CHAPTER 16

The CCSDS Lossless Data Compression Recommendation for Space Applications[1]

PEN-SHU YEH

16.1 INTRODUCTION

In the late 1980s, when the author started working at the Goddard Space Flight Center for the National Aeronautics and Space Administration (NASA), several scientists were in the process of formulating the next generation of Earth-viewing science instruments, such as the Moderate Resolution Imaging Spectroradiometer. The instrument would have over 30 spectral bands and would transmit an enormous amount of data through the communications channel. In an attempt to solve the problem associated with buffering a large amount of data on board and transmitting it to ground stations, the author was assigned the task of investigating lossless compression algorithms for space implementation to compress science data in order to reduce the requirement on bandwidth and storage.

Space-based algorithm implementations face extra constraints relative to ground-based implementations. The difference is mainly to the lack of computing resources in space and possible bit errors incurred in the communication channel. A spacecraft is a physically compact environment where each subsystem (power, attitude control, instrument, data handling, telemetry, etc.) is given only limited budget in terms of power, mass, and weight. Any computation other than that which is absolutely necessary is normally not supported. Taking all these constraints into consideration, a set of requirements was first formulated for selecting a lossless compression algorithm:

a. The algorithm must adapt to changes in data statistics in order to maximize compression performance.
b. The algorithm must be implemented in real time with small memory and little power usage.

[1] Author's note: Part of the material from this chapter is taken from the Consultative Committee for Space Data Systems (CCSDS) publications [1, 2] for which the author is a major contributor.

c. The algorithm must interface with a packet data system such that each packet can be independently decoded without requiring information from other packets.

At the time of the study (early 1990s), the "real time" requirement in (b) required over 7.5 million samples/s for a 30 frames/s 512^2 charge-coupled device sensor. Requirement (c) allows a long scan-line of pixels to be compressed independently into a data packet, which will then be protected by an error control coding scheme. This requirement constrains error propagation within a packet when an uncorrectable channel error occurs.

Several available algorithms were evaluated on test science data: Huffman [3], Ziv–Lempel (see Ref. [4]), arithmetic [5], and Rice [6]. The excellent compression performance and the high throughput rate of the Rice algorithm suggested further study, which resulted in a mathematical proof [7] of its performance. The proof establishes the Rice algorithm as a type of adaptive Huffman code on decorrelated data with a Laplacian distribution. It so happens that most of the data collected on science instruments, be it one- or two-dimensional, or even multispectral, once decorrelated, fits a Laplacian distribution.

Further study on the algorithm brought out a parallel architecture which was then implemented in an application-specific integrated circuit for space applications [8]. An extension to low-entropy data was also devised and incorporated in the original Rice architecture. The resulting algorithm is referred to as the extended_Rice, or e_Rice, algorithm.

In 1994, with sufficient interest from the international space agencies, the e_Rice algorithm was proposed to the CCSDS subpanel as a candidate for recommendation as a standard. In 1997, CCSDS published the Blue Book [1], which adopted the e_Rice algorithm as the recommendation for the standard. A Green Book [2], which serves as an application guide, was also released.

16.2 THE e_Rice ALGORITHM

The e_Rice algorithm exploits a set of variable-length codes to achieve compression. Each code is nearly optimal for a particular geometrically distributed source. Variable-length codes, such as Huffman codes and the codes used by the Rice algorithm, compress data by assigning shorter codewords to symbols that are expected to occur with higher frequency, as illustrated in Section 16.3.1. By using several different codes and transmitting the code identifier, the e_Rice algorithm can adapt to many sources from low entropy (more compressible) to high entropy (less compressible). Because blocks of source samples are encoded independently, side information does not need to be carried across packet boundaries; if decorrelation of source samples is executed only among these blocks, then the performance of the algorithm is independent of packet size.

A block diagram of the e_Rice algorithm is shown in Fig. 16.1. It consists of a preprocessor to decorrelate data samples and subsequently map them into symbols suitable for the entropy coding stage. The input data to the preprocessor, $\boldsymbol{x}$, is a J-sample block of n-bit samples:

$$\boldsymbol{x} = x_1, x_2, \ldots, x_J.$$

The preprocessor transforms input data into blocks of preprocessed samples, δ, where

$$\delta = \delta_1, \delta_2, \ldots, \delta_J.$$

The Adaptive Encoder converts preprocessed samples, δ, into an encoded bit sequence $\boldsymbol{y}$.

The entropy coding module is a collection of variable-length codes operating in parallel on blocks of J preprocessed samples. The coding option achieving the highest compression is

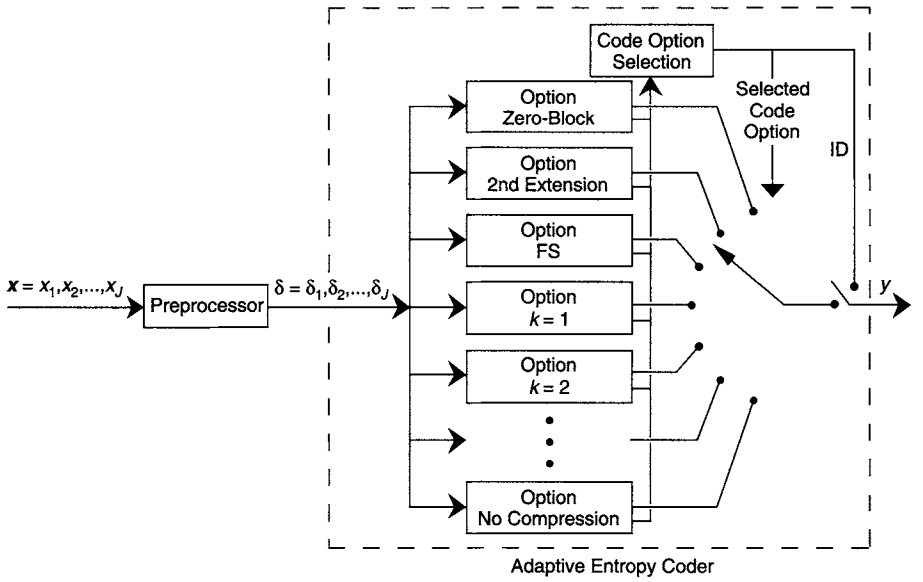


FIGURE 16.1
The encoder architecture.

selected for transmission, along with an ID bit pattern used to identify the option to the decoder. Because a new compression option can be selected for each block, the e_Rice algorithm can adapt to changing source statistics. Although the CCSDS Recommendation specifies that the parameter J be either 8 or 16 samples per block, the preferred value is 16. The value of 16 samples per block is the result of experiments performed on several classes of science data, both imaging and non-imaging. These studies monitored the achievable compression ratio as a function of the parameter J, which was set to 8, 16, 32, and 64 samples/block for the various classes of science data. Values of J less than 16 result in a higher percentage of overhead, which yields a lower compression ratio, whereas values of J higher than 16 yield low overhead but have less adaptability to variations in source data statistics.

16.3 THE ADAPTIVE ENTROPY CODER

16.3.1 Fundamental Sequence Encoding

To see how a variable-length code can achieve data compression, consider two methods of encoding sequences from an alphabet of four symbols:

Symbol	Probability	Code 1	Code 2
s_1	0.6	00	1
s_2	0.2	01	01
s_3	0.1	10	001
s_4	0.1	11	0001

Code 2 is known as the "comma" code; it is also defined as the Fundamental Sequence (FS) code.

Table 16.1 Fundamental Sequence Codewords as a Function of the Preprocessed Samples

Preprocessed Sample Values, δ_i	FS Codeword
0	1
1	01
2	001
⋮	⋮
$2^n - 1$	$\underbrace{0000\ldots00001}_{(2^n - 1 \text{ zeros})}$

Table 16.2 Examples of Split-Sample Options Using Fundamental Sequence Codes

Sample Values	4-bit Binary Representation	FS Code, $k = 0$	$k = 1$ 1 LSB + FS Code	$k = 2$ 2 LSB + FS Code
8	1000	000000001	0 00001	00 001
7	0111	00000001	1 0001	11 01
1	0001	01	1 1	01 1
4	0100	00001	0 001	00 01
2	0010	001	0 01	10 1
5	0101	000001	1 001	01 01
0	0000	1	0 1	00 1
3	0011	0001	1 01	11 1
Total bits	32	38	29	29

Using either code, encoded bitstreams corresponding to symbol sequences are uniquely decodable. For Code 1, we recognize that every two encoded bits correspond to a symbol. For the FS code we recognize that each "1" digit signals the end of a codeword, and the number of preceding zeros identifies which symbol was transmitted. This simple procedure allows FS codewords to be decoded without the use of lookup tables.

The reason that the FS code can achieve compression is that when symbol s_1 occurs very frequently, and symbols s_3 and s_4 are very rare, on average we will transmit fewer than two encoded bits per symbol. In the above example, Code 2 will achieve an average codeword length of $(0.6 + 2 \times 0.2 + 3 \times 0.1 + 4 \times 0.1) = 1.7$ bits per symbol, whereas Code 1 always requires two encoded bits per symbol.

Longer FS codes achieve compression in a similar manner. Table 16.1 illustrates the FS codewords for preprocessed sample values with an n-bit dynamic range.

16.3.2 The Split-Sample Option

Most of the options in the entropy coder are called "split-sample options." The kth split-sample option takes a block of J preprocessed data samples, splits off the k least significant bits (LSBs) from each sample, and encodes the remaining higher order bits with a simple FS codeword before appending the split bits to the encoded FS datastream. This is illustrated in Table 16.2 for the case of k split bit being 0 (no split-bit), 1, or 2 on a sequence of 4-bit samples.

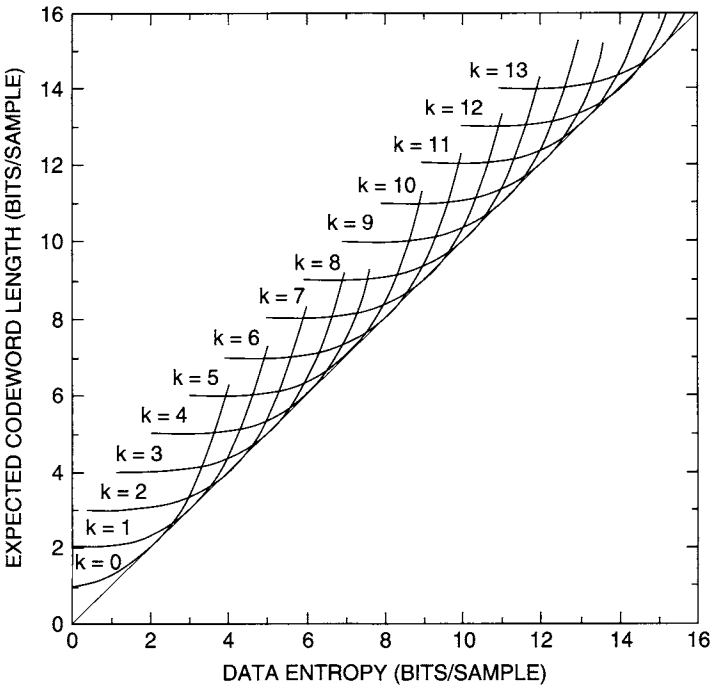


FIGURE 16.2
Performance curve for k.

From Table 16.2 either $k = 1$ or $k = 2$ will achieve data reduction from the original 32 bits to 29 bits. As a convention, when a tie exists, the option with smaller k value is chosen. In this case, $k = 1$ will be selected. When a block of J samples is coded with one split-sample option, the k split bits from each sample are concatenated using the data format specified in Section 16.5.

Each split-sample option in the Rice algorithm is designed to produce compressed data with an increment in the codeword length of about 1 bit/sample (approximately $k + 1.5$ to $k + 2.5$ bits/sample); the code option yielding the fewest encoded bits will be chosen for each block by the option-select logic. This option selection process ensures that the block will be coded with the best available code option on the same block of data, but this does not necessarily indicate that the source entropy lies in that range. The actual source entropy value could be lower; the source statistics and the effectiveness of the preprocessing stage determine how closely entropy can be approached.

A theoretical expected codeword length of the split-sample coding scheme is shown in Fig. 16.2 for various values of k, where $k = 0$ is the fundamental sequence option. These curves are obtained under the assumption that the source approaches a discrete geometric distribution. With this source model, tracing only the portions of these curves that are closest to the ideal diagonal performance curve at any given entropy will result in a theoretically projected performance curve as shown in Fig. 16.3. For a practical system as shown in Fig. 16.1, ID bits of 3- or 4-bit length will be needed for every block of J samples for cases of an 8-option or a 16-option system. In such a system, the source is likely to deviate from the discrete geometric distribution; however, the option with the fewest coded bits will be selected. Actual coding results on aerial imagery, along with the coder's applicability to other data sources with a Gaussian or a Poisson probability distribution function, are provided in Ref. [9].

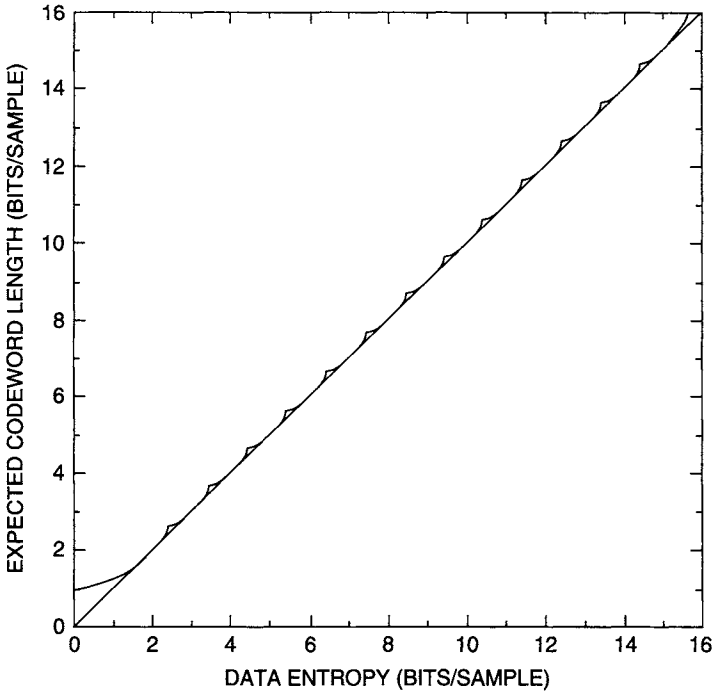


FIGURE 16.3
Effective performance curve.

16.3.3 Low-Entropy Options

16.3.3.1 General

The two code options, the Second-Extension option[2] and the Zero-Block option, provide more efficient coding than other options when the preprocessed data is highly compressible.

16.3.3.2 The Second-Extension Option

When the Second-Extension option is selected, each pair of preprocessed samples in a J-sample block is transformed and encoded using an FS codeword. Let δ_i and δ_{i+1} be adjacent pairs of samples from a J-sample preprocessed data block. They are transformed into a single new symbol, γ, by the following equation:

$$\gamma = (\delta_i + \delta_{i+1})(\delta_i + \delta_{i+1} + 1)/2 + \delta_{i+1}.$$

The $J/2$ transformed symbols in a block are encoded using the FS codeword of Table 16.1. The above process requires J to be an even integer which the recommended values obey ($J = 8$ or 16).

16.3.3.3 Zero-Block Option

The idea of the Zero-Block option was first suggested by J. Venbrux at the University of Idaho's Microelectronics Research Laboratory to deal with long runs of 0's. It is selected when one or

[2] The first extension of a preprocessed sample is the preprocessed sample itself.

Table 16.3 Zero-Block Fundamental Sequence Codewords as a Function of the Number of Consecutive All-Zeros Blocks

Number of All-Zeros Blocks	FS Codeword
1	1
2	01
3	001
4	0001
ROS	00001
5	000001
6	0000001
7	00000001
8	000000001
⋮	⋮
63	0000...0000000001 (63 0's and a 1)

more blocks of preprocessed samples are all zeros. In this case, a single codeword may represent several blocks of preprocessed samples, unlike other options where an FS codeword represents only one or two preprocessed samples.

The set of blocks between consecutive reference samples, r, as described in Section 16.4.2, is partitioned into one or more segments. Each segment, except possibly the last, contains s blocks. The recommended value of s is 64.

Within each segment, each group of adjacent all-zeros blocks is encoded by the FS codewords, specified in Table 16.3, which identify the length of each group. The Remainder-of-Segment (ROS) codeword in Table 16.3 is used to indicate that the remainder of a segment consists of five or more all-zeros blocks.

16.3.4 No Compression

The last option is not to apply any data compression. If it is the selected option, the preprocessed block of samples receives an attached identification field but is otherwise unaltered.

16.3.5 Code Selection

The Adaptive Entropy Coder includes a code selection function, which selects the coding option that performs best on the current block of samples. The selection is made on the basis of the number of bits that the selected option will use to code the current block of samples. An ID bit sequence specifies which option was used to encode the accompanying set of codewords. The ID bit sequences are shown in Table 16.4 for a sample dynamic range up to 32 bits.

For applications not requiring the full entropy range of performance provided by the specified options, a subset of the options at the source may be implemented. The ID specified in Table 16.4 is always required and suggested, even if a subset of the options is used. With this ID set, the same standard decoder can be used for decoding partial implementations of the encoder.

Table 16.4 Selected Code Option Identification Key

Option	$n \leq 8$	$8 < n \leq 16$	$16 < n$
Zero-Block	0000	00000	000000
Second-Extension	0001	00001	000001
FS	001	0001	00001
$k = 1$	010	0010	00010
$k = 2$	011	0011	00011
$k = 3$	100	0100	00100
$k = 4$	101	0101	00101
$k = 5$	110	0110	00110
$k = 6$	—	0111	00111
$k = 7$	—	1000	01000
$k = 8$	—	1001	01001
$k = 9$	—	1010	01010
$k = 10$	—	1011	01011
$k = 11$	—	1100	01100
$k = 12$	—	1101	01101
$k = 13$	—	1110	01110
$k = 14$	—	—	01111
$k = 15$	—	—	10000
⋮	⋮	⋮	⋮
$k = 29$	—	—	11110
No compression	111	1111	11111

Note: A dash indicates no applicable value.

16.4 PREPROCESSOR

The role of the preprocessor is to transform the data into samples that can be more efficiently compressed by the entropy encoder. To ensure that compression is "lossless," the preprocessor must be reversible. The FS and sample-split options work on non-negative integer values. The most effective preprocessor will transform integer data values into all non-negative values with a unimodal distribution resembling a one-sided Lapace function. The example in Section 16.3.1 shows that to achieve effective compression, we need a preprocessing stage that transforms the original data so that shorter codewords occur with higher probability than longer codewords. If there are several candidate preprocessing stages, we would like to select the one that produces the shortest average codeword length.

In general a preprocessor that removes correlation between samples in the input data block will improve the performance of the entropy coder. In the following discussion, we assume that preprocessing is done by a predictor followed by a prediction error mapper. For some types of data, more sophisticated transform-based techniques can offer improved compression efficiency, at the expense of higher complexity.

16.4.1 Predictor

The decorrelation function of the preprocessor can be implemented by a judicious choice of predictor for an expected set of data samples. The selection of a predictor should take into account the expected data as well as possible variations in the background noise and the gain of

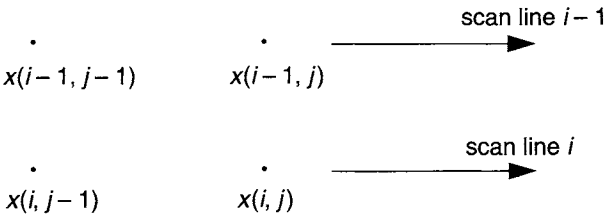


FIGURE 16.4
Typical pixel data arrangement.

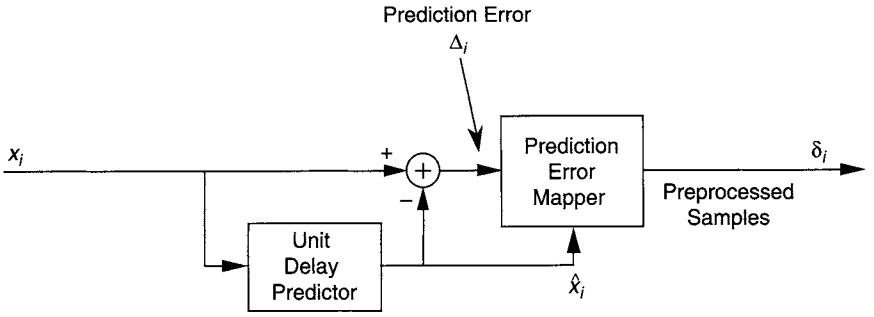


FIGURE 16.5
Preprocessor using a unit-delay predictor.

the sensors acquiring the data. The predictor should be chosen to minimize the amount of noise resulting from sensor non-uniformity.

Given the data arrangement in Fig. 16.4, the predictor value $\hat{x}(i, j)$ of input value $x(i, j)$ can be

(a) A one-dimensional first-order predictor: $\hat{x}(i, j) = x(i, j-1)$, or $\hat{x}(i, j) = x(i-1, j)$.

(b) A two-dimensional predictor: $\hat{x}(i, j) = 1/2[x(i, j-1) + x(i-1, j)]$.

(c) A two-dimensional predictor: $\hat{x}(i, j) = 1/8[3x(i, j-1) + 3x(i-1, j) + 2x(i-1, j-1)]$.

Other types of weighted predictor can be devised depending on applications. For multispectral data, another prediction technique is to use samples from one spectral band as an input to a higher order prediction function for the next band.

One of the simplest predictive coders is a linear first-order unit-delay predictor shown in Fig. 16.5. The output, Δ_i, will be the difference between the input data sample and the preceding data sample.

16.4.2 Reference Sample

It is clear from Fig. 16.5 that for a reversible operation, a reference sample is needed to perform the first prediction. A reference sample is an unaltered data sample upon which successive predictions are based. Reference samples are required by the decoder to recover the original values from difference values. In cases where a reference sample is not used in the preprocessor or where

the preprocessor is absent, it shall not be inserted. The user must determine how often to insert references. When required, the reference must be the first sample of a block of J input data samples. In packetized formats, the reference sample must be in the first Coded Data Set (CDS) in the packet data field, as defined in Section 16.5, and must be repeated after every r blocks of data samples.

16.4.3 Prediction Error Mapper

Based on the predicted value, $\hat{x}_i$, the prediction error mapper converts each prediction error value, Δ_i, to an n-bit non-negative integer, δ_i, suitable for processing by the entropy coder. For most efficient compression by the entropy coding stage, the preprocessed symbols, δ_i, should satisfy

$$p_0 \geq p_1 \geq p_2 \geq \cdots p_j \geq \cdots p_{(2^n-1)},$$

where p_j is the probability that δ_i equals integer j. This ensures that more probable symbols are encoded with shorter codewords.

The following example illustrates the operation of the prediction error mapper after a unit-delay predictor is applied to 8-bit data samples of values from 0 to 255:

Sample Value x_i	Predictor Value $\hat{x}_i$	Δ_i	θ_i	δ_i
101	—	—	—	—
101	101	0	101	0
100	101	−1	101	1
101	100	1	100	2
99	101	−2	101	3
101	99	2	99	4
223	101	122	101	223
100	223	−123	32	155

If we let $x_{\min}$ and $x_{\max}$ denote the minimum and maximum values of any input sample x_i, then clearly any reasonable predicted value $\hat{x}_i$ must lie in the range $[x_{\min}, x_{\max}]$. Consequently, the prediction error value Δ_i must be one of the 2^n values in the range $[x_{\min} - \hat{x}_i, x_{\max} - \hat{x}_i]$, as illustrated in Fig. 16.6. We expect that for a well-chosen predictor, small values of $|\Delta_i|$ are more likely than large values, as shown in Fig. 16.7. Consequently, the prediction error mapping function

$$\delta_i = \begin{cases} 2\Delta_i & 0 \leq \Delta_i \leq \theta \\ 2|\Delta_i|-1 & -\theta \leq \Delta_i < 0 \\ \theta + |\Delta_i| & \text{otherwise,} \end{cases}$$

where $\theta = \text{minimum}\,(\hat{x}_i - x_{\min}, x_{\max} - \hat{x}_i)$, has the property that $p_i < p_j$ whenever $|\Delta_i| > |\Delta_j|$. This property increases the likelihood that the probability ordering of p_i (described above) is satisfied.

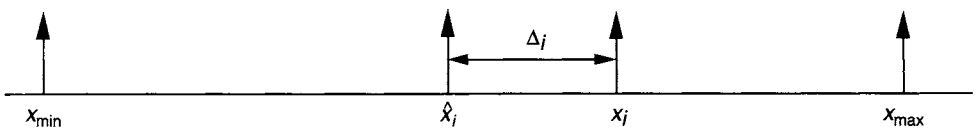


FIGURE 16.6
Relative position of sample value x_i and predictor value $\hat{x}_i$.

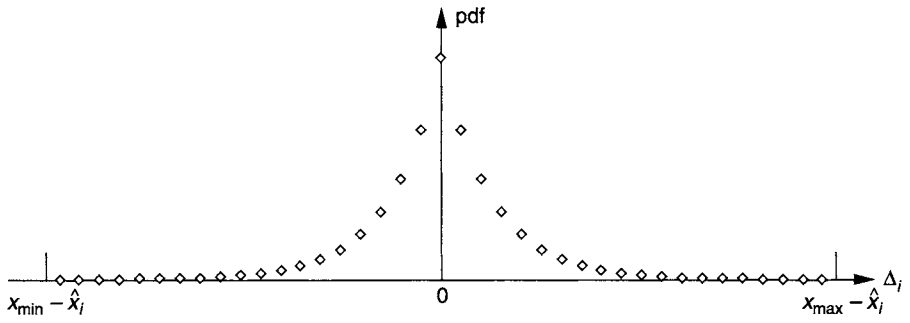


FIGURE 16.7
Typical probability distribution function (pdf) of Δ_i for imaging data.

16.5 CODED DATA FORMAT

The coded bits from each block of J samples follow a predefined format so that cross-support can be achieved among users. A CDS contains these coded bits from one block. The formats of a CDS under different conditions are given in Figs. 16.8a–16.8d for the case when a reference sample is used for encoding. For cases when reference sample is not needed, it will not be included in the coded bitstream, and the CDS fields in Fig. 16.8 which contain $J - 1$ will contain J instead.

For the Second-Extension option, in the case when a reference is inserted, a "0" sample is added in front of the $J - 1$ preprocessed samples, so $J/2$ samples are produced after the transformation. When no reference sample is needed, then the n-bit reference field is not used.

16.6 DECODING

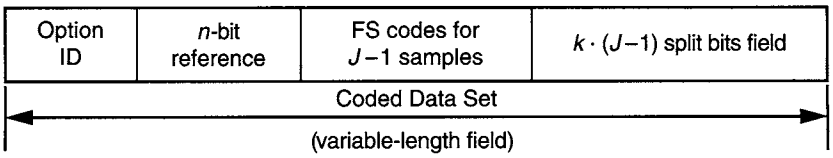
The lossless decoder consists of two separate functional parts, the postprocessor and the adaptive entropy decoder, as shown in Fig. 16.9. The postprocessor performs both the inverse prediction operation and the inverse of the standard mapper operation. The first step toward decoding is to set up the configuration parameters so that both the encoder and the decoder operate in the same mode. These configuration parameters are mission/application specific and are either known *a priori* or are signaled in the Compression Identification Packet defined in [1] for space missions.

The configuration parameters common to both encoder and decoder are as follows:

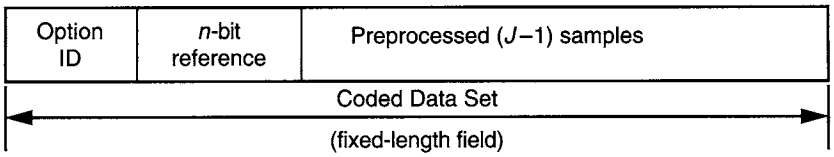
- resolution in bits per sample ($n = 1, \ldots, 32$);
- block size in samples ($J = 8$ or 16);
- number of CDSs contained in a packet ($l = 1, \ldots, 4096$);
- number of CDSs between references ($r = 1, \ldots, 256$);
- predictor type if present;
- sense of digital signal value (positive or bipolar).

For applications such as compressing medical imaging data or image files in a ground archive where packet data structure is not always used for error protection, these decoding parameters should be included in the compressed file as header information. Then Packet Start and Stop in Fig. 16.9 are not needed. The number of CSDs in a scan-line or the number of pixels in a scan-line can be used to decode to the correct number of data samples.

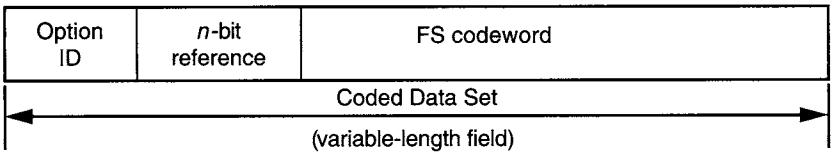
Decoding the coded bits is straightforward once the ID bits are interpreted since the coding is done using FS as the basic structure. Only the second-extension code needs an additional short



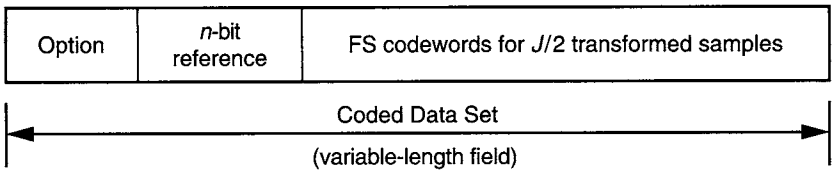
(a) CDS format with sample-splitting and reference



(b) CDS format when no-compression option is selected



(c) CDS format when zero-block option is selected



(d) CDS format when 2nd-extension option is selected

FIGURE 16.8
Format for a coded data set.

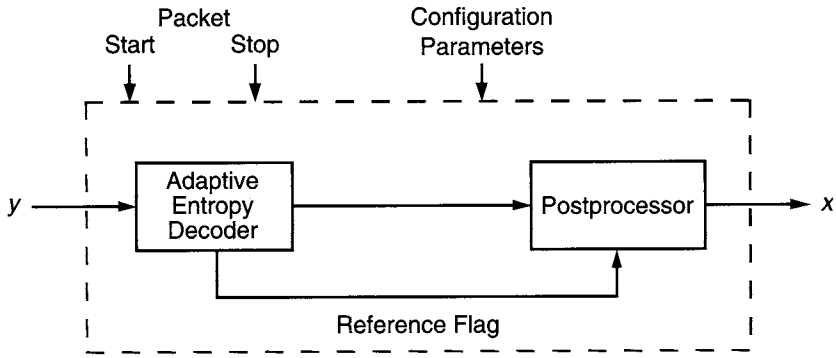


FIGURE 16.9
Decoder block diagram.

Table 16.5 Decoding Logic for the Second-Extension Option

m	β	m_s
0	0	0
1, 2	1	1
3, 4, 5	2	3
6, ..., 9	3	6
10, ..., 14	4	10
15, ..., 20	5	15
21, ..., 27	6	21
28, ..., 35	7	28

lookup table to replace computation. For this option, $J/2$ FS codewords will be extracted first. These are then decoded using the following steps:

Step 1. Obtain m by counting the 0's of the FS codeword.
Step 2. Obtain β and m_s using Table 16.5 logic.
Step 3. Calculate $\delta_{i+1} = m - m_s$.
Step 4. Calculate $\delta_i = \beta - \delta_{i+1}$.

For the Zero-Block option, the single FS codeword following the ID and reference indicates the number of all-zeros blocks or the ROS condition listed in Table 16.3. Using this table, an appropriate number of all-zeros blocks are generated as input to the postprocessor.

The postprocessor reverses the mapper function, given the predicted value $\hat{x}_i$. When the preprocessor is not used during encoding, the postprocessor is bypassed as well. The inverse mapper function can be expressed as

$$\text{if } \delta_i \le 2\theta, \quad \Delta_i = \begin{cases} \delta_i/2 & \text{when } \delta_i \text{ is even} \\ -(\delta_i + 1)/2 & \text{when } \delta_i \text{ is odd} \end{cases}$$

$$\text{if } \delta_i > 2\theta, \quad \Delta_i = \begin{cases} \delta_i - \theta & \text{when } \theta = \hat{x}_i - x_{\min} \\ \theta - \delta_i & \text{when } \theta = x_{\max} - \hat{x}_i, \end{cases}$$

where $\theta = \text{minimum}(\hat{x}_i - x_{\min}, x_{\max} - \hat{x}_i)$.

There will be J (or $J - 1$ for a CDS with a reference sample) prediction error values Δ_1, $\Delta_2, \ldots, \Delta_J$ generated from one CDS. These values will be used in the postprocessor to recover the J-sample values $x_1, x_2, \ldots, x_J$.

To decode packets that may include fill bits, two pieces of information must be communicated to the decoder. First, the decoder must know the block size, J, and the number of CDSs in a packet, l. Second, the decoder must know when the data field in the packet begins and when it ends. From these two pieces of information, the number of fill bits at the end of the data field can be determined and discarded.

The number of valid samples in the last CDS in the packet may be less than J. If so, the user will have added fill samples to the last CDS to make a full J-sample block, and the user must extract the valid samples from the last J-sample block. It is foreseeable that an implementation can be done with variable size for the last block, but that would be outside the scope of the current CCSDS recommendation.

16.7 TESTING

A set of test data for a different data dynamic range, n, has been created. This test set can be used to verify the encoder implementation. This limited set of test vectors can be obtained from the software that is downloadable from the CCSDS Panel-1 home page: http://www.ccsds.org/ccsds/p1_home.html. This data set will cause all the split-sample options, the no-compression option, and the Second-Extension option to be selected once. The Zero-Block option will be selected at least once for up to $n = 14$ when only 256 data points are requested. If the user allows more than 256 test data points to be written, then the Zero-Block option will be exercised for all n up to $n = 16$. Current test programs generate test data for resolution from $n = 4$ to $n = 16$ bits.

16.8 IMPLEMENTATION ISSUES AND APPLICATIONS

For space applications, several systems issues relating to embedding data compression on board a spacecraft have been addressed. These include onboard packetization for error containment, buffering for constant but bandwidth-limited downlink rate, sensor response calibration, and compression direction for optimized compression performance [2].

For applications on ground, normally the requirement is much more relaxed; most applications do not use any packet data structure with added channel coding to protect from transmission or storage bit errors. There is also no need to impose a limit on buffering either input or compressed data. To achieve the best performance, users only have to engineer a preprocessor that would decorrelate the input signal into a non-negative integer signal that resembles a Laplace distribution. In certain situations the knowledge of how data are acquired from various sensors and formatted can be extremely useful in optimizing compression performance. An example is data collected from an array of different types of sensors (e.g., strain gauge, thermometer, accelerometer, fuel gauge, pressure gauge) from spacecraft or any other machinery. This data is commonly arranged in sets in the sequence of time steps T_i when data is collected, shown in Fig. 16.10.

It is customary to first try to apply a compression algorithm directly on the time sequence of the data as it is collected in such manner. The resulting poor performance can be attributed to the different characteristics of these sensors. To optimize performance, compression should be applied to data collected from each sensor in a time sequence independent of other sensors as shown in Fig. 16.11.

In an effort to broaden the domain of applications, this algorithm has been applied to processed science data such as sea surface temperature and vegetation index in floating-point representations and achieved over 2:1 lossless compression as well. This can be accomplished by judiciously executing compression in the direction shown in Fig. 16.12. In this scenario, the preprocessor will

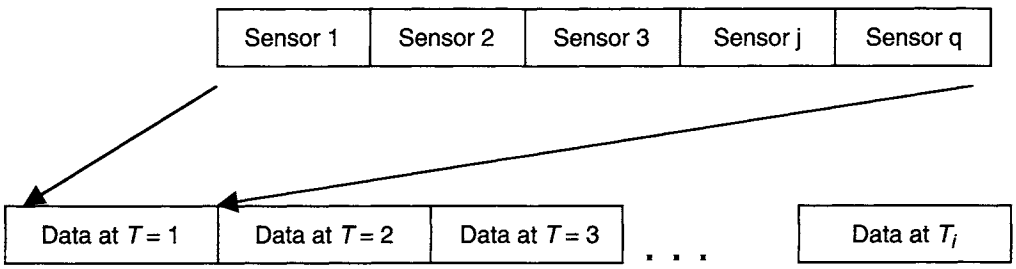


FIGURE 16.10
Common sensor data arrangement at different time steps.

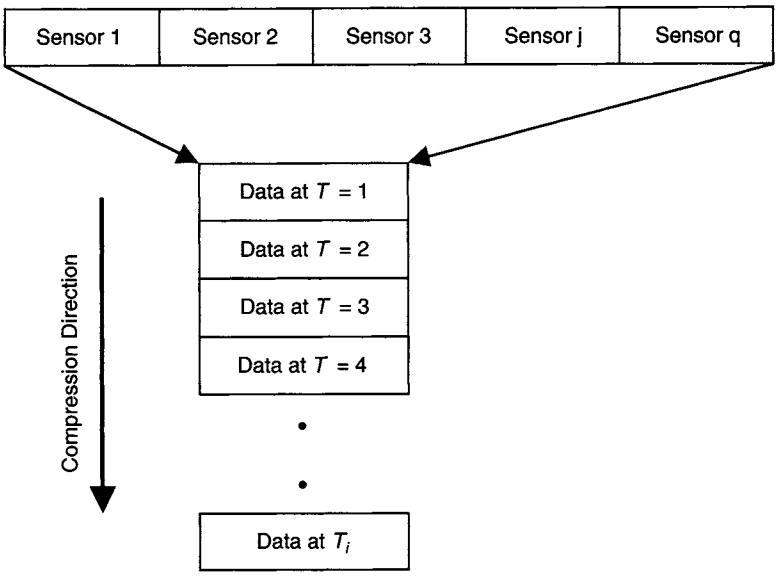


FIGURE 16.11
Optimal compression scheme.

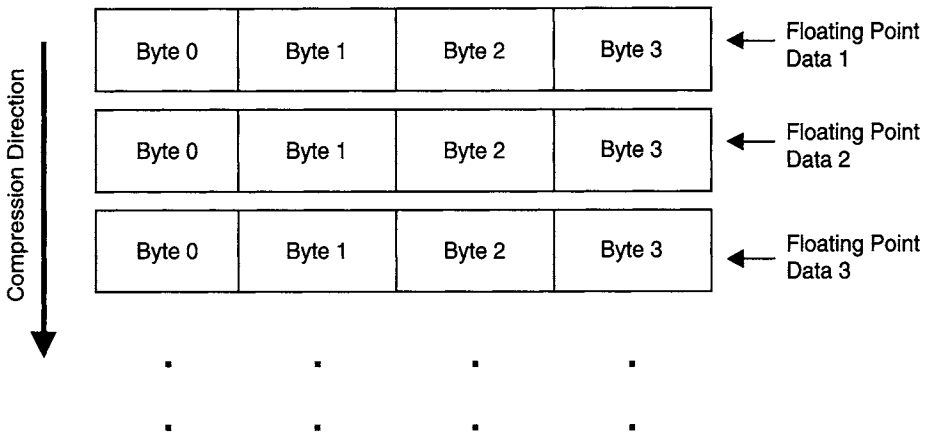


FIGURE 16.12
Compressing floating-point science data.

take the 4 bytes from one floating-point number as if they were from four different data sources. Each data source now has data of 1-byte (value 0–255) dynamic range. A unit delay predictor is then used before the entropy coder.

The scheme in Fig. 16.12 can be implemented with four buffer arrays, each holding the ith byte (i being 0, . . . , 3) from the floating-point data. The length of the array can be fixed at the length corresponding to one scan-line or any number of J-sample blocks. The compressed bits will then be concatenated at the end of compressing each array. For double precision or complex-valued science data with 8 bytes per data point, the same scheme can be used.

The CCSDS e_Rice compression scheme has been utilized on medical imaging data with good results [10]. Applications also have found their way into science data archives, seismic data, acoustic data, and high-energy particle physics as well.

16.9 ADDITIONAL INFORMATION

1. One source of available high-speed integrated circuits and software for the e_Rice algorithm is www.cambr.uidaho.edu.
2. The Second-Extension option in the e_Rice algorithm is protected by U.S. Patent 5,448,642 assigned to NASA. For implementation for commercial usage, contact the technology commercialization office at the Goddard Space Flight Center in Greenbelt, Maryland.

DEDICATION

The author dedicates this chapter to the memory of Warner H. Miller, whose vision initiated the technology development of the e_Rice algorithm and whose encouragement helped achieve its completion.

16.10 REFERENCES

1. *Lossless Data Compression*. 1997. Consultative Committee for Space Data Systems CCSDS 121.0-B-1 Blue Book, May 1997.
2. *Lossless Data Compression*. 1997. Consultative Committee for Space Data Systems CCSDS 120.0-G-1 Green Book, May 1997.
3. Huffman, D. A., 1952. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, Vol. 40, pp. 1098–1101.
4. Welch, T. A., 1984. A technique for high-performance data compression. *IEEE Computer*, Vol. 17, No. 6, pp. 8–19, June 1984.
5. Witten, I. H., R. M. Neal, and J. G. Cleary, 1987. Arithmetic coding for data compression. *Communications of the ACM*, Vol. 30, No. 6, pp. 520–540. June 1987.
6. Rice, R. F., 1979. Practical universal noiseless coding. In *Proceedings of the SPIE Symposium*, Vol. 207, San Diego, CA, August 1979.
7. Yeh, P.-S., R. F. Rice, and W. H. Miller, 1993. On the optimality of a universal noiseless coder. In *Proceedings of the AIAA Computing in Aerospace 9 Conference*, San Diego, CA, October 1993.
8. Venbrux, J., P.-S. Yeh, and M. N. Liu, 1992. A VLSI chip set for high-speed lossless data compression. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 2, No. 4, December 1992.
9. Yeh, P.-S., R. F. Rice, and W. H. Miller, 1991. *On the Optimality of Code Options for a Universal Noiseless Coder*, NASA/JPL Publication 91-2, February 1991.
10. Venbrux, J., P.-S. Yeh, G. Zweigle, and J. Vessel, 1994. A VLSI chip solution for lossless medical imagery compression." In *Proceedings of the SPIE's Medical Imaging 1994*, Newport Beach, CA.