fs\left[m_J^b\right]\right) * \left(lsb_J^c * fs\left[m_J^c\right]\right) \tag{B-9}$$

without incurring any decoding difficulty, provided that the n_a, n_b, n_c, and k_a, k_b, k_c, quantities are known by a decoder. But presumably these quantities will only change because of the internal decisions of the three separate adaptive coders employing PSI1,k code options.[24] In that case, three corresponding code identifiers would precede (B-9) to reveal any changes.

[24]For example, three separate PSI14 coders, all with starting option PSI1,0 (i.e., λ = 1).

APPENDIX C

COMPARING PSI0,1 WITH PSI1,0

By definition,

$$\mathcal{L}(\Psi_{0,1}[\tilde{\delta}^n]) = \mathcal{L}(\Psi_0[\tilde{M}^{n,1}]) + J \text{ bits} \tag{C-1}$$

where $\tilde{M}^{n,1}$ is the sequence of the most-significant n-1 bit samples of $\tilde{\delta}^n$. We wish to compare this with

$$F_0 = \mathcal{L}(\Psi_1[\tilde{\delta}^n]) = \mathcal{L}(\Psi_1[\tilde{M}^{n,0}]) \text{ bits} \tag{C-2}$$

By Eq. 80,

$$F_1 = \mathcal{L}(\Psi_1[\tilde{M}^{n,1}]) \text{ bits} \tag{C-3}$$

is the length of the Fundamental Sequence for $\tilde{M}^{n,1}$. By Ref. 5

$$F_1 \geq \frac{1}{2}(F_0 + J - \Sigma\,(\text{lsbs of } \tilde{\delta}^n)) \tag{C-4}$$

which gives us Eq. 81 for k = 1, if the split least-significant bits are random.

But from (41)

$$\mathcal{L}(\Psi_0[\tilde{M}^{n,1}]) \leq \left\lfloor \frac{F_1}{3} \right\rfloor + 2(F_1 - J) \tag{C-5}$$

Then, by using (C-1), we get

$$\mathcal{L}(\Psi_{0,1}[\tilde{\delta}^n]) \leq \left\lfloor \frac{F_1}{3} \right\rfloor + 2F_1 - J$$

or

$$3\mathcal{L}(\Psi_{0,1}[\tilde{\delta}^n]) \leq 7F_1 - 3J$$

By substituting (C-4)

$$6\mathcal{L}(\Psi_{0,1}[\tilde{\delta}^n]) \le 7(F_0 + J - \Sigma \text{ lsbs}) - 6J = 7F_0 + J - 7\Sigma \text{ lsbs}$$

if we ignore the integer requirement, then

$$\mathcal{L}(\Psi_{0,1}[\tilde{\delta}^n]) \le \gamma_{0,1} = \frac{1}{6}[7F_0 + J - 7\Sigma \text{ lsbs}] \qquad \text{(C-6)}$$

The minimum and maximum values of any $\gamma_{0,1}$ occur when all the split lsbs are ones or zeroes, respectively, giving us

$$\min \gamma_{0,1} = \frac{7}{6}F_0 - J \qquad \text{(C-7)}$$

$$\max \gamma_{0,1} = \frac{7}{6}F_0 + \frac{J}{6} \qquad \text{(C-8)}$$

When the lsbs are distributed randomly, $\gamma_{0,1}$ would lie between these limits, as

$$E\{\gamma_{0,1} \mid \text{random lsbs}\} = \frac{7}{6}F_0 - \frac{5}{12}J \qquad \text{(C-9)}$$

These results are plotted in Fig. C-1.

The figure plots $\gamma_{0,1}$ from (C-7)–(C-9) as a function of F_0. It also shows F_0 plotted against itself since this is what we wish to compare.

Remember, we are really interested in the average properties of $\gamma_{0,1}$ in (C-6), that is, $E\{\gamma_{0,1}\}$. The $\gamma_{0,1}$ plots in Fig. C-1 are contingent on specific distributions of the split least-significant bits – all ones, all zeroes, or random.

Not until entropies start exceeding about 3 bits/sample (corresponding to $F_0 >$ 3J) can the distribution of lsbs be considered random. Then $E\{\gamma_{0,1}\}$ and the performance of PSI0,1 will follow the center curve for $\gamma_{0,1}$ (Eq. C-9). But by then, this curve lies above F_0, having crossed at $F_0 = 5J/2$.

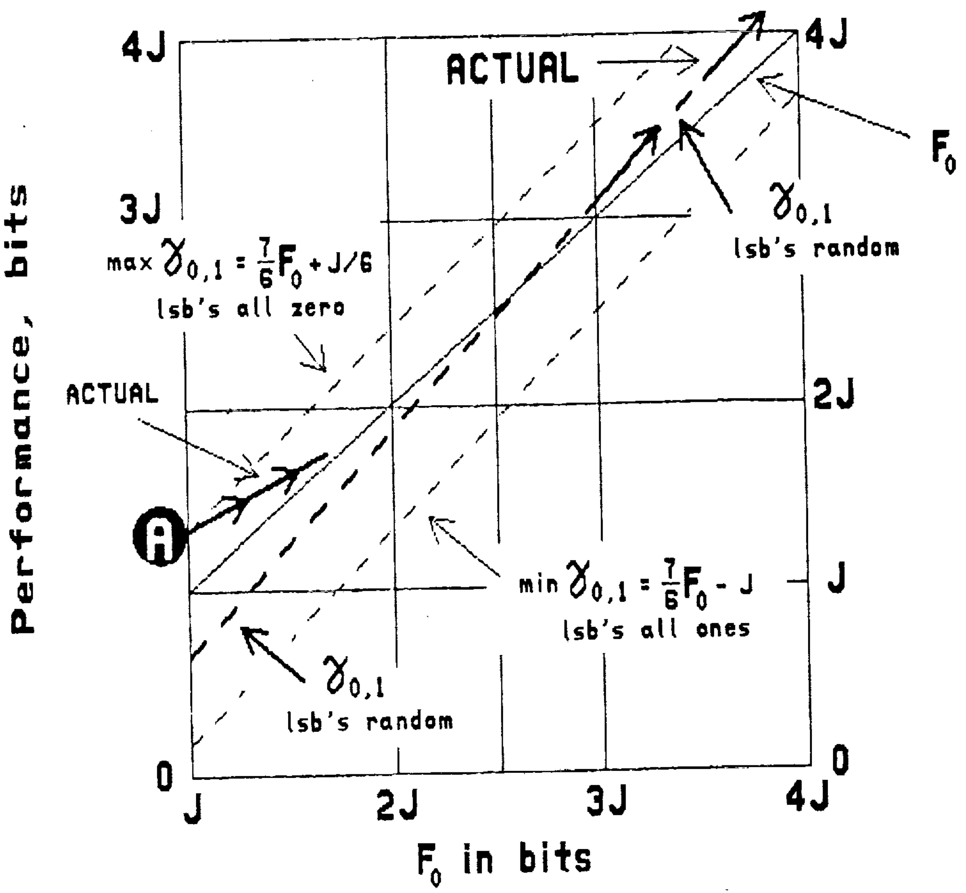


Fig. C-1. $\gamma_{0,1}$ Performance Bounds

Although there are sequences that can be coded better with PSI0,1 than PSI1, on the average, PSI1 will do better. Furthermore, as $F_0 > 5J/2$, PSI14 will be choosing the next Split-Sample mode PSI1,1 which will outperform PSI1 itself.

As entropies are lowered, the lsbs will tend toward more and more zeroes until at $\bar{H}_\delta = 0$ all the lsbs MUST BE ZERO. There is only one possibility for $\gamma_{0,1}$ as shown by point (A) in Fig. C-1. Here $\gamma_{0,1}$ corresponds to max $\gamma_{0,1}$ in Eq. C-8. Clearly, the real $E\{\gamma_{0,1}\}$ will gradually move away from point (A) and merge with the graph for random lsbs. Until this happens, the average performance of PSI1 should be better than PSI0,1.

But we have already considered the higher entropies. Thus, it would appear that for all practical purposes, the average performance of PSI1 should always be better than that of PSI0,1.

APPENDIX D

GAIN/OFFSET SENSOR NOISE

GAUSSIAN ENTROPY FUNCTION

The density function for a Gaussian random variable ζ is given by the familiar

$$f(\zeta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(\zeta-m)^2/2\sigma^2} \quad \text{(D-1)}$$

where

$$m = \text{mean value} \quad \text{(D-2)}$$

and

$$\sigma = \text{Standard Deviation} \quad \text{(D-3)}$$

It can be shown that a good approximation to the entropy of a quantized ζ, for $\sigma > 1$, is given by the Gaussian entropy function [16]

$$H_G(\sigma) = \frac{1}{2} \log_2 [2\pi e(\sigma^2 + \frac{1}{12})] = 2.047 + \frac{\ln(\sigma^2 + \frac{1}{12})}{2 \ln 2} \quad \text{(D-4)}$$

The inverse of D-4 is given as

$$\sigma = (e^{[(H_G(\sigma)-2.047)/1.386]} - \frac{1}{12} \quad \text{(D-5)}$$

PREDICTION ENTROPY

Let $\tilde{P} = \{p_j\}$ denote an observed distribution of prediction error differences, and let

$$\sigma(\tilde{P}) \quad \text{(D-6)}$$

be its calculated standard deviation in data numbers (DN). A direct calculation of the entropy of $\tilde{P}$ is then from Eq. (10)

$$H(\tilde{P}) = -\sum_{j} p_j \log_2 p_j \text{ bits/sample} \tag{D-7}$$

From numerous simulations involving various data sources [12], it was observed that for a broad range of differential entropies,

$$H(\tilde{P}) = H_G\,(\sigma(\tilde{P})) \tag{D-8}$$

Discrepancies of less than 0.1 bit/sample in the two sides of (D-7) were typical. Thus, the Gaussian model can be assumed to be reasonably accurate in estimating differential entropies for real quantized data sources.

Introducing Noise

The sample-to-sample sensor noise effects caused by variations in gain sensitivity and offset for some modern instruments can also be well modeled as Gaussian, and independent of the real signal.[25] We can use these results to obtain a reasonable measure of the expected effect of this gain/offset noise on code rate. It is of particular interest to investigate this impact because this form of noise is potentially correctable within the instrument data system [17].

Figure D-1 shows the Preprocessor portion of module PSI14,K+ with an input that includes this sensor noise.

Here, s_i is the real ith signal value and x_i is the corresponding value seen by the Preprocessor AFTER a Gaussian noise signal, n_i, is added to it. The ith difference signal, Δ_i, becomes

[25]For our purposes here, the real signal may actually already include other non-correctable noise effects, such as shot noise.

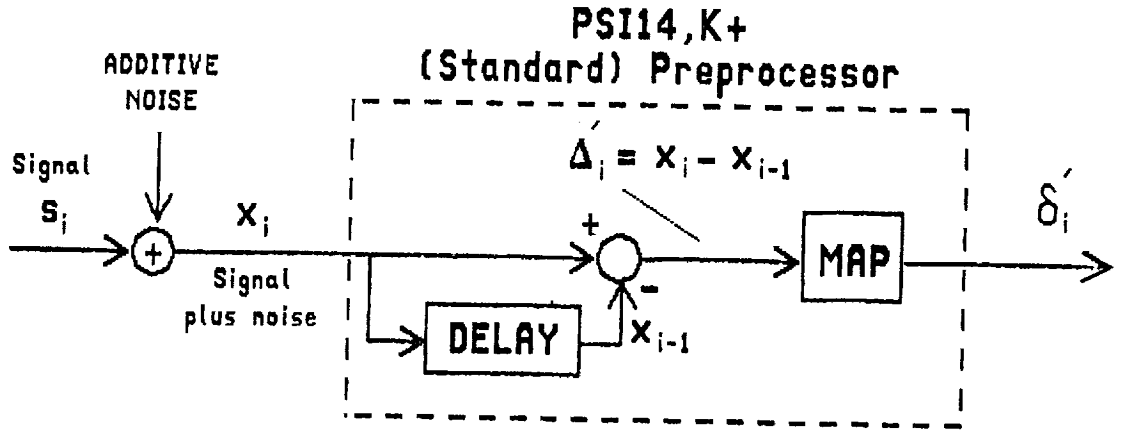


Fig. D-1. Signal Plus Noise

$$\begin{aligned}\Delta'_i &= x_i - x_{i-1}\\ &= s_i + n_i - (s_{i-1} + n_{i-1})\\ &= (s_i - s_{i-1}) + (n_i - n_{i-1})\\ &= \Delta_i + n'_i \qquad \text{(D-9)}\end{aligned}$$

where Δ_i is the difference signal if noise were not present, and n'_i is a new zero mean Gaussian random variable with Standard Deviation σ'_n. This suggests the statistically equivalent diagram in Fig. D-2 where we now make use of (D-7) to assume that Δ_i is a zero mean Gaussian random variable with standard deviation σ_s.

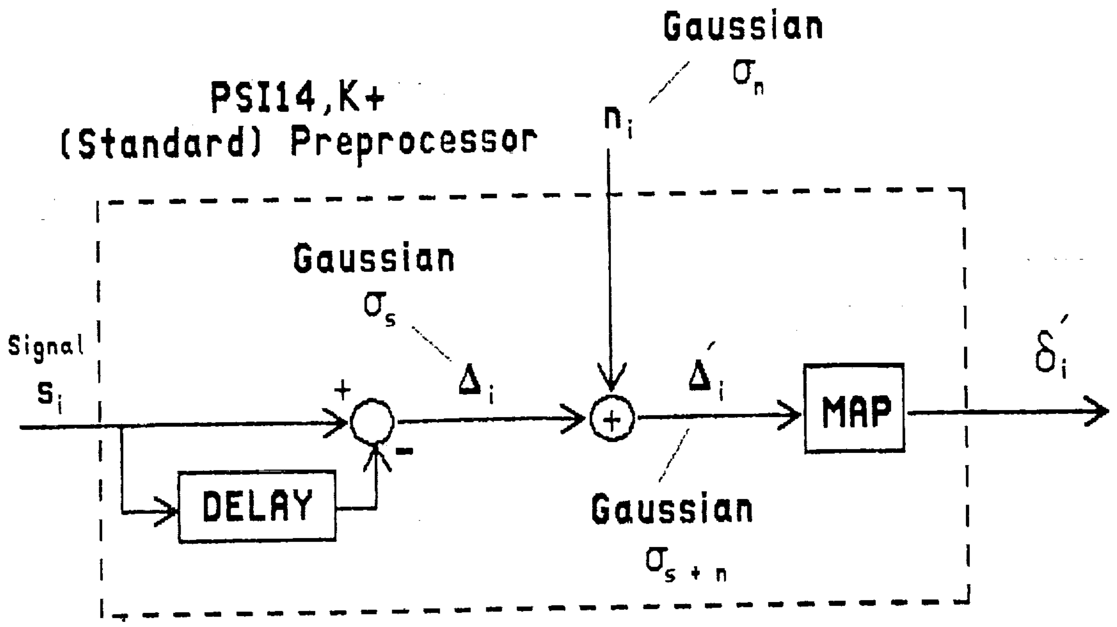


Fig. D-2. Difference Signal Plus Noise

Here the error signal that eventually gets coded is more clearly the sum of the two independent zero mean Gaussian signals Δ_i and n'_i. Then summarizing, we have

σ'_n = Standard Deviation for Gaussian Noise Signal

σ_s = Standard Deviation for (Gaussian) Signal Predictor Error (D-10)

σ_{s+n} = Standard Deviation Gaussian Signal $= \sigma_s + \sigma'_n$

We can now investigate coding efficiency with and without noise by evaluating the impact on Gaussian entropy $H_G(\sigma_{s+n}) = H_G(\sigma_s + \sigma'_n)$ for various levels of noise, as specified by σ'_n or $H_G(\sigma'_n)$.

We will not exhaustively study these effects here since we are not dealing with any specific instrument. It will suffice to get a feel for the effects. Plots of $H_G(\sigma_s + \sigma'_n)$ vs $H_G(\sigma_s)$ for various levels of $H_G(\sigma'_n)$ are shown in Fig. D-3. Some similar curves first appeared in Ref. 1.

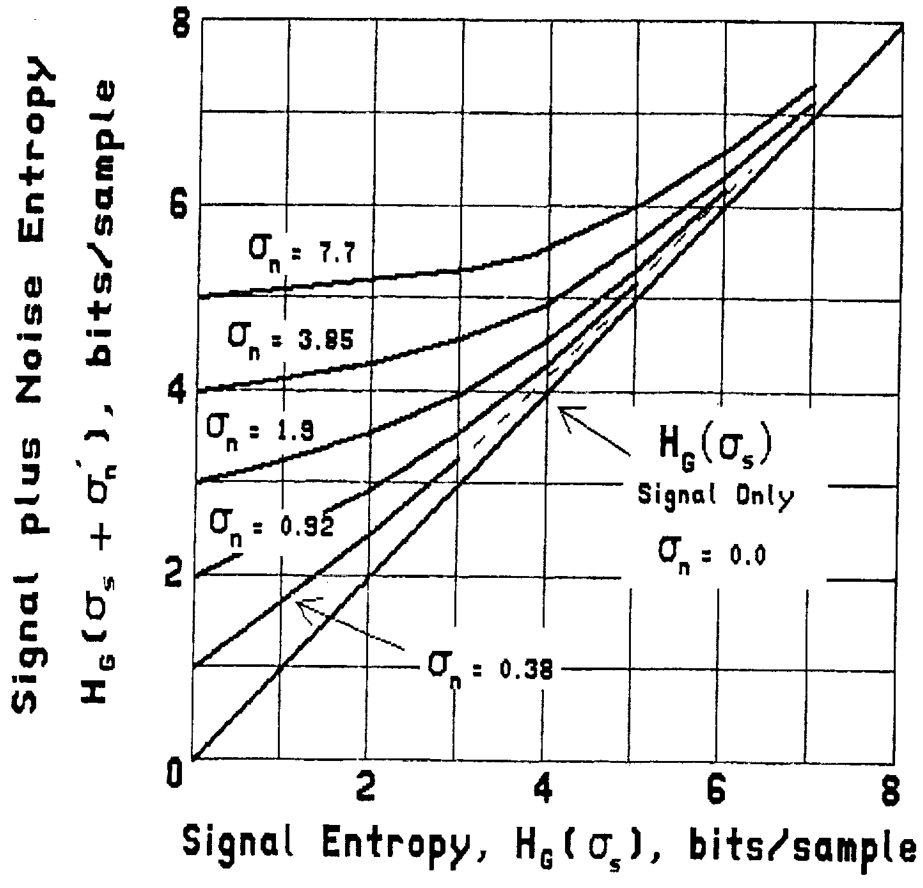


Fig. D-3. Signal Plus Noise Entropies

DISCUSSION

The curve to use for comparison in Fig. D-3 is the 45° line that represents signal entropy alone (i.e., $H_G(\sigma_S)$ vs $H_G(\sigma_S)$).

As noise is increased above zero, the overall entropy, $H_G(\sigma_S + \sigma'_n)$, increases at all signal entropy values. But the net increase caused by the noise diminishes dramatically as the underlying signal entropy increases. For example, when the signal entropy is $H_G(\sigma_S) = 0$, the impact of a noise signal with 4 bits/sample of entropy ($H_G(\sigma'_n) = 4$) is clearly to increase $H_G(\sigma_S + \sigma'_n)$ from zero to 4. But if the actual signal entropy is $H_G(\sigma_S) = 6$ bits/sample, the impact of a $H_G(\sigma'_n) = 4$ bits/sample noise signal is to raise the overall entropy $H_G(\sigma_S + \sigma'_n)$ by a comparatively insignificant 0.3 bit/sample.

AVIRIS Image

Consider again the plot of prediction entropy for a flat-field corrected 224-spectral-band AVIRIS image in Fig. 42. This is essentially a plot of "signal entropy" over all the individual bands. The entropy of a measured average prediction error distribution was 4.9 bits/sample. At such a high value, the impact of gain/offset noise with entropies of as much as 4 bits/sample would, by Fig. D-3, be generally less than about 0.5 bit/sample. But in reality, the impact would be higher.

Many of the spectral bands exhibit signal entropies much lower than 4.9 bits/sample. For example, at a signal entropy of $H_G(\sigma_s) = 3$ bits/sample, a noise with $H_G(\sigma'_n) = 4$ bits/sample would increase the overall entropy, $H_G(\sigma_s + \sigma'_n)$, by about 1.6 bits/sample. Clearly, this represents a significant reduction in coding efficiency for the spectral band that has a 3 bits/sample signal entropy.

Equalizing entropies. Note that the addition of significant levels of noise to all AVIRIS bands will tend to equalize the individual entropies. Suppose that without noise, entropies varied from 1 to 6 bits/sample, a spread of 5 bits/sample. With the addition of noise with, say, $H_G(\sigma'_n) = 5$ bits/sample, the overall entropies would vary over the range $5.1 \leq H_G(\sigma_s + \sigma'_n) \leq 6.6$ bits/sample, a spread of only 1.5 bits/sample.

REFERENCES

1) R. F. Rice, "The Rice Machine," **JPL 900-408** (internal document), Jet Propulsion Laboratory, Pasadena, California, September 1, 1970.

2) R. F. Rice and J. R. Plaunt, "Adaptive Variable Length Coding for Efficient Compression of Spacecraft Television Data," **IEEE Trans. of Comm. Technol.**, Vol. COM-19, Part I, December 1971, pp. 889–897.

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

4) R. F. Rice, "Practical Universal Noiseless Coding," **SPIE Seminar Proceedings**, vol. 207, San Diego, California, August 1979.

5) R. F. Rice and J. Lee, "Some Practical Universal Noiseless Coding Techniques – Part II," **JPL Publication 83-17**, Jet Propulsion Laboratory, Pasadena, California, March 1, 1983.

6) R. F. Rice, "Universal Noiseless Coding Techniques (Electronic Slide Show)," **NASA Workshop on Data Compression**, Snowbird, Utah, May 1988.

7) R. F. Rice, Pen-Shu Yeh, and Warner Miller, "Algorithms for a Very High Speed Universal Noiseless Coding Module," **JPL Publication 91-1**, Jet Propulsion Laboratory, Pasadena, California, February 15, 1991.

8) Jack Venbrux, et al., "A Very High Speed Compression/Decompression Chip Set," **Proceedings IEEE Data Compression Conference**, Snowbird, Utah, April 1991; and **JPL Publication 91-13**, Jet Propulsion Laboratory, Pasadena, California, July 15, 1991.

9) R. Anderson, et al., A Very High Speed Data Compression Chip for Space Imaging Applications," **Proceedings IEEE Data Compression Conference**, Snowbird, Utah, April 1991.

10) D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," **Proc. IRE**, vol. 40, pp. 1098-1101, 1952.

11) Pen-shu Yeh, Robert Rice, and Warner Miller, "On the Optimality of Code Options for a Universal Noiseless Coder," **JPL Publication 91-2**, Jet Propulsion Laboratory, Pasadena, California, February 15, 1991.

12) J. Lee and K. Y. Liu, Private Communication, Jet Propulsion Laboratory, Pasadena, California.

13) R. F. Rice, et al., "Block Adaptive Rate Controlled Image Data Compression," **Proceedings 1979 National Telecommunications Conference**, Washington, D.C., November 1979.

14) A. Goetz and M. Herring, "The High Resolution Imaging Spectrometer (HIRIS) for EOS," **IEEE Transactions of Geoscience and Remote Sensing**, vol. 27, pp. 136–144, 1989.

15) R. Green, "Proceedings of the Second Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) Workshop," **JPL Publication 90-54**, Jet Propulsion Laboratory, Pasadena, California, June 4 and 5, 1990.

16) S. Dolinar, "Maximum Entropy Probability Distributions under L_p Norm Constraints," **TDA Progress Report 42-104**, vol. October–December 1990, Jet Propulsion Laboratory, Pasadena, California, February 15, 1991.

17) Robert Nathan, Private Communication, JPL Image Processing Lab, Jet Propulsion Laboratory, Pasadena, California.