DATA COMPRESSION STRATEGIES FOR RDAT/DDS MEDIA IN HOSTILE ENVIRONMENTS

OWEN DAVID JOHN THOMAS

Division of Mathematics and Computing

A submission presented in partial fulfilment of the requirements of the University of Glamorgan/Prifysgol Morgannwg for the degree of Doctor of Philosophy

September 1996

Acknowledgments

This work was supported by the Higher Education Funding Council for Wales through their DEVR initiative.

I am grateful to D. Burke, late of British Gas, for helpful discussions in the early stages of this work.

I offer sincerest thanks to my supervisors, Alan Ryley and Derek Smith, for their steadfast faith and patience and for their skilful guidance of my research.

Declaration

This is to certify that neither this thesis nor any part of it has been presented or is being currently submitted in candidature for any degree other than the degree of Doctor of Philosophy of the University of Glamorgan.

Candidate.....Owen Thomas.....

Certificate of Research

This is to certify that, except where specific reference is made, the work presented in this thesis is the result of the investigation undertaken by the candidate.

Candidate.....

Director of Studies.....

Abstract

This thesis investigates the prevention of error propagation in magnetically recorded compressed data when severe environmental conditions result in uncorrected channel errors. The tape format DDS is examined and a computer simulation of its error correction procedures is described. This software implementation uses explicit parity byte equations and these are presented for all three Reed-Solomon codes. The simulation allows the calculation of the uncorrected error patterns when the recording is compromised and uncorrected byte errors are determined for given initial random and burst errors.

Some of the more familiar data compression algorithms are visited before the little known adaptive Rice algorithm is described in detail. An analytic example is developed which demonstrates the coding mechanism.

A synchronized piecewise compression strategy is adopted in which the synchronizing sequences are placed periodically into the compressed data stream. The synchronizing sequences are independent of the compression algorithm and may occur naturally in the compressed data stream. A cyclic count is added to the compressed data stream to number the groups of data between synchronizing sequences and prevent slippage in the data.

The Rice algorithm is employed in the strategy to compress correlated physical data. A novel compressor is developed to compress mixed correlated physical data and text within the synchronization strategy. This compressor uses the Rice algorithm to compress the correlated data and a sliding window algorithm to compress the text and switches between the two algorithms as the data type varies. The sliding window compressor LZB is adopted when the same principles are applied to the robust compression of English text alone. LZB is modified to improve compression of relatively small pieces of English text.

The synchronization strategy incorporating these algorithms has been simulated computationally. This simulation is linked to that of DDS in each test performed when the errors are both random and bursty. The decompressed data is compared to the original. The strategy is demonstrated to be effective in preventing error propagation beyond the data immediately affected by errors without significant damage to the compression ratio.

Contents

'I'd like two dozen eggs, seven pounds of potatoes, three loaves of bread, a cabbage and four large onions. Have you got that? It's not a lot so you don't have to take the car - the walk'll do you good'.

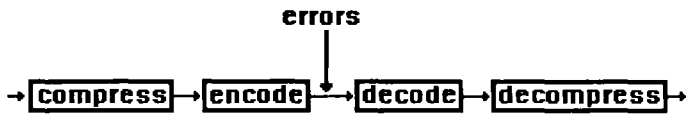
Some time later - 'I wish I'd written a list, this doesn't seem right at all; three dozen cabbages, two large eggs, four loaves of bread, seventeen pounds of potatoes and ... a radish! I wonder if they sell spinach.'

Chapter One

Introduction

1.1 Introduction

This thesis examines the effects of errors on coded information. The coding is a serial combination of data compression and coding for error control. Data compression extracts redundancy from the data leaving behind the information. Lossless compression allows the original data to be recovered intact at a later stage. Coding for error control requires the addition to the data, be it compressed or otherwise, of 'functions' of the data. The quantity of functions added determines the extent to which errors can be corrected. The whole of the examination reported here is summarized in the diagram below



The error control encoding and decoding of the digital tape medium DDS (Digital Data Storage) are 'givens' [1]. The challenge undertaken is to devise compression and decompression strategies which still perform effectively, or indeed perform at all, when errors remain uncorrected. This thesis offers, perhaps uniquely, an end-to-end *simulation* of the

above, in which techniques and algorithms of varying degrees of celebrity are combined in an unusual way.

1.2 Encode for Error Control before Compression?

The newcomer to this area of research, without the benefit of any work to which he might refer, may puzzle over the order in which data compression and error control occur. The following arguments should convince him that there can only be one order.

The two possibilities are shown below:

Case A

P Q R S T
-----> Compress -----> Encode -----> Decode -----> Decompress -->

Case B

P Q R S T
-----> Encode -----> Compress -----> Decompress -----> Decode -->

The errors are introduced into both systems at point R.

Let m(s) = The average number of errors output if s errors are input to decompressor - normally m(s)>s

$\phi(s)$ = The average number of errors output if s errors are input to decoder.

$\phi(0) = \phi(1) = \phi(2) \ldots = \phi(e) = 0$ i.e. the decoder can correct up to e errors.

Consider now the number of bits per 'block' following the data through the systems. CR1 and CR2 are the compression ratios (compression ratio = compressed size over uncompressed size) in case A and case B respectively (<1 in each case).

Case	P	Q	R	S	T
A	k/C1	k	n	k	k/C1
B	k/C2	n/C2	n	n/C2	k/C2

Argument 1 C2 > C1 because of a loss of semantic features on encoding and particularly interleaving e.g. loss of runs for run length encoding, loss of Ziv Lempel repetitions - the output of the encoder would be a binary string which would have to be compressed into another binary string for storage.

Argument 2 Consider random errors and assume C1 = C2. If there are t errors per block then

Output errors per block

t	0	1	2	3	e	e+1	e+2	e+δ
Case A	0	0	0	0...0 0.....	0	m(ϕ(e+1)	m(ϕ(e+2))	m(ϕ(e+δ))
Case B	0	0	0	0...0 ϕ(m(r+1))	ϕ(m(e))	ϕ(m(e+1))	ϕ(m(e+2))	ϕ(m(e+δ))

$m(1) < e \ldots m(r) \leq e$

where r represents the maximum number of errors which, when input into the decompressor, allows the number of propagated errors to fall within the error correction capability of the decoder.

The functions m and ϕ are both fairly unpredictable but in normal circumstances the results for t = r+1, r+2 ..... e give case A the advantage. Therefore both arguments show that case A is the better.

1.3 A Real Problem

DDS drives and tapes are an example of a proprietary recording technology that is both readily accessible and portable. Such factors help make DDS an important tool for the engineer. However, proprietary data storage devices are designed for use within a normal working environment such as an office or laboratory. Were the engineer to wish to capture, compress and record data remotely in a hostile environment, such as, for example, in an underground pipeline or on board a spacecraft, he would discover that these same devices do not operate with consistent reliability. The incidence of errors caused by the effects of phenomena such as vibration, moisture, temperature fluctuations, dust and radiation might cross the threshold of correctability. Any uncorrected channel errors arising out of the failure of the device to cope with the adverse conditions encountered could have devastating consequences for the compressed data. Conventional compression algorithms have an unfortunate tendency to propagate errors throughout all the data that is subsequently decompressed. This creates a need for a compression strategy which can be applied to the data independently of the recording system and which limits the propagation of errors.

1.4 Digital Data Storage (DDS)

DDS offers [1] high data density - a 60m tape, contained in a cartridge measuring approximately 3 in. by 2 in. by ½ in., is able to store 1.3 Gbyte of data. The integrity requirements of data for computer applications make effective error control essential. The DDS format adds further error control procedures to the digital audio format [2] of 4mm tape. Error

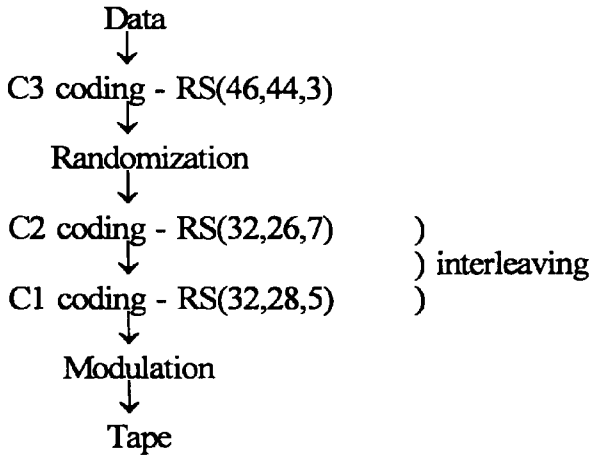
control of the audio format is contained within the tracks themselves. The DDS format permits the use of any combination of the following:

- read-after-write
- third level of Reed-Solomon error control coding
- multiple-group-writing

Read-after-write allows the reading of the data immediately after writing - if there are any errors uncorrected by the error control, it is written repeatedly, up to 128 times, until it is free of errors. The third level of error control protects the tracks and can correct any two data tracks within a group of forty four. Multiple-group-writing rewrites a group a fixed number of times before the next group is written. If errors are found in reading a group then a repetition of the group can be read. In a hostile environment it would be desirable to use all three of the above measures but each measure reduces the storage capacity of the tape. Existing data compression strategies could, perhaps, allow these measures to be used so that the effective storage capacity of the tape is not too dissimilar to that for uncompressed data in a non-hostile environment. It is likely, however, that in a hostile environment the more mechanical procedures of repeated writings to the tape would be affected by errors in the same manner as the original so that only the third level of error control coding is truly effective. Moreover, in hostile environments data gathering is intrinsically difficult which makes repeated data gathering unattractive. Hence maximum data gathering capacity is required. Therefore, the use of the third level of error control coding alone might be the preferred option. In this thesis it will be assumed that the third level of Reed-Solomon error control coding is the only option invoked.

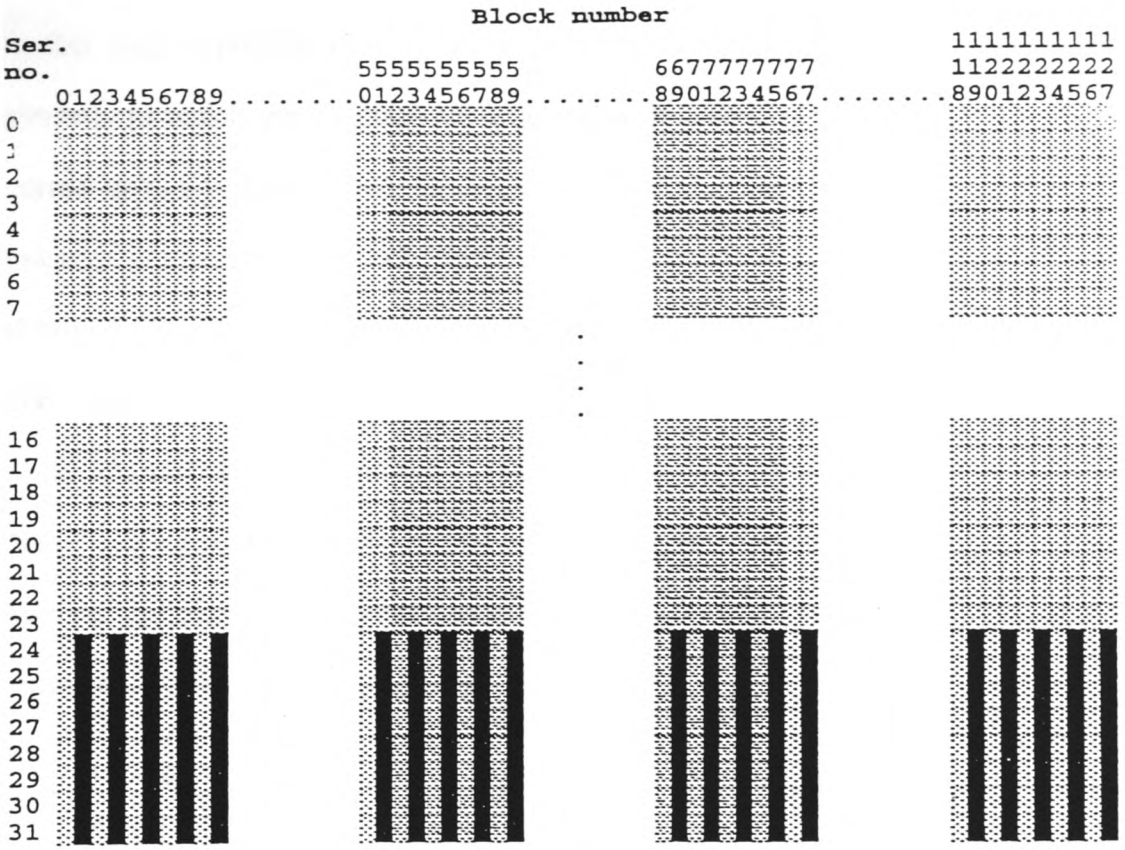
DDS structures the data into sequentially numbered *Basic Groups* of 126632 bytes and this is the smallest unit that can be coded as a whole if the format is used to maximum effect. The

encoding process may be summarized as follows, where C1, C2 and C3 label the three Reed-Solomon RS(n,k,d) codes - d is the minimum distance of the codes:



This multi-ordered Reed Solomon coding requires that the parity bytes generated in the C3 coding are protected by the C2 coding. Similarly, the parity bytes generated by the C2 coding are grouped with data bytes and protected by C1 coding. Prior to any coding the Basic Group is broken into 22 *Sub-Groups* each of 5756 bytes. The C3 parity bytes are calculated across the Basic Group - the C3 codeword consists of two data bytes from each Sub-Group plus two parity bytes. The data in every Sub-Group is written in a single *Frame* (2 contiguous Tracks) onto the tape. The burst error correction of the Reed-Solomon codes is reinforced by interleaving. Consequently, a long burst error corrupts a few bytes in many codewords rather than many bytes in just a few codewords.

In the DDS format the C1 and C2 parity bytes and the data which they protect are arranged as shown below [**1**] - the interleave depth of C1 is two bytes and the interleave depth of C2 is four blocks:



where

≡ data bytes

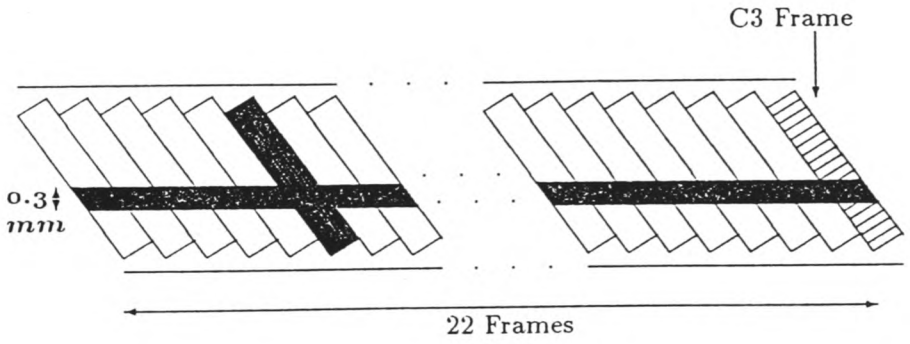
≡ C2 parity bytes

≡ C1 parity bytes

The data bytes contain 34 DDS 'housekeeping' bytes.

Each serially numbered row contains four C2 codewords. The first has a byte in each of blocks 0, 4, 8, ... etc., the second has a byte in each of blocks 1, 5, 9, ... etc. and so on. Every even numbered block, together with the block following, holds two complete C1 codewords. The bytes of each C1 codeword are placed in alternate serially numbered rows, firstly filling down through the even numbered block and then filling down through the odd numbered block. The innermost C1 code can locate and correct two symbol errors. If there are more than two errors then all symbols within the word are flagged as potential errors and passed onto the C2 code. The C2 code, with its six parity bytes per codeword, is capable of locating and correcting three symbol errors. In practice, however, only at most two symbols are corrected

in this way to prevent miscorrections [2]. If there are more than two symbol errors the symbols to which the C1 flags are attached are assumed to be in error. The C2 code can correct up to six errors if the locations are known (i.e. erasures) and if there are more than six erasures all symbols in the codeword are flagged and passed onto the C3 code. The interleave depth of C2 at four blocks means that an error covering up to twenty four blocks could be corrected without resort to C3. Such an error is equivalent to a longitudinal corruption of width around 0.3mm. The C1 and C2 parity bytes are calculated for the data within individual Sub-Groups and are stored within the same Frame. The C3 parity bytes are coded in the same way as the data and are stored in a Frame at the end of the Group. The C3 code can, therefore, correct any two tracks in a Basic Group.



The DDS-2 format [3] employs the same Reed-Solomon error control codes and interleaving as the DDS-1 format. The principal differences between the formats are:

- 1.5 times more tracks per unit length of tape than DDS-1;
- longer tape (120m);
- read-after-write repeats up to 256 times;
- marginally different track angle.

The size of the Basic Group in the DDS-3 format [4] is 384296 bytes which is broken down into twenty two Sub-Groups as in DDS-2 and DDS-1. Of the three Reed-Solomon error control codes employed, C2 and C3 are nominally the same as those in the earlier formats. In the new C2 code the parity bytes are at the ends of the codeword rather than in the centre. The new C1 code is RS(62,56,7) i.e. a three error locating and correcting code. The interleave depth of C1 is two bytes and the interleave depth of C2 is three *Fragments* (Blocks). Amongst further differences between this and the DDS-2 format are:

- doubling of the storage capacity of each track;
- less 'housekeeping' data on each track;
- longer tape (125m)

These changes constitute a threefold increase in storage capacity per unit length of tape over that of DDS-2 - the new 125m tape will store 12GB of uncompressed user data.
Compliance with this new standard will not require that a drive be able to read or write the earlier DDS formats. In practice, initially at least, compatibility with the earlier formats is provided by manufacturers [5].
This thesis concentrates on DDS-1.

1.5 Data Compression

The option exists to compress data before recording on a DDS tape. The standard which applies when compression is used with DDS is known as DDS-DC [6]. The standard specifies the 'housekeeping' aspects of handling compressed data. It does not specify any one

compression algorithm for use with this format. At around the time of the Standard's introduction there were two algorithms competing for pre-eminence - Hewlett Packard's DCLZ algorithm and the sliding window algorithm LZS of Stac Electronics [7]. Since then the DCLZ algorithm seems to have become a *de facto* industry standard algorithm for hardware compression of DDS tapes [8,9]. Indeed, the DDS-2 Standard refers to DCLZ simply as an *example* of a data compression algorithm for use with this format.

1.6 The Research Programme

At the outset it was decided that a simulation of the error control procedures of DDS was required so that the environment-dependent error patterns could be calculated. This simulation would allow the local behaviour of the compression strategies in the presence of errors to be determined. A synchronized piecewise approach was chosen as the basis for the compression strategy so that errors will propagate only as far as the next synchronizing sequence. Piecewise compression usually results in the weakening of the compression performance. The combination of this effect with the addition of the redundancy added to ensure synchronization has to be weighed against the improved data recovery. The Rice algorithm [10] was investigated firstly as it operates in a piecewise fashion and is adaptive. Adaptive algorithms are preferred since any overhead such as a probability table is susceptible to errors and, if affected, errors would propagate on decompression even if the code itself were unaffected. This fact, at once, ruled out the use of self-synchronizing Huffman [11] codes since such codes are necessarily static in nature. Notwithstanding this fact, self synchronization can never be guaranteed nor can the absence of 'slippage' in the decompressed data. Slippage refers to the loss of positioning in the decompressed data relative to the original arising as a

consequence of there having been either more or fewer values decompressed than originally compressed. The prevention of slippage in the decompressed data was a core requirement in the design specification of the compression strategies. The Rice algorithm compresses correlated physical data. The project acquired a DDS tape containing a mixture of correlated physical data and text data which was captured remotely in a potentially hostile environment. It was, therefore, considered to be particularly important that this data be compressed effectively and this is achieved with a novel eclectic compressor. The piecewise compression of English text was investigated even though it is unlikely such data would be recorded in a hostile environment. The attainment of satisfactory performance in this case provides a 'proof of principle' for the compression of more compressible text data such as graphics.

1.7 The Structure of the Presentation

Chapter Two introduces Reed-Solomon codes and describes the simulation of the error control encoding and decoding of DDS. The six erasure correction capability of C2 is detailed. The performance of DDS in the presence of burst errors and high random error rates is discussed. The third chapter begins with a discussion of the types of data and redundancy and then proceeds to describe the better known compression algorithms. The chapter concludes with a comprehensive account of the highly relevant but little known Rice algorithm. The principal results are presented in Chapter Four. The phenomenon of error propagation in compressed data is introduced firstly, heralding an examination of the means of prevention. Fully synchronized compression strategies for correlated physical, mixed correlated physical and text data and English text data are described in turn. Results for data recovery in varying hostile environments are presented.

Chapter Two

The Simulation of DDS

2.1 Purpose of the Simulation

The purpose of this work is to provide a practical solution to the very real problem of poor data recovery from compressed DDS tapes operating outside normal environmental parameters. This work can only claim to do so if it has been tested successfully in the presence of realistic error patterns. In the absence of the necessary hardware to recreate such patterns the next best step is to simulate the behaviour of the hardware computationally. The errors may, indeed, be random bit errors, as in the theoretical work of some researchers, but it is more likely in practice that the errors will be complex burst errors. The simulation of the error control coding of DDS with the input of appropriate raw errors provides the error patterns to be encountered by the data compression techniques.

2.2 Simulation

The simulation has formed a major part of the work completed. The standard specifies the encoding of DDS. It is interesting to note that the standard does not specify the decoding procedures. The task of simulating the encoding of DDS computationally, as specified by the standard, was made easier than it might otherwise have been as a fellow researcher in the

division had already simulated CDROM [**12**]. He followed the CDROM standard and located the literature on decoding RS codes with four parity bytes. The error control coding of CDROM and DDS are similar in many respects - both use Reed-Solomon error control coding and interleaving strategies and so some of the CDROM software was adapted, with little difficulty, for use in the DDS simulation.

2.3 Reed-Solomon Coding

2.3.1 Preamble

The Reed-Solomon codes used in DDS (and CDROM) are GF(2^8) codes, that is, the symbols used are of eight bits [**1**]. The polynomial which defines calculation in the Galois fields for all the codes is $p = x^8 + x^4 + x^3 + x^2 + 1$ and the primitive element α is (00000010). In the decoding procedures syndromes S_i are calculated and any non-zero syndromes indicate the presence of errors. The positions of the errors can be determined as each symbol in a codeword is multiplied by a different power of α in the encoding process. As an illustration of this consider the following example of a Reed-Solomon decoding procedure:-

Suppose the Reed-Solomon codeword is composed of five data bytes A-E followed by parity bytes P and Q. Then

$S_0 = A + B + C + D + E + P + Q$

and

$S_1 = \alpha^6 A + \alpha^5 B + \alpha^4 C + \alpha^3 D + \alpha^2 E + \alpha^1 P + \alpha^0 Q.$

If there is an error in symbol B then

$S_1/S_0 = \alpha^5/\alpha^0 = \alpha^5$ (see **Equation 2.4**)

giving the location of the symbol in error. The correct symbol value is determined by adding S_0 to the symbol in error.

2.3.2 The Error Location Polynomial

Suppose there are n symbols V_i from the Galois field (2^u) in a Reed-Solomon codeword of which r are parity check symbols. Then

$$\sum_{i=0}^{n-1} \alpha^{ji} V_i = 0 \qquad (2.1)$$

where j runs from 0 to (r-1).

The syndromes S_j are

$$S_j = \sum_{i=0}^{n-1} \alpha^{ji} V_i^* \qquad (2.2)$$

where the V_i^* are the symbols received. The error E_i is the difference between the received symbol and the original symbol.

$$E_i = (V_i^* - V_i) \qquad (2.3)$$

Substituting for the received symbol in the equation for the syndromes gives

$$S_j = \sum_{i=0}^{n-1} \alpha^{ji}(E_i + V_i) = \sum_{i=0}^{n-1} \alpha^{ji} E_i \qquad (2.4)$$

The *error location polynomial* may be constructed as follows, where ν is the actual number of errors and {I} is the set of error locations:

$$\sigma = \prod_{i \in \{I\}} (1 - \alpha^{-i} x) = \sum_{m=0}^{\nu} \sigma_m x^m = \sum_{m=0}^{t} \sigma_m x^m \qquad (2.5)$$

If $\nu \leq t$, where t is the number of symbol errors the code can correct (t = (d - 1)/2), then σ_m will be zero for m>ν.

If α^i is substituted for x in the above equation, then

$$\sum_{m=0}^{t} \sigma_m \alpha^{mi} = 0 \qquad (2.6)$$

for $i \in \{I\}$.

It may be shown [**13**] that

$$\sum_{m=0}^{t} \sigma_m S_{m+k} = 0 \qquad (2.7)$$

for k = 0, 1, ... , (t - 1). This equation may be written equivalently as

$$\begin{bmatrix} S_0 & S_1 & \cdot & \cdot & \cdot & S_t \\ S_1 & S_2 & \cdot & \cdot & \cdot & S_{t+1} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ S_{t-1} & S_t & \cdot & \cdot & \cdot & S_{2t-1} \end{bmatrix} \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \cdot \\ \cdot \\ \cdot \\ \sigma_{t-1} \\ \sigma_t \end{bmatrix} = 0 \qquad \textbf{(2.8)}$$

If the matrix M_t formed by eliminating the last column of the t × (t+1) matrix, M, above is non-singular then, using Cramer's rule

$$\frac{\sigma_m}{\sigma_t} = \frac{\Delta_{tm}}{\Delta_{tt}} \qquad \textbf{(2.9)}$$

for m = 0 to (t-1) and where Δ_{tt} is the determinant of M_t. Δ_{tm} is the determinant of the matrix formed by replacing the m^{th} column in the matrix M_t with the last column of the matrix M for m=0, 1, ... , (t - 1) where the sign of each element is reversed. If there are less than t errors then the above equation may be replaced by [13]

$$\frac{\sigma_m}{\sigma_\nu} = \frac{\Delta_m}{\Delta_\nu} \qquad \textbf{(2.10)}$$

where

$\Delta_m = \Delta_{mm} = 0$ for $m > \nu$

$\Delta_m = \Delta_{\nu m}$ for $m \leq \nu$.

$\sigma_0 = 1$ so all σ_m may be found.

The modified error location polynomial, for $i \in \{I\}$, becomes

$$\sum_{m=0}^{t} \Delta_m \alpha^{mi} = 0 \qquad (2.11)$$

2.3.3 The Parity Byte Equations for the DDS Reed-Solomon Codes

The implementation of the Reed-Solomon encoding in the DDS simulation requires the calculation of the parity byte equations. In the usual way the parity bytes satisfy the equation

$$H * V = 0 \qquad (2.12)$$

where H is the parity check matrix and V is the vector whose components are the bytes of the Reed-Solomon codeword and are labelled V_1, V_2 etc. and * means matrix multiplication. The parity byte equations are calculated by solving the above equation manually.

The parity check matrix for the C3 code is [1]

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 1 & 1 \\ \alpha^{45} & \alpha^{44} & \alpha^{43} & \dots & \alpha^{2} & \alpha & 1 \end{bmatrix} \qquad (2.13)$$

where the number of rows equals the number of parity bytes and V^T is

$$[V1 \;\; V2 \;\; V3 \;\; \dots \;\; V44 \;\; P1 \;\; P2] \qquad (2.14)$$

The C3 parity bytes were found to be

$$P_1=\alpha^{6}V_1+\alpha^{190}V_2+\alpha^{96}V_3+\alpha^{250}V_4+\alpha^{132}V_5+\alpha^{59}V_6+\alpha^{81}V_7+\alpha^{159}V_8+\alpha^{154}V_9+\alpha^{200}V_{10}$$
$$+\alpha^{7}V_{11}+\alpha^{111}V_{12}+\alpha^{245}V_{13}+\alpha^{10}V_{14}+\alpha^{20}V_{15}+\alpha^{41}V_{16}+\alpha^{156}V_{17}+\alpha^{168}V_{18}+\alpha^{79}V_{19}+\alpha^{173}V_{20}$$
$$+\alpha^{231}V_{21}+\alpha^{229}V_{22}+\alpha^{171}V_{23}+\alpha^{210}V_{24}+\alpha^{240}V_{25}+\alpha^{17}V_{26}+\alpha^{67}V_{27}+\alpha^{215}V_{28}+\alpha^{43}V_{29}$$
$$+\alpha^{120}V_{30}+\alpha^{8}V_{31}+\alpha^{199}V_{32}+\alpha^{74}V_{33}+\alpha^{102}V_{34}+\alpha^{220}V_{35}+\alpha^{251}V_{36}+\alpha^{95}V_{37}+\alpha^{175}V_{38}+\alpha^{87}V_{39}$$
$$+\alpha^{166}V_{40}+\alpha^{113}V_{41}+\alpha^{75}V_{42}+\alpha^{198}V_{43}+\alpha^{25}V_{44} \qquad \textbf{(2.15)}$$

$$P_2=\alpha^{191}V_1+\alpha^{97}V_2+\alpha^{251}V_3+\alpha^{133}V_4+\alpha^{60}V_5+\alpha^{82}V_6+\alpha^{160}V_7+\alpha^{155}V_8+\alpha^{201}V_9+\alpha^{8}V_{10}$$
$$+\alpha^{112}V_{11}+\alpha^{246}V_{12}+\alpha^{11}V_{13}+\alpha^{21}V_{14}+\alpha^{42}V_{15}+\alpha^{157}V_{16}+\alpha^{169}V_{17}+\alpha^{80}V_{18}+\alpha^{174}V_{19}$$
$$+\alpha^{232}V_{20}+\alpha^{230}V_{21}+\alpha^{172}V_{22}+\alpha^{211}V_{23}+\alpha^{241}V_{24}+\alpha^{18}V_{25}+\alpha^{68}V_{26}+\alpha^{216}V_{27}+\alpha^{44}V_{28}$$
$$+\alpha^{121}V_{29}+\alpha^{9}V_{30}+\alpha^{200}V_{31}+\alpha^{75}V_{32}+\alpha^{103}V_{33}+\alpha^{221}V_{34}+\alpha^{252}V_{35}+\alpha^{96}V_{36}+\alpha^{176}V_{37}$$
$$+\alpha^{88}V_{38}+\alpha^{167}V_{39}+\alpha^{114}V_{40}+\alpha^{76}V_{41}+\alpha^{199}V_{42}+\alpha^{26}V_{43}+\alpha^{1}V_{44} \qquad \textbf{(2.16)}$$

The C2 parity check matrix is [1]

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots\cdots & 1 & 1 & 1 \\ \alpha^{31} & \alpha^{30} & \alpha^{29} & \alpha^{28} & \cdots\cdots & \alpha^{2} & \alpha & 1 \\ \alpha^{62} & \alpha^{60} & \alpha^{58} & \alpha^{56} & \cdots\cdots & \alpha^{4} & \alpha^{2} & 1 \\ \alpha^{93} & \alpha^{90} & \alpha^{87} & \alpha^{84} & \cdots\cdots & \alpha^{6} & \alpha^{3} & 1 \\ \alpha^{124} & \alpha^{120} & \alpha^{116} & \alpha^{112} & \cdots\cdots & \alpha^{8} & \alpha^{4} & 1 \\ \alpha^{155} & \alpha^{150} & \alpha^{145} & \alpha^{140} & \cdots\cdots & \alpha^{10} & \alpha^{5} & 1 \end{bmatrix} \qquad \textbf{(2.17)}$$

and V^T is

$$[V1\ V2\ V3\ \ldots\ V13\ Q1\ Q2\ \ldots\ Q6\ V14\ \ldots\ V26] \qquad \textbf{(2.18)}$$

The six C2 parity bytes are

$$Q_1=\alpha^{174}V_1+\alpha^{33}V_2+\alpha^{92}V_3+\alpha^{192}V_4+\alpha^{180}V_5+\alpha^{76}V_6+\alpha^{177}V_7+\alpha^{162}V_8+\alpha^{108}V_9+\alpha^{225}V_{10}+$$
$$\alpha^{205}V_{11}+\alpha^{228}V_{12}+\alpha^{166}V_{13}+\alpha^{240}V_{20}+\alpha^{146}V_{21}+\alpha^{203}V_{22}+\alpha^{175}V_{23}+\alpha^{190}V_{24}+\alpha^{68}V_{25}+$$
$$\alpha^{117}V_{26}+\alpha^{127}V_{27}+\alpha^{21}V_{28}+\alpha^{120}V_{29}+\alpha^{127}V_{30}+\alpha^{22}V_{31}+\alpha^{213}V_{32} \qquad \textbf{(2.19)}$$

$$Q_2=\alpha^{163}V_1+\alpha^{175}V_2+\alpha^{69}V_3+\alpha^{82}V_4+\alpha^{138}V_5+\alpha^{15}V_6+\alpha^{203}V_7+\alpha^{100}V_8+\alpha^{169}V_9+\alpha^{46}V_{10}+$$
$$\alpha^{187}V_{11}+\alpha^{169}V_{12}+\alpha^{0}V_{13}+\alpha^{151}V_{20}+\alpha^{180}V_{21}+\alpha^{149}V_{22}+\alpha^{208}V_{23}+\alpha^{204}V_{24}+\alpha^{150}V_{25}+$$
$$\alpha^{112}V_{26}+\alpha^{212}V_{27}+\alpha^{4}V_{28}+\alpha^{42}V_{29}+\alpha^{97}V_{30}+\alpha^{58}V_{31}+\alpha^{243}V_{32} \qquad \textbf{(2.20)}$$

$$Q_3=\alpha^{151}V_1+\alpha^{102}V_2+\alpha^{149}V_3+\alpha^{252}V_4+\alpha^{221}V_5+\alpha^{166}V_6+\alpha^{80}V_7+\alpha^{64}V_8+\alpha^{45}V_9+\alpha^{45}V_{10}+$$
$$\alpha^{201}V_{11}+\alpha^{89}V_{12}+\alpha^{134}V_{13}+\alpha^{240}V_{20}+\alpha^{29}V_{21}+\alpha^{121}V_{22}+\alpha^{92}V_{23}+\alpha^{175}V_{24}+\alpha^{102}V_{25}+$$
$$\alpha^{132}V_{26}+\alpha^{145}V_{27}+\alpha^{27}V_{28}+\alpha^{218}V_{29}+\alpha^{212}V_{30}+\alpha^{221}V_{31}+\alpha^{217}V_{32} \qquad \textbf{(2.21)}$$

$$Q_4=\alpha^{42}V_1+\alpha^{41}V_2+\alpha^{27}V_3+\alpha^{28}V_4+\alpha^{87}V_5+\alpha^{200}V_6+\alpha^{182}V_7+\alpha^{147}V_8+\alpha^{215}V_9+\alpha^{127}V_{10}+$$

$$\alpha^{151}V_{11}+\alpha^{54}V_{12}+\alpha^{5}V_{13}+\alpha^{119}V_{20}+\alpha^{69}V_{21}+\alpha^{176}V_{22}+\alpha^{15}V_{23}+\alpha^{10}V_{24}+\alpha^{24}V_{25}+\alpha^{35}V_{26}+$$

$$\alpha^{116}V_{27}+\alpha^{166}V_{28}+\alpha^{192}V_{29}+\alpha^{84}V_{30}+\alpha^{32}V_{31}+\alpha^{76}V_{32} \qquad \textbf{(2.22)}$$

$$Q_5=\alpha^{73}V_1+\alpha^{138}V_2+\alpha^{172}V_3+\alpha^{112}V_4+\alpha^{69}V_5+\alpha^{17}V_6+\alpha^{167}V_7+\alpha^{200}V_8+\alpha^{249}V_9+\alpha^{248}V_{10}+$$

$$\alpha^{184}V_{11}+\alpha^{210}V_{12}+\alpha^{176}V_{13}+\alpha^{245}V_{20}+\alpha^{154}V_{21}+\alpha^{167}V_{22}+\alpha^{21}V_{23}+\alpha^{139}V_{24}+\alpha^{65}V_{25}+$$

$$\alpha^{163}V_{26}+\alpha^{225}V_{27}+\alpha^{88}V_{28}+\alpha^{27}V_{29}+\alpha^{9}V_{30}+\alpha^{110}V_{31}+\alpha^{93}V_{32} \qquad \textbf{(2.23)}$$

$$Q_6=\alpha^{48}V_1+\alpha^{107}V_2+\alpha^{207}V_3+\alpha^{195}V_4+\alpha^{91}V_5+\alpha^{192}V_6+\alpha^{177}V_7+\alpha^{123}V_8+\alpha^{240}V_9+\alpha^{220}V_{10}+$$

$$\alpha^{243}V_{11}+\alpha^{181}V_{12}+\alpha^{15}V_{13}+\alpha^{161}V_{20}+\alpha^{218}V_{21}+\alpha^{190}V_{22}+\alpha^{205}V_{23}+\alpha^{83}V_{24}+\alpha^{132}V_{25}+$$

$$\alpha^{142}V_{26}+\alpha^{36}V_{27}+\alpha^{135}V_{28}+\alpha^{142}V_{29}+\alpha^{37}V_{30}+\alpha^{228}V_{31}+\alpha^{109}V_{32} \qquad \textbf{(2.24)}$$

The C1 parity check matrix is [1]

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots\cdots & 1 & 1 & 1 \\ \alpha^{31} & \alpha^{30} & \alpha^{29} & \alpha^{28} & \cdots\cdots & \alpha^{2} & \alpha & 1 \\ \alpha^{62} & \alpha^{60} & \alpha^{58} & \alpha^{56} & \cdots\cdots & \alpha^{4} & \alpha^{2} & 1 \\ \alpha^{93} & \alpha^{90} & \alpha^{87} & \alpha^{84} & \cdots\cdots & \alpha^{6} & \alpha^{3} & 1 \end{bmatrix} \qquad \textbf{(2.25)}$$

and V^T is

$$[V1\ V2\ V3\ \ldots\ V28\ P1\ P2\ P3\ P4] \tag{2.26}$$

The four C1 parity bytes are

$$P_1=\alpha^{249}V_1+\alpha^{142}V_2+\alpha^{180}V_3+\alpha^{197}V_4+\alpha^{5}V_5+\alpha^{155}V_6+\alpha^{153}V_7+\alpha^{132}V_8+\alpha^{143}V_9+\alpha^{244}V_{10}+$$
$$\alpha^{101}V_{11}+\alpha^{76}V_{12}+\alpha^{102}V_{13}+\alpha^{155}V_{14}+\alpha^{203}V_{15}+\alpha^{104}V_{16}+\alpha^{58}V_{17}+\alpha^{152}V_{18}+\alpha^{173}V_{19}+$$
$$\alpha^{95}V_{20}+\alpha^{88}V_{21}+\alpha^{43}V_{22}+\alpha^{134}V_{23}+\alpha^{205}V_{24}+\alpha^{143}V_{25}+\alpha^{131}V_{26}+\alpha^{163}V_{27}+$$
$$\alpha^{75}V_{28} \tag{2.27}$$

$$P_2=\alpha^{205}V_1+\alpha^{252}V_2+\alpha^{218}V_3+\alpha^{199}V_4+\alpha^{202}V_5+\alpha^{41}V_6+\alpha^{136}V_7+\alpha^{106}V_8+\alpha^{119}V_9+$$
$$\alpha^{238}V_{10}+\alpha^{193}V_{11}+\alpha^{103}V_{12}+\alpha^{123}V_{13}+\alpha^{242}V_{14}+\alpha^{83}V_{15}+\alpha^{178}V_{16}+\alpha^{30}V_{17}+\alpha^{214}V_{18}+$$
$$\alpha^{148}V_{19}+\alpha^{138}V_{20}+\alpha^{112}V_{21}+\alpha^{154}V_{22}+\alpha^{157}V_{23}+\alpha^{96}V_{24}+\alpha^{49}V_{25}+\alpha^{198}V_{26}+\alpha^{189}V_{27}+$$
$$\alpha^{249}V_{28} \tag{2.28}$$

$$P_3=\alpha^{67}V_1+\alpha^{11}V_2+\alpha^{131}V_3+\alpha^{40}V_4+\alpha^{7}V_5+\alpha^{41}V_6+\alpha^{80}V_7+\alpha^{147}V_8+\alpha^{151}V_9+\alpha^{17}V_{10}+$$

$$\alpha^{245}V_{11}+\alpha^{253}V_{12}+\alpha^{208}V_{13}+\alpha^{66}V_{14}+\alpha^{228}V_{15}+\alpha^{116}V_{16}+\alpha^{162}V_{17}+\alpha^{244}V_{18}+\alpha^{13}V_{19}+$$

$$\alpha^{171}V_{20}+\alpha^{213}V_{21}+\alpha^{236}V_{22}+\alpha^{71}V_{23}+\alpha^{177}V_{24}+\alpha^{253}V_{25}+\alpha^{162}V_{26}+\alpha^{59}V_{27}+$$

$$\alpha^{78}V_{28} \qquad (2.29)$$

$$P_4=\alpha^{148}V_1+\alpha^{186}V_2+\alpha^{203}V_3+\alpha^{11}V_4+\alpha^{161}V_5+\alpha^{159}V_6+\alpha^{138}V_7+\alpha^{149}V_8+\alpha^{250}V_9+\alpha^{107}V_{10}+$$

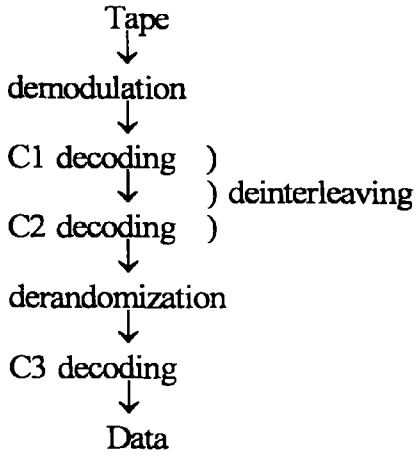
$$\alpha^{82}V_{11}+\alpha^{108}V_{12}+\alpha^{161}V_{13}+\alpha^{209}V_{14}+\alpha^{110}V_{15}+\alpha^{64}V_{16}+\alpha^{158}V_{17}+\alpha^{179}V_{18}+\alpha^{101}V_{19}+$$

$$\alpha^{94}V_{20}+\alpha^{49}V_{21}+\alpha^{140}V_{22}+\alpha^{211}V_{23}+\alpha^{149}V_{24}+\alpha^{137}V_{25}+\alpha^{169}V_{26}+\alpha^{81}V_{27}+$$

$$\alpha^{6}V_{28} \qquad (2.30)$$

2.4 C2 Decoding

The DDS decoding may be summarized as follows



The C2 Reed-Solomon decoding of DDS with six parity bytes per codeword is the most complex single decoding procedure of both DDS and CDROM. The solution techniques of C2 decoding encompass all those of C1 and C3 decoding and more besides. Therefore, only C2 decoding will be described.

The innermost C1 code flags symbols deemed to be at risk i.e. the symbols contained in a C1 codeword with more than two symbols in error. The C2 code addresses these flagged symbols i.e. erasures; the flags can be used to correct up to six erasures. The C2 decoding procedures are [**14**]:

1 $N(e) \leq 2$	N(e) error correction
2 $N(e) > 2$, $N(F) = 1$ or 2	N(F) erasure correction and 2 error correction
3 $N(e) > 2$, $N(F) = 3$ or 4	N(F) erasure correction and 1 error correction
4 $N(e) > 2$, $N(F) = 5$ or 6	N(F) erasure correction

where N(e) is the number of detected errors and N(F) is the number of flags; an *erasure* is an error for which the location is known but the magnitude is not. Three error location and correction is not used for fear of miscorrection [**2**] - there would be no spare syndromes to be used as a consistency check.

The C2 code is a three error locating and correcting code (i.e. t = 3). The equation for the coefficients of the error location polynomial is:

$$\begin{bmatrix} S_0 & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{bmatrix} \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \end{bmatrix} = \sigma_3 \begin{bmatrix} S_3 \\ S_4 \\ S_5 \end{bmatrix} \qquad (2.31)$$

The determinants Δ_{33}, Δ_{32}, Δ_{31} and Δ_{30} are

$$\Delta_{33} = S_2(S_1S_3 + S_2S_2) + S_3(S_0S_3 + S_1S_2) + S_4(S_1S_1 + S_2S_0) \qquad (2.32)$$

$$\Delta_{32} = S_3(S_1S_3 + S_2S_2) + S_4(S_0S_3 + S_1S_2) + S_5(S_1S_1 + S_2S_0) \qquad (2.33)$$

$$\Delta_{31} = S_0(S_4S_4 + S_3S_5) + S_1(S_3S_4 + S_2S_5) + S_2(S_3S_3 + S_2S_4) \qquad (2.34)$$

$$\Delta_{30} = S_1(S_4S_4 + S_3S_5) + S_2(S_3S_4 + S_2S_5) + S_3(S_3S_3 + S_2S_4) \qquad (2.35)$$

The error location polynomial is

$$\Delta_{33}\alpha^{3i} + \Delta_{32}\alpha^{2i} + \Delta_{31}\alpha^{i} + \Delta_{30} = 0 \qquad (2.36)$$

If there are two errors then the error location polynomial is

$$\Delta_{22}\alpha^{2i} + \Delta_{21}\alpha^{i} + \Delta_{20} = 0 \qquad (2.37)$$

where

$$\Delta_{22}=S_1S_1+S_2S_0 \quad (2.38)$$

$$\Delta_{21}=S_0S_3+S_1S_2 \quad (2.39)$$

$$\Delta_{20}=S_1S_3+S_2S_2 \quad (2.40)$$

In the case of just a single error then the error location polynomial is

$$\Delta_{11}\alpha^i+\Delta_{10}=0 \quad (2.41)$$

where

$$\Delta_{11}=S_0 \quad (2.42)$$

$$\Delta_{10}=S_1 \quad (2.43)$$

2.4.1 Single Byte Error Correction

This is performed as in **Section 2.3.1**. There is a single byte error if and only if

$$location=\frac{S_1}{S_0}=\frac{S_2}{S_1}=\frac{S_3}{S_2}=\frac{S_4}{S_3}=\frac{S_5}{S_4} \quad (2.44)$$

i.e. the syndromes all agree on the location of the error. If the above relationships hold then it can be seen that the coefficient Δ_{33} in the three error location polynomial and the coefficient

Δ_{22} in the two error location polynomial are both zero. The correct symbol is obtained by adding S_0 to the symbol in error.

2.4.2 Correction of Two Byte Errors

In DDS the error location polynomial reduces to a quadratic if

$$S_2(S_1S_3+S_2S_2)+S_3(S_0S_3+S_1S_2)+S_4(S_1S_1+S_2S_0)=0 \qquad (2.45)$$

The error location polynomial becomes

$$(S_1^2+S_0S_2)x^2+(S_1S_2+S_0S_3)x+(S_2^2+S_1S_3)=0 \qquad (2.46)$$

and is solved following the procedure within the paper of Ko and Tjhung [15].

Substitute

$$x=\beta y \qquad (2.47)$$

where

$$\beta=\frac{S_1S_2+S_0S_3}{S_1^2+S_0S_2} \qquad (2.48)$$

The error location polynomial becomes

$$y^2+y+\gamma=0 \tag{2.49}$$

and

$$\gamma=(S_1S_3+S_2^2)\left(\frac{S_1S_2+S_0S_3}{S_1^2+S_0S_2}\right)^2 \tag{2.50}$$

Only one constant in the error location polynomial now depends upon the codeword. The error location polynomial is solved using a table look-up approach. Suppose that $f_a(\gamma)$ and $f_b(\gamma)$ are the roots of the error location polynomial. Then

$$(y+f_a(\gamma))\,(y+f_b(\gamma))=y^2+y+\gamma \tag{2.51}$$

The locations of the errors are

$$\alpha^{-a}=\beta f_a(\gamma) \tag{2.52}$$

$$\alpha^{-b}=\beta f_b(\gamma) \tag{2.53}$$

The error magnitudes may then be found from **Equation 2.4**.

2.4.3 Erasure Correction

If there are five or six flagged bytes the flags may be used to correct five or six bytes. The **Equation 2.4** becomes

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ \alpha^a & \alpha^b & \alpha^c & \alpha^d & \alpha^e & \alpha^f \\ \alpha^{2a} & \alpha^{2b} & \alpha^{2c} & \alpha^{2d} & \alpha^{2e} & \alpha^{2f} \\ \alpha^{3a} & \alpha^{3b} & \alpha^{3c} & \alpha^{3d} & \alpha^{3e} & \alpha^{3f} \\ \alpha^{4a} & \alpha^{4b} & \alpha^{4c} & \alpha^{4d} & \alpha^{4e} & \alpha^{4f} \\ \alpha^{5a} & \alpha^{5b} & \alpha^{5c} & \alpha^{5d} & \alpha^{5e} & \alpha^{5f} \end{bmatrix} \begin{bmatrix} Ea \\ Eb \\ Ec \\ Ed \\ Ee \\ Ef \end{bmatrix} \quad \textbf{(2.54)}$$

The errors may be found through the process of Gaussian elimination. A simpler set of purely hypothetical equations may be used to illustrate this process in GF(2^8). Suppose there are three equations

$$E_a+E_b+E_c=S_0$$

$$\alpha^a E_a+\alpha^b E_b+\alpha^c E_c=S_1$$

$$\alpha^{2a} E_a+\alpha^{2b} E_b+\alpha^{2c} E_c=S_2$$

The modulo 2 arithmetic means that there is no distinction between addition and subtraction. The elimination of the coefficients of E_a in the second and third equations yields

$$(\alpha^a+\alpha^b)\, E_b+(\alpha^a+\alpha^c)\, E_c=\alpha^a S_0+S_1$$

$$(\alpha^{2a}+\alpha^{2b})\, E_b+(\alpha^{2a}+\alpha^{2c})\, E_c=\alpha^{2a} S_0+S_2$$

Eliminating the coefficient of E_b in this last equation gives

$$(\alpha^a+\alpha^c)\,(\alpha^b+\alpha^c)\, E_c=\alpha^{a+b} S_0+(\alpha^a+\alpha^b)\, S_1+S_2$$

The set of three equations becomes

$$E_a+E_b+E_c=S_0$$

$$(\alpha^a+\alpha^b)\, E_b+(\alpha^a+\alpha^c)\, E_c=\alpha^a S_0+S_1$$

$$(\alpha^a+\alpha^c)\ (\alpha^b+\alpha^c)\ E_c=\alpha^{a+b}S_0+(\alpha^a+\alpha^b)\ S_1+S_2$$

These equations may be written in matrix form as

$$\begin{bmatrix}1 & 1 & 1\\ 0 & (\alpha^a+\alpha^b) & (\alpha^a+\alpha^c)\\ 0 & 0 & (\alpha^a+\alpha^c)\ (\alpha^b+\alpha^c)\end{bmatrix}\begin{bmatrix}E_a\\ E_b\\ E_c\end{bmatrix}=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & \alpha^b & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ \alpha^a & 1 & 0\\ 0 & \alpha^a & 1\end{bmatrix}\begin{bmatrix}S_0\\ S_1\\ S_2\end{bmatrix}$$

The application of Gaussian elimination to **Equation 2.4** yields:

$$\begin{bmatrix}Q_0\\ Q_1\\ Q_2\\ Q_3\\ Q_4\\ Q_5\end{bmatrix}=\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1\\ 0 & \alpha^b+\alpha^a & \alpha^c+\alpha^a & \alpha^d+\alpha^a & \alpha^e+\alpha^a & \alpha^f+\alpha^a\\ 0 & 0 & (\alpha^c+\alpha^a)\ (\alpha^c+\alpha^b) & (\alpha^d+\alpha^a)\ (\alpha^d+\alpha^b) & (\alpha^e+\alpha^a)\ (\alpha^e+\alpha^b) & (\alpha^f+\alpha^a)\ (\alpha^f+\alpha^b)\\ 0 & 0 & 0 & (\alpha^d+\alpha^a)\ (\alpha^d+\alpha^b)\ (\alpha^d+\alpha^c) & (\alpha^e+\alpha^a)\ (\alpha^e+\alpha^b)\ (\alpha^e+\alpha^c) & (\alpha^f+\alpha^a)\ (\alpha^f+\alpha^b)\ (\alpha^f+\alpha^c)\\ 0 & 0 & 0 & 0 & (\alpha^e+\alpha^a)\ (\alpha^e+\alpha^b)\ (\alpha^e+\alpha^c)\ (\alpha^e+\alpha^d) & (\alpha^f+\alpha^a)\ (\alpha^f+\alpha^b)\ (\alpha^f+\alpha^c)\ (\alpha^f+\alpha^d)\\ 0 & 0 & 0 & 0 & 0 & (\alpha^f+\alpha^a)\ (\alpha^f+\alpha^b)\ (\alpha^f+\alpha^c)\ (\alpha^f+\alpha^d)\ (\alpha^f+\alpha^e)\end{bmatrix}\begin{bmatrix}Ea\\ Eb\\ Ec\\ Ed\\ Ee\\ Ef\end{bmatrix}$$

(2.55)

where

$$\begin{bmatrix}Q_0\\ Q_1\\ Q_2\\ Q_3\\ Q_4\\ Q_5\end{bmatrix}=M1*M2*M3*M4*M5*\begin{bmatrix}S_0\\ S_1\\ S_2\\ S_3\\ S_4\\ S_5\end{bmatrix}\qquad\textbf{(2.56)}$$

and

$$M1=\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&\alpha^{e}&1\end{bmatrix} \quad \textbf{(2.57)}$$

$$M2=\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&\alpha^{d}&1&0\\0&0&0&0&\alpha^{d}&1\end{bmatrix} \quad \textbf{(2.58)}$$

$$M3=\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&\alpha^{c}&1&0&0\\0&0&0&\alpha^{c}&1&0\\0&0&0&0&\alpha^{c}&1\end{bmatrix} \quad \textbf{(2.59)}$$

$$M4=\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&\alpha^{b}&1&0&0&0\\0&0&\alpha^{b}&1&0&0\\0&0&0&\alpha^{b}&1&0\\0&0&0&0&\alpha^{b}&1\end{bmatrix} \quad \textbf{(2.60)}$$

$$M5=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \alpha^a & 1 & 0 & 0 & 0 & 0 \\ 0 & \alpha^a & 1 & 0 & 0 & 0 \\ 0 & 0 & \alpha^a & 1 & 0 & 0 \\ 0 & 0 & 0 & \alpha^a & 1 & 0 \\ 0 & 0 & 0 & 0 & \alpha^a & 1 \end{bmatrix} \qquad (2.61)$$

Given the flags and the syndromes the errors can be calculated. If there are five flags then E_r is zero.

2.4.3.1 Modified Syndromes

If there are more than two detected errors and one, two, three or four flags then modified syndromes [16] have to be determined for use in the error location strategies of the procedures for correction of one and two byte errors. The locations so generated are combined with those indicated by the flags for erasure correction.

For example, if there are two flags then the error location polynomial may be expressed as:

$$\sigma = (\alpha^a + x) (\alpha^b + x) = \alpha^{a+b} + (\alpha^a + \alpha^b) x + x^2 = B_0 + B_1 x + x^2$$

Then the j^{th} modified syndrome $= B_0 S_{0+j} + B_1 S_{1+j} + S_{2+j}$, for j = 0 to 3.

In general, if there are i flags then error positions α^a, α^b ... are known and can be inserted into the relevant matrices. Then

$$S_j^* = (M1*M2*M3*M4*M5)_{row(i+1)} * \begin{bmatrix} S_{0+j} \\ S_{1+j} \\ S_{2+j} \\ S_{3+j} \\ S_{4+j} \\ S_{5+j} \end{bmatrix} \qquad (2.62)$$

where the superscript * labels the modified syndromes and j runs from 0 to 5-i. The matrix M1*M2*M3*M4*M5 is lower triangular.

2.5 Error Correction Performance

The errors which affect the data stored on a tape fall within two categories - random errors and burst errors. Errors do not, generally, fall exclusively within one or other category but are a combination of both. Random errors are exactly that; they are errors each affecting a single bit caused by, for example, random electrical fluctuations or random variations in the medium manufacturing process. Burst errors are, potentially, more devastating. In a burst error a large number of bits are affected, usually with a single cause e.g. a scratch on the surface of the medium or perturbation of the write heads. The error correction strategies employed by DDS are designed to be effective in correcting both types of error. The error correction performance has been analyzed in the literature [17,18]. A typical random bit error rate that arises in media testing is one bit error in a hundred thousand bits. Even at a hundred times this error rate the three levels of error control coding can provide an output with an error rate of one symbol in 10^{35} - very effective indeed. The two levels of error control of the audio format allow the correction of a burst up to 0.3 mm wide running parallel to the edge of the tape or of a burst up to 2.6 mm wide running perpendicular to the edge of the tape. If the widths of the bursts

of errors are any larger then not even the third level of error control coding can correct them. The third level of error control coding, by its very nature, is effective in correcting helical burst errors i.e. errors occurring along the tracks as the data is written (the tracks are written at an angle of 6.35 degrees to the edge of the tape) but, again, only two tracks may be corrected in a group of 44 data tracks.

2.5.1 Evaluation of the Performance of the Computer Simulation

The simulation of the encoding and decoding procedures of DDS was written in PASCAL and run on a Dec Alpha as it provides the memory and high-speed computation required. Random errors, lying within the expected correction capability of DDS, were introduced into the data before decoding. The simulation did correct random errors up to the limit of the expected correction capability of DDS. The largest burst error that DDS is expected to correct was introduced into the data before decoding and, again, the simulation performed as anticipated and corrected the entire burst (of up to 6336 bits per track [17]).

2.5.2 Implications for Data Compression

Random error rates beyond those explored in the literature were input into the simulation and the resultant uncorrected errors were calculated. Uncorrected errors were readily observed in

the simulations when the bit random error rate was 0.004 and greater. The results of this exercise are shown below (there were 32 simulations at each error rate):

bit random error rate	**corrected symbol error rate**
0.01	0.023
0.009	0.013
0.008	0.0059
0.007	0.0025
0.006	0.0013
0.005	0.00037
0.004	0.000015

Longitudinal corruptions with widths in excess of that which can be corrected by the DDS format were input into the simulation. The maximum width of longitudinal error correctable by DDS is 0.3 mm [**17**] or 24 blocks i.e. a block has an effective width of $^{1}/_{80}$ mm. At a corruption width of $^{25}/_{80}$ mm, for example, there were 16 error-free runs of 108 bytes and 11 error-free runs of 124 bytes in a Frame. Recalling that there are 5756 data bytes in a Frame it can be seen that error-free runs amount to 53.7% of the total data. The effects of larger corruptions are summarised in the table below:

corruption width (mm)	**numbers of significant error-free runs of output data bytes per Frame**
25/80	16 of 108, 11 of 124
26/80	27 of 108
27/80	16 of 100, 11 of 108
28/80	27 of 100
29/80	16 of 92, 11 of 100

The interleaving in DDS is designed to disperse the erroneous bytes of a burst error over many codewords to increase the magnitude of the maximum correctable burst. The interleaving has unfortunate implications for compressed data in the presence of excessive corruption even when its size only marginally exceeds that of the maximum correctable burst. As shown above, only relatively short runs of error-free bytes are produced on decoding, containing around half of the total data bytes. The significance of the above table will become apparent later when data compression strategies are linked to the DDS simulation.

Chapter Three

Data Compression

3.1 Data and Redundancy

Digital data may be broadly classified as either correlated physical data or text data. Correlated physical data is the result of the analogue-to-digital conversion of measurements of physical properties - for example, the brightness levels in an image or the temperature at a point in a continuum. Laeser et al [**19**] state that, as far as images are concerned, '... *adjacent pixels usually have brightness levels that are close in value*'. Similarly, the temperature difference between two neighbouring points in a continuum is likely to be small compared to the absolute values of the temperatures. Laeser et al highlight the fact that compression of correlated data can be achieved simply by coding the differences between adjacent values, given an absolute starting value or DC value. The removal of the correlation between adjacent values through the difference calculation generates, in effect, text data.

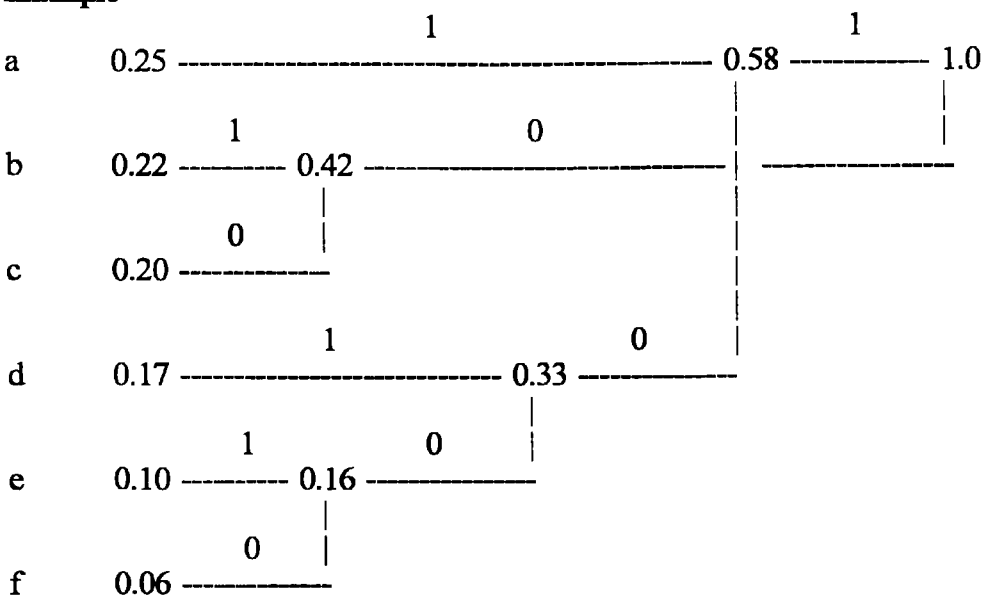
In text data, itself, any particular character is not linked to that preceding or following it in terms of numerical magnitude but rather through the recurrence of particular patterns within the text as a whole. Compression of a group of characters, known as a phrase, is possible if reference to a previous occurrence of the same phrase can be made by a code which is shorter than the phrase it replaces; i.e., dictionary-based compression. Compression of text is also possible by assigning short codes to characters occurring frequently and longer codes to characters which occur less often i.e. statistical compression. An example of statistical

compression is the Morse code for coding English text. Letters occurring frequently such as 'e' and 't' are assigned the shortest codes while less frequently occurring characters such as 'x' and 'z' are assigned longer codes. Dictionary-based compression and statistical compression may both be *static* or *adaptive*. Static statistical compression, for example, requires that the frequency of occurrence of individual characters be determined in a first pass over the data. Then, with the codes computed from the frequencies, the data may be coded on a second pass. Static statistical compression incurs an overhead penalty in the form of a table of characters and their individual frequencies/probabilities. Adaptive statistical compression adjusts the frequencies and recomputes the codes as each character is encountered on a single pass, i.e. it adapts to the data.

3.2 Huffman Coding

Huffman coding is an example of a statistical data compression algorithm. The Huffman algorithm involves the construction of a binary tree. At each stage of the algorithm the two lowest probabilities are combined and treated as a single new probability. This is continued until a single value (1.0) is reached. The Huffman code for each symbol is generated by working backwards to that symbol, labelling the path from a combined value with a 0 or a 1.

Example



The upper paths have been labelled with a 1 and the lower with a 0. Working backwards from right to left the codes for the characters are:

a	11
b	01
c	00
d	101
e	1001
f	1000

3.3 Arithmetic Coding

Huffman coding necessarily represents each symbol by an integral number of bits. Each symbol must be represented by at least one bit irrespective of its probability. Arithmetic coding codes groups of symbols rather than individual symbols and in effect represents individual symbols with fractions of bits. Every symbol is allocated a segment of the range [0,1) in direct proportion to its probability of occurrence. As each symbol is encountered the working range is reduced to the proportion allocated to that symbol.

Example

Assume the following symbols and their respective probabilities of occurrence:

a	0.1	[0.0,0.1)
b	0.4	[0.1,0.5)
c	0.1	[0.5,0.6)
d	0.1	[0.6,0.7)
e	0.2	[0.7,0.9)
f	0.1	[0.9,1.0)

Let the sequence to be coded be cabdeb. Coding begins:

start [0,1)
c [0.5,0.6)
a [0.5,0.51)
b [0.501,0.505)
d [0.5034,0.5038)
e [0.50368,0.50376)
b [0.503688,0.50372)

The sequence is then represented by any number within this final range.

3.4 LZ77 Dictionary Compression

LZ77 uses a sliding window [**20**] consisting of two parts. The first of these is the larger of the two and forms a dictionary which is used to code the second part. This second part is known as the look-ahead buffer and the LZ77 algorithm matches strings of symbols in the look-ahead buffer with the same strings in the dictionary. The strings are coded by tokens or '*triples*' consisting of

a. the position of the start of the string in the dictionary

b. the length of the string

c. the first symbol in the look-ahead buffer that follows the string.

As an example consider the sequence

a c b a b a b b a b c a e **a c b a a b b a** b c c a b a c a

where the large bold characters form the look-ahead buffer and the preceding form the dictionary. The first four characters in the look-ahead buffer occur in the dictionary at position one. These characters may be coded as [1 - 4 - 'a']. The window then moves five characters along:

a b b a b c a e a c b a a **b b a b c c a b** a c a .....

The first five characters in the look-ahead buffer occur in the dictionary at position two. These characters may be coded as [2 - 5 - 'c'].

The offset-position - length pair is termed a '*pointer*'. If a character is encountered for the first time then then the pointer component of the triple is wasted. The LZSS [21,22] compressor allows a '*free mixture of pointers and characters*'. LZSS dispenses with the character following the pointer and prefixes pointers and characters with a single bit for their identification. If LZSS were used to compress a large file then few characters would be expected to be coded alone and the output would consist mainly of pointers.
Consider the LZSS coding of the same sequence as above:

a c b a b a b b a b c a e **a c b a a b b a** b c c a b a c a

Once again the first four characters in the look-ahead buffer occur at position one. The output is: '1' * [1 - 4]. The window now moves four characters along:

b a b b a b c a e a c b a **a b b a b c c a** b a c a .....

The first six characters in the look-ahead buffer occur at position two. The output is:

'1' * [2 - 6].

There is a break even point in LZSS. If ξ is the number of bits used to represent the displacement and τ is the number of bits used to represent the match length then the smallest match length coded by a pointer is $\lceil (1 + \xi + \tau)/(1 + 8) \rceil$ ($\lceil q \rceil$ is the least integer not smaller than q); i.e. if $(1 + \xi + \tau) = a(1 + 8)$ for $n < a < n + 1$ (n integer) then it is more economical to code matches of length n as individual characters.

In LZSS the elements of the pointer are of a constant fixed length. LZB [**21**] phases in the offset position so that if, for example, there are fifty symbols in the dictionary thus far, then the offset position need only be $\lceil \log_2 50 \rceil = 6$ bits in length - there is no preloading of the dictionary. The match length in LZB is represented by a simple variable length code, γ. The following is a slight variation on the code γ although the codeword lengths are the same:

n	**γ(n)**
1	1
2	010
3	011
4	00100
5	00101
6	00110
7	00111
8	0001000

etc.

If p represents the smallest number of characters coded in a pointer then a match of length m is coded as $\gamma(m-p+1)$. The length of $\gamma(m-p+1)$ is $2\lfloor \log_2(m-p+1) \rfloor + 1$ bits ($\lfloor q \rfloor$ is the greatest integer not larger than q). The best value of p should be determined by experiment.

3.5 LZ78 Dictionary Compression

LZ78 compression [**23**] uses the whole string of the previously seen symbols as a dictionary and not a small part of it as in LZ77. As in LZ77, however, LZ78 codes strings with tokens. These tokens consist of:

a. label of previously seen string

b. symbol following previously seen string

The new string is added to the dictionary. The mechanics of this algorithm can be understood by considering a simple example.

The input string is 'abbccddd'.

Symbol(s)	**Token**		**Dictionary**	
Coded	**Label**	**Symbol**	**Number**	**String**
a	0	a	1	a
b	0	b	2	b
bc	2	c	3	bc
c	0	c	4	c
cd	4	d	5	cd
d	0	d	6	d
dd	6	d	7	dd

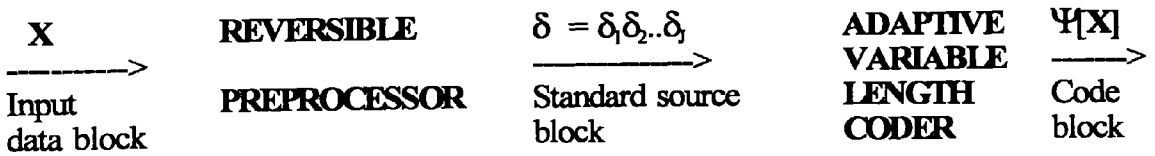
3.6 The Rice Algorithm

3.6.1 Overview of the Rice Algorithm

The Rice Algorithm was developed in the 1970's by R. F. Rice of the Jet Propulsion Laboratory in California for use in deep space missions [**24,25,26,27,28**].

This algorithm performs data compression by reducing the statistical redundancy of the data and works adaptively by selecting the optimal of several Huffman-equivalent [**29**] codes on blocks of J samples.

The algorithm works in two stages. The first of these is termed preprocessing and is designed to remove the correlation between the samples. The second stage compresses the processed data in groups, choosing for each group the optimum code.



3.6.2 Preprocessing

The preprocessing stage removes the correlation within the data and outputs samples that are independent and with a probability distribution that facilitates good compression by the second stage.

The first part of the preprocessor is a simple differencing calculation. The input data block, **X**, of integers is transformed into a block of differences

$$\Delta = \Delta_1\Delta_2....\Delta_j$$

where $\Delta_i = x_i - x_{i-1}$.

The second part maps each Δ_i into a standard source δ_i with the following operations

$$\delta_i = 2\Delta_i \qquad 0 \le \Delta_i \le \theta$$

$$\delta_i = 2|\Delta_i| - 1 \qquad -\theta \le \Delta_i < 0$$

$$\delta_i = \theta + |\Delta_i| \qquad \text{otherwise}$$

where $\theta = \min(x_{i-1}, x_{max} - x_{i-1})$ and $x_{max} = 2^n - 1$, for n-bit data.

The δ will closely approximate a Standard Form Source, as defined in the next section.

3.6.2.1 Standard Form Source (s)

A Standard Form Source has the following properties:-

a) Samples of s are positive integers 0,1,2,.....

b) Samples of s are independent

c) The probabilities of source symbols are ordered such that

$P(s_j=0) \ge P(s_j=1) \ge P(s_j=2)$ .....

i.e. the smaller integers are more likely to occur.

3.6.3 The Fundamental Sequence

The Fundamental Sequence denoted by ψ_1 is the starting point for all Rice compression. The fundamental sequence coding of a positive integer m is simply m zeroes followed by a 1. The

fundamental sequence coding of a sequence of integers $\mathbf{x}=x_1x_2x_3...x_J$ is the concatenation of the individual fundamental sequences corresponding to each integer.

Example: if the sequence **x** = 203104 then the fundamental sequence corresponding to **x** is $\psi_1(\mathbf{x})$ = 0011000101100001

3.6.4 The Basic Compressor

The Basic Compressor is able to compress data lying within a relatively narrow entropy range. Despite its having been superseded by Split Samples coding [**30,31**], which can compress data over a far greater entropy range, the Basic Compressor is included here for completeness. In order to understand the Basic Compressor options it is necessary to define some terms.

3.6.4.1 Third Extension

The third extension of a sequence of 0's and 1's is the grouping of these symbols into groups of three with the last group packed with zeroes if necessary. For example, the third extension of the sequence 11010110001 is 110 101 100 010.

3.6.4.2 Coding of the Third Extension

The third extensions are coded according to the following table:

Third Extension	Code
000	1
001	001
010	010
100	011
011	00000
101	00001
110	00010
111	00011

The Basic Compressor consists of four coding options:

$\psi_0(x)$ coded third extension of the complement of the fundamental sequence corresponding to x.

$\psi_1(x)$ the fundamental sequence corresponding to x.

$\psi_2(x)$ coded third extension of the fundamental sequence corresponding to x.

$\psi_3(x)$ the binary equivalent of x.

The option chosen to code a particular sequence will, obviously, be the option that produces the shortest code. The complete coding for that sequence will be the two bit identifier of the option used followed by the code ψ_i.

The complement is used in the first of these four options as it allows the efficient coding of strings of zeroes. If the data were **s** = 000000 then the fundamental sequence code for **s** would be 111111 and its complement would be 000000. The option ψ_0 would be 11.

3.6.4.3 Example

Consider the sequence **s** = 12041003.

The fundamental sequence corresponding to **s** is

ψ_1 = 01001100000101110001

The complement of ψ_1 is

comp(ψ_1) = 10110011111010001110

and the third extension of the complement is

comp(ψ_1)^3 = 101 100 111 101 000 111 000

This last sequence can be coded using the above table to find

ψ_0 = 00001011000110000110000111

The third extension of the fundamental sequence is

ψ_1^3 = 010 011 000 010 111 000 100

which can be coded using the above table to give

ψ_2 = 010000001010000111011

The binary representation of the sequence is

ψ_3 = 001010000100001000000011

The lengths of the code options computed above are:

ψ_0 = 25 bits

ψ_1 = 19 bits

ψ_2 = 21 bits

ψ_3 = 24 bits

The shortest complete coding of the example sequence is the fundamental sequence preceded by the two-bit identifier of the option used

01010011000010111000 1

Fortunately, it is not necessary to apply each code option and then choose the one which produces the shortest code. There is a decision rule based on the length of the fundamental sequence.

The length, F, of the fundamental sequence for a sequence of J integers is

$$F = J + \Sigma x_i$$

Define variables L0,L1,L2,L3 corresponding to the four code options:

$L0 = (F/3) + 2(F - J)$

$L1 = F$

$L2 = (F/3) + 2J$

$L3 = Jn$

The option chosen to code the data block is simply the option corresponding to the smallest variable Li.

3.6.5 Split Samples

The technique of split samples uses the fundamental sequence as a starting point and allows the efficient coding of higher entropy data than that which can be coded efficiently by the basic compressor. Consider the following data values and their respective probabilities:

Value	Prob.	Huffman Code	Fund. Seq.
0	0.55	1	1
1	0.25	01	01
2	0.12	001	001
3	0.05	0001	0001
4	0.02	00001	00001
5	0.01	00000	000001

The fundamental sequence coding of these data is only very slightly less efficient than the Huffman code and for J data samples with this probability distribution the fundamental sequence would be the code option chosen. Now suppose that the data δ are

16 12 10 7 2 0 1 7

The fundamental sequence for these data would be 63 bits long. The five bit binary coding of these data is

16	10000
12	01100
10	01010
7	00111
2	00010
0	00000
1	00001
7	00111

Split samples assumes that the least significant bits of the binary code are random and therefore cannot be compressed. If the k least significant bits are split off then the data may be expressed as

$\delta = \mathbf{M}^k * \mathbf{L}_k$

where $\mathbf{M}^k$ are the most significant bits, * now denotes concatenation and $\mathbf{L}_k$ are the least significant bits. As k increases the $\mathbf{M}^k$ retain the standard form source probability ordering - the δ are assumed to be preprocessed.

If the two least significant bits are assumed to be random then the most significant bits are (base 10):

4 3 2 1 0 0 0 1

If these are now coded using Huffman coding then

Value	Freq.	Huffman Code	Fund. Seq.
	1		
0	3----------8	1	1
	1 0 |		
1	2----------5----	01	01
	1 0 |		
2	1-------3----	001	001
	1 0|		
3	1---2---	0001	0001
	0|		
4	1---	0000	00001

Fundamental sequence coding of the most significant bits is efficient and the optimal split samples code of the original data is then the fundamental sequence for the most significant bits plus the least significant bits :-

$\psi_{1,2}(\delta) = \psi_1(M^2) * \mathbf{L}_2$ i.e. 0000100010010111101000010111000011 1 - 35 bits

and this is more efficient than the simple base 2 representation of 40 bits. The entropy where $\psi_{1,k}$ achieves its best performance is:

$H_5^k \approx k + 2$ bits per sample.

As with the Basic Compressor there is a decision rule so that there is no need to compute each possible option and choose the shortest.

If F_0 is the length of fundamental sequence without sample splitting, J is the number of sample values to be compressed and k is the number of least significant bits per sample then the boundaries to adjacent $\psi_{1,k}$ decision regions are given by:

$F_0 = J/2 + J(2^{k+1})$ bits

Any $\psi_{1,k}$ option will generate a longer code than ψ_{bu}, the data itself, if:

$F_0 > [(n-k)2^{k+1} + 1 - 2^k] J/2$

where n is the number of bits required to represent the uncompressed data [**10,32**]. For a six option compressor with n-bit data the decision regions are :

Option ID	Option	Region
000	$\psi_{1,0}$	$F_0 \leq 5J/2$
001	$\psi_{1,1}$	$5J/2 < F_0 \leq 9J/2$
010	$\psi_{1,2}$	$9J/2 < F_0 \leq 17J/2$
011	$\psi_{1,3}$	$17J/2 < F_0 \leq 33J/2$
100	$\psi_{1,4}$	$33J/2 < F_0 \leq (32n-143)J/2$
101	ψ_{bu}	$(32n-143)J/2 < F_0$

3.6.6 The Performance of the Rice Algorithm

The Rice algorithm performs well in comparison with other, better known algorithms. This has been illustrated by Venbrux et al [**33**] in the lossless compression of image data. The results of their comparative work are summarized below:

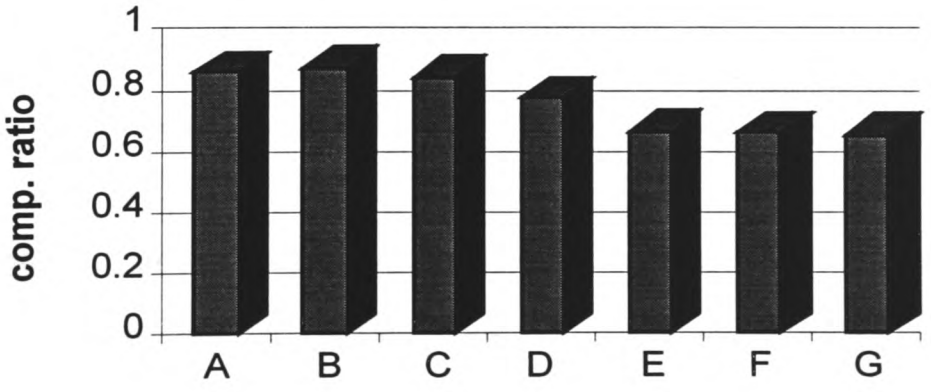


Fig. 3.1 Relative Performance of the Rice Algorithm (comp. ratio = output size/input size)

where

A LZ78 without preprocessing
B Adaptive Huffman without preprocessing
C Arithmetic without preprocessing
D LZ78 with preprocessing
E Adaptive Huffman with preprocessing
F Arithmetic with preprocessing
G Rice coding

It can be seen that the performance of Rice coding is comparable with that of arithmetic coding with preprocessing for this sort of data. Rice coding has the advantage of being more easily integrated into an error tolerant scheme.

3.6.7 The Split Samples Block Size

The block size J is an important variable in the Rice algorithm. A low value allows the compressor to adapt more quickly to changes in data entropy. Too low a value means that the overhead of the code option ID becomes significant. Values of fifteen or sixteen are generally accepted as ideal [33].

Yeh et al [29] have shown how the Rice algorithm can be related to a Huffman code for data having a Laplacian distribution. They also show how the optimum Rice option changes as the parameter of the Laplacian distribution changes.

The following example demonstrates how the optimum Rice option adjusts as the data distribution changes. The optimum Rice block length is thereby determined.

Example

Suppose the data are $\delta_i = i$, $i = 1 \ldots 2^M$ This is a rather specific sequence but it does allow a fully analytic solution to be obtained.

Let the block length J be $J = 2^n$. The length of the fundamental sequence without sample splitting for the j^{th} block of J is given by

$$F_{0_j} = J + \sum_{i=(j-1)J+1}^{i=jJ} i = J + \frac{J}{2}((j-1)J+1+jJ) = \frac{1}{2}(3J+J^2(2j-1)) \qquad (3.1)$$

The number of split bits required for block j, k_j, is found from:

$$\frac{J}{2} + J2^{k_j} < F_{0_j} \leq \frac{J}{2} + J2^{k_j+1} \qquad (3.2)$$

Therefore k_j is the integer below

$$\log_2\left[\frac{(F_{0_j} - \frac{J}{2})}{J}\right] = \log_2(1 + J(j - \frac{1}{2})) \qquad (3.3)$$

Putting $J=2^n$ and rearranging, k_j is the integer below

$$n + \log_2(j - \frac{1}{2} + \frac{1}{2^n}) \qquad (3.4)$$

If $n \geq 2$ then the following k^j values are found:

j	k_j
1	n-1
2	n
3	n+1
4	n+1
5	n+2
6	n+2
7	n+2
8	n+2
9	n+3
10	n+3

etc.

The estimated length of the $\psi_{l,k}$ Rice code option [**10**] is

$$L(\psi^j_{1,k_j}) = 2^{-k_j} F_{0_j} + \frac{J}{2}(1-2^{-k_j}) + Jk_j \qquad \textbf{(3.5)}$$

There are 2^{M-n} blocks of J values. The sum L of the lengths of the Rice options is ($M \neq n$)

$$2^{-(n-1)}\left(\frac{1}{2}(3.2^n+2^{2n})\right)+\frac{2^n}{2}(1-2^{-(n-1)})+(n-1)2^n+$$

$$\sum_{a=1}^{(M-n)}\sum_{b=1}^{(2^{a-1})}\left(2^{-(n+a-1)}\frac{1}{2}(3.2^n+2^{2n}(2(2^{a-1}+b)-1))+\frac{2^n}{2}(1-2^{-(n+a-1)})+(n+a-1)2^n\right)$$

(3.6)

where $2^{a-1} + b = j$.

$$L=2+2^{n-1}+n2^{n}+\sum_{a=1}^{(M-n)}\sum_{b=1}^{(2^{a-1})}(2^{1-a}+b2^{n+1-a}+n2^{n}+a2^{n}+2^{n-1}-2^{n-a}) \qquad (3.7)$$

$$L=2+2^{n-1}+n2^{n}+\sum_{a=1}^{M-n}(2^{a-1}(2^{1-a}+n2^{n}+a2^{n}+2^{n-1}-2^{n-a})+2^{n+1-a}(1+2^{a-1})2^{a-2})$$

(3.8)

$$L=2+2^{n-1}+n2^{n}+\sum_{a=1}^{M-n}(1+2^{a}(n+1)2^{n-1}+a2^{n+a-1})$$

Using $S_n = x + x^2 + x^3 + ... + x^n = (x^{n+1} - x)/(x-1)$

and $T_n = x + 2x^2 + 3x^3 + ... + nx^n = (nx^{n+1} - S_n)/(x-1)$

$$L=2+2^{n-1}+n2^{n}+M-n+(n+1)(2^{M}-2^{n})+(M-n-1)2^{M}+2^{n} \qquad (3.9)$$

$$L=2+2^{n-1}+M-n+M2^{M} \qquad (3.10)$$

(If M = 5 and n = 3 then a manual split-samples coding of the data is 168 bits long - if M = 5 and n = 3 are placed into the above expression then L = 168)

The number of bits per sample value is given by

$$\frac{2+2^{n-1}+M-n+M2^{M}}{2^{M}}+\frac{1}{2^{n}}\lceil \log_2 N_{opt} \rceil \qquad (3.11)$$

where N_{opt} is the number of options within the compressor.

So that the analysis is valid there must be a sufficient number of options to allow the optimum number of split bits to be chosen:- the number of options must be greater than or equal to M+2.

Compression performance is optimized if

$$\frac{2^{n-1}-n}{2^M}+\frac{1}{2^n}\lceil \log_2 N_{opt} \rceil \qquad (3.12)$$

is minimized.

Let $Z_n = 2^{n-1} - n + 2^M\lceil \log_2 N_{opt} \rceil 2^{-n}$

Z_n is a minimum if

$Z_{n-1} - Z_n > 0$ and $Z_{n+1} - Z_n > 0$

After some manipulation it is found that

$$2^{2n}-4.2^n-4.2^M\lceil \log_2 N_{opt} \rceil < 0 \qquad (3.13)$$

and

$$2^{2n}-2.2^n-2^M\lceil \log_2 N_{opt} \rceil > 0 \qquad (3.14)$$

Therefore

$$1 + \sqrt{(1 + 2^M\lceil \log_2 N_{opt} \rceil)} < 2^n < 2 + 2\sqrt{(1 + 2^M\lceil \log_2 N_{opt} \rceil)} \qquad (3.15)$$

or

$$\log_2(1 + \sqrt{(1 + 2^M\lceil \log_2 N_{opt} \rceil)}) < n < 1 + \log_2(1 + \sqrt{(1 + 2^M\lceil \log_2 N_{opt} \rceil)}) \qquad (3.16)$$

Sample Results

1 If M = 10 and N_{opt} = 16 then 6.02 < n < 7.02, i.e. n = 7 (J = 128) for optimum performance.

2 If M = 6 and N_{opt} = 8 then 3.90 < n < 4.90, i.e. n = 4 (J = 16) for optimum performance.

3 If M = 30 and N_{opt} = 32 then 16.16 < n < 17.16, i.e. n = 17 (J = 131072) for optimum performance.

In the second sample result the data values 1 to 64 are broken into four blocks of 16 for the best compression performance. In the first result the same values 1 to 64 are compressed together with the values 65 to 128 in a single block for optimum performance. Similarly, in the third result the first 1024 (2^{10}) values are now compressed in a single block with 130048 other values.

Chapter Four

Robust Data Compression

4.1 Error Propagation in Compressed Data

Lelewer and Hirschberg [34], in their review of data compression, considered the susceptibility to errors of data compression schemes. They examined the tendency of some Huffman codes to self-synchronize. This property is illustrated in a later section. Ferguson and Rabinowitz [11] gave an example of a situation in which the Huffman code can never synchronize. Suppose the Huffman code is:

A	01
B	10
C	11
D	000
E	001

and consider the following sequence repeated an infinite number of times:

ADABCDABCDBECAABDECA

Now suppose the second bit is in error so that the decoder receives

000 000 11 01 10 000 11 01 10 001 000 11 10 10 11 000 000 11 10 1

The first three bits are decoded to give D. Decoding the above sequence alone leaves the very last bit, i.e. the second bit of the last A to be combined with the bits at the start of the

repeated sequence to form a codeword. The first three bits of the repeated sequence, when preceded by the last bit of the prior sequence are decoded to give BB exactly, with no spare bits. The cycle repeats and the decoder stays out of synchronization permanently.

One of the most recent of the many attempts to improve the self-synchronization of Huffman codes is that due to Escott and Perkins [35] who improved upon the codes of Ferguson and Rabinowitz. The total inability of arithmetic codes to tolerate errors was recognised by Lelewer and Hirschberg, Teuhola and Raita [36] and Kobler [37]. Lelewer and Hirschberg state that '*there is no evidence that adaptive methods are self-synchronizing*'. This includes, of course, the methods of Ziv and Lempel; Woolley [38] gave a detailed account of how errors would affect data compressed using LZ78. The Rice algorithm was originally designed for the compression of pictures captured in space missions on a line by line basis so that a whole line might be lost through error propagation. Rice suggests that [39] '*these effects can be countered to a certain extent by incorporating more frequent updates ...* '.

4.2 Preventing the Propagation of Errors in Compressed Data - Piecewise Compression

The process of locating the start of a valid coding after the occurrence of errors can be difficult. This process shall be termed *weak synchronization*. *Strong synchronization* shall refer to the process where synchronization is achieved *without* slippage in the output data. A self-synchronizing Huffman code will provide examples of each form of synchronization. The code is:

a	**1**
b	**01**
c	**001**
d	**000**

The sequence a.d.a.c.a.b is coded as 1.000.1.001.1.01.

If the first 1 becomes a 0 then decoding gives 000.01.001.1.01 or d.b.c.a.b. Weak self-synchronization has occurred since there is slippage in the decoded symbols. If the first 0 becomes a 1 then decoding gives 1.1.001.001.1.01 or a.a.c.c.a.b. Strong self-synchronization has occurred since the number of decoded symbols matches the number of encoded symbols. The basic strategy explored in this chapter is to perform piecewise compression with relatively frequent synchronization. A piecewise approach confines an error to the piece in which the error occurs. Synchronizing sequences at the ends of the pieces act as barriers to error propagation. There is no alternative to the piecewise compression approach with adaptive compression algorithms since locating the start of a valid codeword is not enough. The interpretation of that codeword depends upon the interpretation of the preceding code. An incorrect interpretation of the code preceding a codeword will cause that codeword to be interpreted incorrectly. Piecewise compression requires that synchronization be forced rather than awaited, as with self-synchronizing Huffman codes. Forcing weak synchronization requires the addition of redundancy to the compressed data stream in the form of a synchronizing sequence/bit pattern.

4.2.1 Clear Codewords

A codeword, here, refers to a synchronizing sequence/bit pattern. In the context of Huffman codes a *clear codeword* is a bit sequence which cannot be formed by the concatenation of other codewords but could be formed through the effect of errors. The definition of clear codewords here needs to be extended. The Rice algorithm allows, as a final option, the passing through of the data. Therefore, a clear codeword in the present case is a bit pattern which cannot be formed through the concatenation of split samples components and/or the data. The reading of the clear codeword provides the starting point for future correct code interpretation. Clear codewords have been developed for synchronization in digital video applications [**40**] where the clear codeword was introduced as part of the code itself in the construction process. It is worthwhile investigating clear codewords for synchronization of Rice coding.

In the digital video application the clear codeword was a string of 1's with a 0 at the end and so a word with a similar construction is sought for the Rice algorithm and evaluated. Note that if the Rice algorithm cannot compress the data then the data is passed through uncompressed and every possible combination of 0's and 1's could be generated through the concatenation of the data values. The clear codeword sought is a string of 1's or a string of 0's so such patterns should not be allowed. Long sequences of 1's or 0's in the compressed data can be prevented if the strings corresponding to the uncompressed data values are separated by a 0 or a 1 respectively. In the last chapter it was seen that the boundaries to adjacent $\psi_{1,k}$ decision regions are given by $F_0 = J/2 + J(2^{k+1})$ bits. The maximum length of fundamental sequence codes occurring may be found by assuming that the sample values consist of J-1 zero data values plus one non-zero integer. If $\psi_{1,k}$ is the option chosen then the maximum possible length of fundamental sequence (FS_{max}) is

$$2^{k}\,[J/2 + J(2^{k+1}) - J] + 1 = 2J - J(2^{-(k+1)}) + 1$$

where multiplying on the left hand side by 2^{-k} is the binary equivalent of moving the decimal point k places to the left.

If J is sixteen and the compressor has six options ($k_{max} = 4$) then FS_{max} is 33 bits (32 0's and a 1). A good candidate for a clear codeword would appear to be a string of 0's followed by a 1 and which is longer than FS_{max}. If the data do pass uncompressed then the values would be separated with a 1. The length of the required clear codeword could be reduced to a value nearer FS_{max} by preventing the formation of long strings of 0's through concatenation of the least significant bits and the ID codes would also require some modification, e.g to disallow 000. However, the clear synchronizing codewords for Rice coding are not efficient. The codewords are relatively long and further redundancy is required to make them work. As well as these shortcomings, the codewords could be formed by chance when the code is affected by errors. Mere recognition of a clear codeword, therefore, would not be sufficient to establish synchronization. It seems that codewords should only give an indication of synchronization rather than a guarantee. That being so, far shorter codewords could serve this purpose. This chapter develops this theme of using short synchronizing sequences which might occur naturally in the compressed data stream. False synchronization is minimized by additional means. These sequences are used not only with Rice coding but also with mixed Rice/LZ77 coding and LZB.

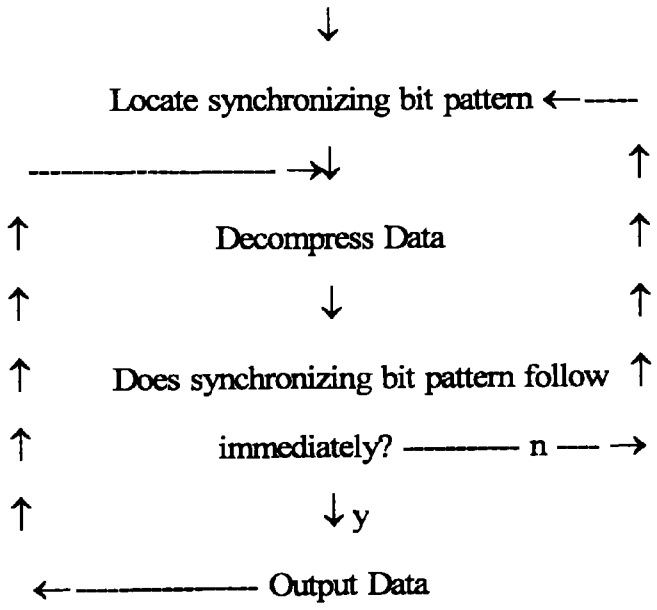
4.2.2 A Robust Synchronizing Scheme

The following scheme is designed to recover when the synchronizing pattern occurs naturally in the data. It is possible to use a pattern which occurs naturally in the data as a synchronizing sequence if it is placed *periodically* into the data stream. Correct, weak synchronization can

be achieved if the synchronizing bit patterns occur at expected positions. The compressed data stream takes the form:

... sync compressed data sync compressed data sync ...

The synchronization scheme is represented below:



This synchronization scheme provides only weak synchronization. The establishment of strong synchronization requires the addition of a little more redundancy. A repeated *cyclic count* is introduced to number the groups of data between synchronizing sequences and establish strong synchronization. If an error occurs and data is lost the count allows the data that is decompressed subsequently to be placed correctly into the output data stream by shifting along by the number of values lost. The compressed data stream now takes the form:

... sync compressed data count...count sync compressed data count...count sync...

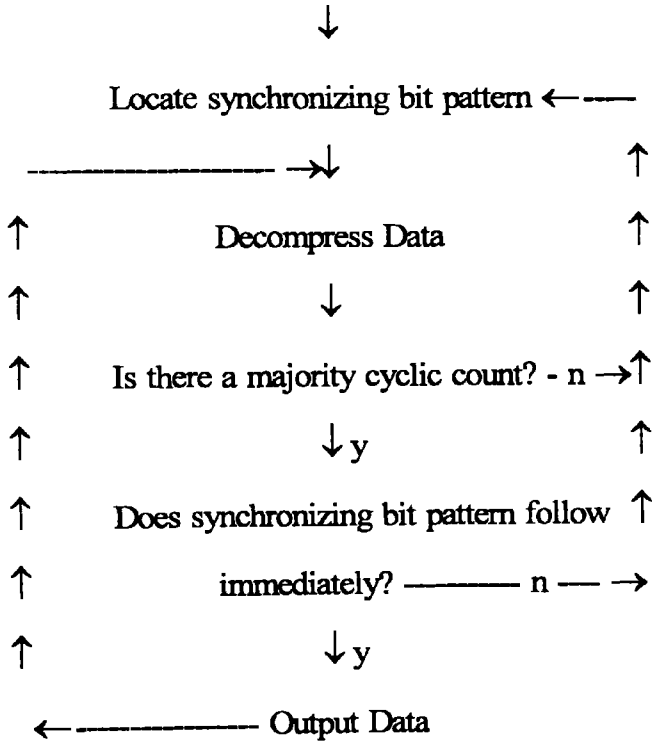
The count is of a fixed length of eight bits. The Reed-Solomon symbols of the error correction codes are of eight bits so that a single symbol error will usually affect up to two counts. The count is, therefore, written more than twice to make the reading of a valid count much more likely. For example, if the count is written four times then the count is:

a. One of four identical counts; or

b. One of three identical counts; or

c. One of two identical counts, the other two being different one from the other.

The complete synchronization scheme becomes:



An alternative form of cyclic count was considered and tested. A number of zero bits equal to the count itself was inserted into the bit stream. The count would be determined simply by counting the number of bits between the end of the compressed data code and the following synchronizing sequence. Magnitude errors affecting the count bits would, in theory, be inconsequential since only the number of count bits is important. Unfortunately, trials of this

form of count, even with quite modest error rates, proved unsatisfactory. Errors affecting one or more variable length components of the compressed data changed the 'length' of the code so that the number of count bits was misread.

A great advantage of the form of count adopted is that it is, itself, a form of synchronizing sequence. The use of mathematically-based error correction codes to protect the count would necessarily assume that the detected synchronizing sequence immediately following be an actual synchronizing sequence before decoding could begin. The form of count adopted makes no such assumption and it is only when a valid count is detected that the test for the following synchronizing sequence is undertaken. This is a very powerful guard against false synchronization. Also, if the count were protected by error correction and synchronization is otherwise deemed to have been established it is conceivable that severe errors could cause false error correction. There would be no way of determining whether or not this had taken place and the count produced would be assumed 'correct'. More powerful codes could be employed but these would be far longer than the form of count adopted. Moreover, the time taken to decode such codes would be prohibitive.

A 'global' strategy for decoding a group of cyclic counts could also be considered. A major disadvantage of any such strategy is that the data would need to be buffered on decompression. One such strategy might take a group of cyclic counts and protect the group with parity to form a codeword for error correction. The cyclic counts and parity would be collected together for error correction. Apart from time taken for error correction, the major drawback of this approach is that if errors were to cause the loss of synchronization, even briefly, then one or more counts would not be picked up and there would be vacant positions in the codeword. Error correction would be impossible.

4.2.3 Error Tolerant Compressors in High Error Environments

This chapter describes the use of several different compression algorithms in the proposed synchronized piecewise compression strategy. The data type and piece size vary also. The power of the synchronized compression strategy in every case is best illustrated when it is subject to random errors. Conceivably there may only be a single resynchronization required after an isolated burst error. Frequent random errors will force the strategy to regain synchronization on an ongoing basis. Typical random bit error rates range from 10^{-4} to 10^{-5} which fall well within the error correction capability of DDS [**17**]. It is not inconceivable that in high levels of radiation higher random bit error rates could be induced. Uncorrected random errors are readily observed in the DDS simulation when the random bit error rate exceeds 0.004. In the next section confirmation of the power of the synchronization scheme in the case of random errors will be followed by an examination of data recovery in the presence of more realistic burst errors.

4.3 Error Tolerant Compression of Correlated Physical Data

The Rice algorithm has been chosen as the basis for the strategy to compress correlated physical data as, by its very nature, it works piecewise so that the overall piecewise compression strategy may be employed without otherwise reducing compression performance.

4.3.1 The Error Tolerant Compressor

A computer simulation of the robust data compression scheme described earlier in this chapter has been developed in which compression is performed using the Rice algorithm. Recall that very short synchronizing sequences are used which might appear naturally in the compressed data. Periodic insertion of the sequence into the compressed data stream allows weak synchronization to be established. The insertion of a cyclic count allows the conversion from weak to strong synchronization. The compression program asks the user for the Rice block length, J, and for the number of Rice blocks, F_s, between synchronizations. The complete compressed data stream which is used in producing the results presented here is:

$\leftarrow$---- F_s---- $\rightarrow$

... sync ref id code ... id code ref count count count count sync ...

where id is the identifier of the Rice option used. The synchronizing sequence is relatively short - 10011101. This was chosen deliberately to have low correlation for a few bits slippage. The reference starting value is repeated as a check for errors. If there is a mismatch then synchronization is deemed not to have been established.

The simulation has been written in PASCAL which is run in conjunction with the simulation of DDS. The end-to-end simulation is run in four stages for ease of file management:

1 Compression of the data

2 DDS encoding of the compressed data

3 Introduction of errors and DDS decoding of data and errors

4 Decompression of decoded data and comparison with original

4.3.2 Compression Performance

The robust synchronization scheme, incorporating the Rice algorithm for the compression of sampled correlated physical data, has been tested. The test data was a set of eight-bit terrain elevations on a rectangular grid. Consider five different cases in which the Rice block length and the number of blocks between synchronizations is varied. Compression performance in each case is shown below where T denotes the number of values compressed into a DDS Basic Group of 126632 bytes. The value T represents the total number of compressed data values appearing between synchronizing sequences summed over the Basic Group. If the very last data byte of the Basic Group does not contain a synchronizing sequence then there will be some compressed data in the Basic Group which does not fall between two synchronizing sequences. That data is not counted.

Case	Rice block length	F_s	T	Compressed Size (bits per value) adjust	Compressed Size (bits per value) non-adjust
I	12	10	244541	4.1424	3.7460
II	16	4	227110	4.4608	3.7221
III	16	8	247422	4.0944	3.7223
IV	16	16	261883	3.8680	3.6816
V	16	24	266035	3.8080	3.6833

The compression performance improves as the number F_s of blocks between synchronizations increases. This improvement has arisen since the overhead decreases when F_s increases.

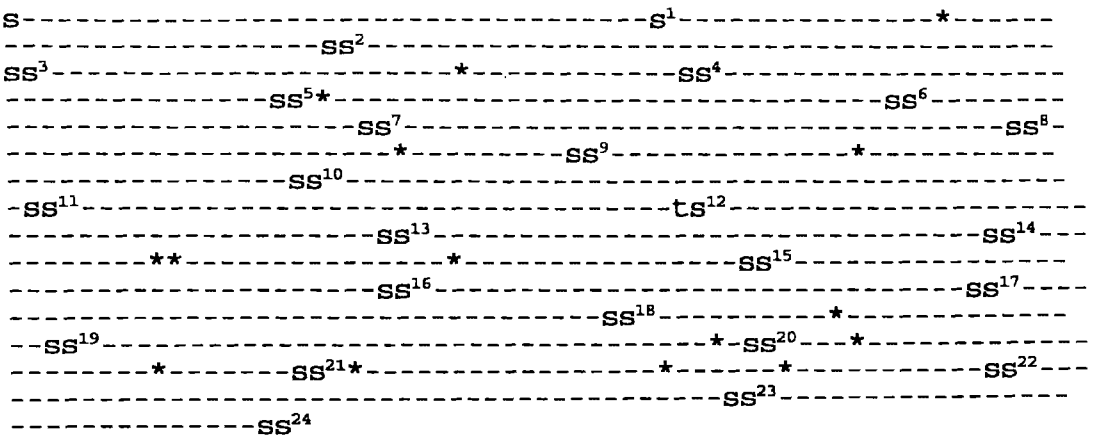
4.3.3 Random Errors

The performance of the strategy in the presence of random errors was investigated with the parameters of cases I and II above. These were chosen to test the resynchronization strategy most rigorously. Random bit error rates as high as 0.01 were considered in thirty two simulations at each error rate. No loss of synchronization was observed. The corrected byte error rates are shown in **Section 2.5.2**. It was conjectured that if R represents the proportion of values decompressed correctly and b_e represents the corrected byte error rate then:

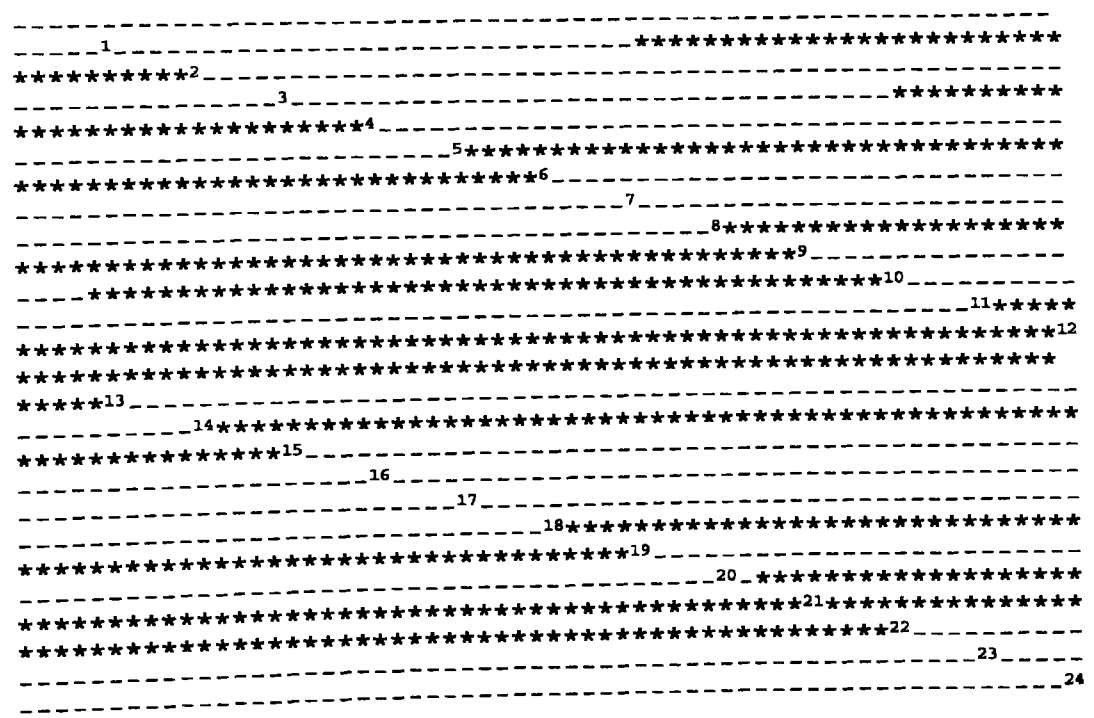
$$R = (1 - b_e)^N$$

The validity of this relationship was confirmed by taking logarithms and performing regression analysis. In case I and case II the coefficient of determination was very close to one. In case I the value of N was calculated to be 40.356 and in case II the value of N was 25.527. In case II, for example, if the corrected byte error rate is 0.0013 then 97.73% of the data is recovered correctly. This relationship is reasonable because if there are P bytes, say, between synchronization sequences, then the probability that every byte is correct is $(1 - b_e)^P$.

In **Section 4.2.3** it was argued that a high random error rate would best test the power of the synchronization strategy. An example of compressed data in case II is shown below after DDS decoding in a high random error rate environment where the initial random error rate was 0.007. Little s signifies an error-free byte which contains all or part of a synchronizing sequence and little t represents an erroneous byte which contains all or part of a synchronizing sequence. All other error-free bytes are denoted by a '-' and all other erroneous bytes are labelled '*'. The superscripts label the preceding Rice block.



The decompression of the above yields the following where error-free data values are denoted by a '-' and erroneous or 'missing' values are denoted by a '*'. The superscripts label the ends of the decompressed Rice blocks.



Blocks 1, 3, 5, 7, 8, 11, 14, 16, 17, 18, 23 and 24 are error-free on exit from the decoder and so are decompressed correctly. The compressed blocks 2, 4 and 10 each contain a single byte error and around half of the data in each case is decompressed correctly. Blocks 6, 9, 15, 19, 21 and 22 contain one or more errors in the compressed data and the whole of each block is lost save in the case of block 21 where a single value (the reference value) is recovered correctly. The synchronizing sequence between blocks 12 and 13 is in error and so each block is lost with the present synchronizing scheme. Note that in block 20 there is single byte error affecting the cyclic count. The repetition code protecting the count deals with this and the whole of block 20 is recovered correctly.

4.3.4 Burst Errors

The interleaving within the DDS encoding improves the error correction capability of the codes. However, when errors are uncorrected by C3 they are dispersed throughout the data on deinterleaving. The data bytes within an erroneous Track (half a Frame) will be spread throughout the Sub-Group. The data bytes corrupted by a longitudinal corruption the length of a coded Basic Group and which remain uncorrected will be spread throughout the Basic Group on decoding. The C3 code is able to correct any two tracks in a Basic Group. If strong synchronization is to be maintained when there are more than two Tracks in error a complete period of the cyclic count must not be lost. The strategy is clear - to increase the number of consecutive Frames whose corruption can be tolerated, the number of coded values between synchronizing sequences must be increased. This is data dependent as for higher entropy data the lengths of the codes will be longer. Define N_f to be the maximum number of consecutive

Frames whose corruption through the loss of entire Tracks allows strong synchronization to be maintained. The value of N_f in each of cases II to V are:

Case	N_f
II	1
III	2
IV	5
V	8

Since there is no interleaving between Frames then it follows that data recovery is proportional to the number of uncorrupted Frames.

The error control procedures of DDS are able to correct a longitudinal corruption up to 0.3 mm wide. Corruption widths are increased in steps of $^1/_{80}$ mm in the simulation of the error control procedures of DDS. Longitudinal corruptions the length of a Basic Group or more, cannot be tolerated in cases IV and V. Data recovery in cases II and III is illustrated below:

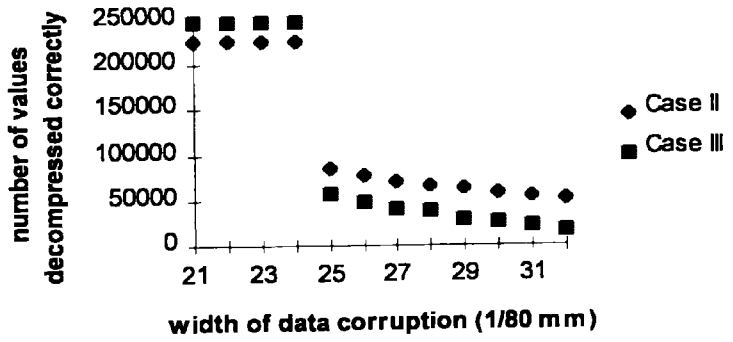
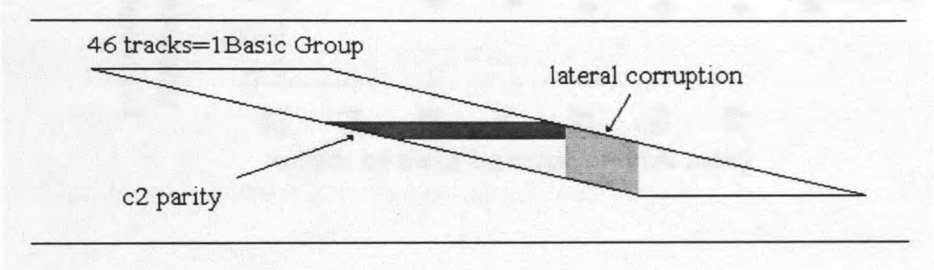


Fig. 4.1 Data Recovery within a DDS Basic Group in the Presence of a Longitudinal Corruption

In case II the compressed data stream between synchronizing sequences is smaller than in case III so that more intact individual compressed data streams appear between deinterleaved errors. Hence the number of values recovered correctly is greater in case II. The errors do not propagate beyond the Basic Group.

A lateral corruption has been input into the DDS simulation and data recovery in case II has been determined. This investigation is illustrated below:



The C2 code can correct a burst of up to 24 blocks (**p 8**). Correction of 24 blocks would be achieved if the first block in the burst had an even serial number. In this case no C1 codewords not wholly contained within the same 24 blocks would be corrupted (**p 7**). If the first block were to have an odd serial number then corruption of 24 blocks would lead to uncorrectable C1 codewords in the block immediately preceding the burst and the block immediately following the burst. This would be uncorrectable by the C2 code since some C2 codewords would contain seven symbol errors. Consideration of the geometry and dimensions of the tracks suggested that a lateral corruption is best simulated by incrementing by one the serial number of the start block in each succeeding track affected by the burst. Data recovery in the presence of a lateral corruption is illustrated in the graph below. Start 1 labels a corruption with a starting position as in the diagram above. Start 2 labels a corruption with

a starting position shifted to the left of that in the above diagram by about $^1/_6{}^{th}$ of the length of the 'user data' part of a track. A corruption with a width of 24 blocks would be alternately correctable and uncorrectable in succeeding tracks.

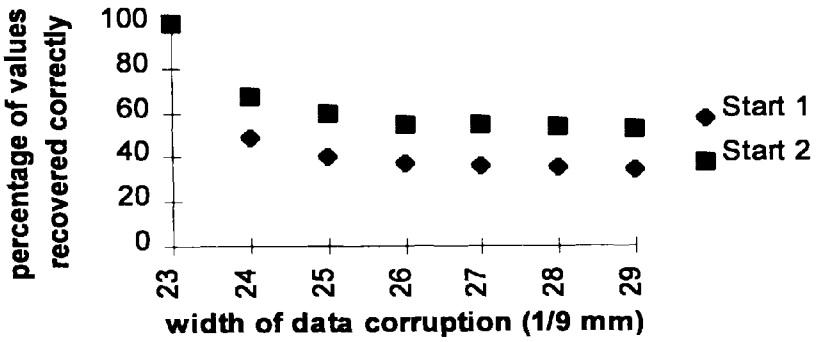


Fig. 4.2 Data Recovery within a DDS Basic Group in the Presence of a Lateral Corruption

Data recovery in the presence of a lateral corruption is greater than in the presence of the equivalent (in terms of blocks) longitudinal corruption. As may be seen in the earlier diagram a lateral corruption will affect the 'non-data' C2 parity while a longitudinal corruption may not.

4.4 Error Tolerant Compression of Mixed Correlated Physical and Text Data

The compression of two quite different types of data requires an eclectic compressor. The compressor must identify the type of data which it is encountering and then select a compression algorithm which can perform effective compression of that data.

The Rice algorithm may be regarded as possessing an in-built text detector since if the algorithm is unable to compress the data, as would be the case for text, then, as a final option the data is passed through uncompressed. The approach adopted here is to use the statistically-based Rice algorithm to compress the whole of the data. However, the final option is replaced by compression based upon the LZ77 dictionary algorithm [**20**]. Hence LZ77 must be made to work more efficiently when compressing small pieces.

4.4.1 The Error Tolerant Compressor

If an LZ77 based algorithm were used to compress a large file then few characters would not fall within a previously seen phrase. The output of the LZ77 derivative, LZSS [**21,22**] would consist mainly of pointers i.e. the pair identifying the position of the start of the match in the dictionary and the match length. Since the volume of data to be compressed by the proposed strategy may be only tens of characters, consecutive pointers in the output code are unlikely. The proposed text compressor, therefore, is LZSS but with pointers followed by characters as in the triples of LZ77. The pointer in LZ77 is a waste of capacity when there is no matching phrase. An example of the code produced by the proposed compressor follows later in this section.

The implementation of the proposed strategy will be described with reference to the following example. Suppose that there are four Rice blocks of five samples each between synchronizations and that the strategy encounters the following:

ref	**block1**					**block2**					**block3**					**block4**				
10	10	11	11	12	13	98	6	70	31	20	21	22	24	25	26	98	6	70	32	60

where ref is the starting value for the preprocessing of the Rice algorithm. The Rice algorithm detects automatically that block1 and block3 comprise correlated samples since the blocks can be compressed by sample splitting. Block1 and block3 are examined firstly. The differences and preprocessed values for block1 and block3 are:

	block1					**block3**				
differences	0	1	0	1	1	1	1	2	1	1
preprocessed values	0	2	0	2	2	2	2	4	2	2

The first option of the Rice compressor, fundamental sequence coding of the samples without sample splitting, is chosen by the algorithm to compress block1, yielding:

1 001 1 001 001

Similarly, the algorithm chooses the second option of the Rice compressor (with one split bit) to compress block3 to give

01 01 001 01 01 * 0 0 0 0 0

The Rice algorithm determines, by default, that block2 and block4 comprise text data since these blocks cannot be compressed by sample splitting. When data is deemed to be text data, the dictionary compression of the raw, unpreprocessed data is attempted. Preprocessing is not used because it would shorten runs of characters and recurring groups of characters would be shortened. This is best illustrated with a simple example:

suppose the data is

... 2 4 7 7 7 7 9

preprocessing would yield

... ? 4 6 0 0 0 4

i.e. the run length has been shortened from four to three.

The dictionary is preloaded with '0's and the look-ahead buffer is empty. At the start of the compression of groups of blocks between synchronizations there can be no matching phrase and since there are no '0's in block2 the dictionary compressor passes the values in this block straight through. Each symbol is prefixed by a single bit, 0, to denote a character rather than a pointer followed by a character. At the start of the compression of block4 the dictionary is

look ahead buffer

... 0 0 0 0 0 98 6 70 31 20 ▌ 98 6 70 32 60

The first three characters in the look ahead buffer appear in the dictionary. Block4 is encoded as:

< 1 'offset into dictionary' '3' '32' > < 0 '60' >

where

1 is the bit indicating that a pointer follows

'3' is the match length

'32' is the character immediately following the match in the look ahead buffer

0 is the bit indicating that a character follows

'60' is the last character in the look ahead buffer

By contrast LZ77 would code the look ahead buffer as

< 'offset into dictionary' '3' '32' > < 'offset into dictionary' '0' '60' >

while LZSS would code the look ahead buffer with

< 1 'offset into dictionary' '3' > < 0 '32'> < 0 '60' >

The compression program asks the user for the Rice block length, J, and for the number of Rice blocks, F_s, between synchronizations. The complete compressed data stream which is used in producing the results presented here is:

$\leftarrow$---- F_s---- $\rightarrow$

... sync ref id code ... id code ref count count count count sync ...

where id is the identifier of the Rice option used. The synchronizing sequence is again 10011101.

In order to assess the performance of this scheme a simulation has again been written in PASCAL and is run in conjunction with the simulation of DDS. The end-to-end simulation is run in the same four stages of the earlier tests.

4.4.2 Compression Performance

The test data is part of that generated in an industrial measurement procedure and each value is of eight bits. The following diagram illustrates the mix of 'housekeeping' text and correlated physical data.

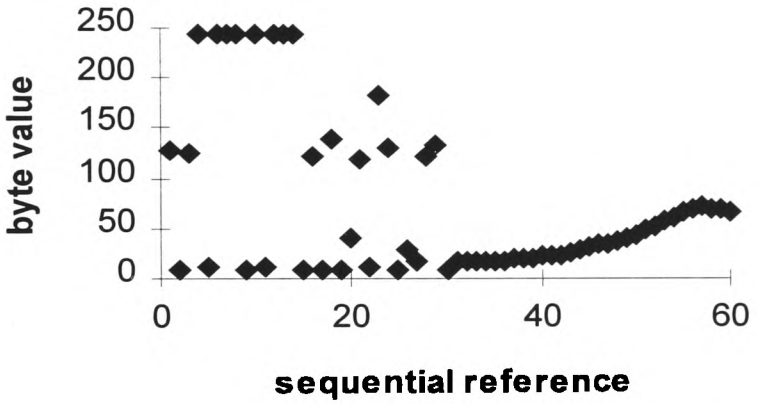


Fig. 4.3 The Mix of Data

Bytes one to twenty nine are text data while bytes thirty to sixty are correlated physical data values. The entropy of the unpreprocessed data is around 5.1 bits per value and a 'continuous' LZSS coding with a 2K buffer achieves a performance of 3.33 bits per value. The Rice block length used in producing the following figures is sixteen. The number of options is varied; three, four, five and eight option compressors are considered. The options for an r-option compressor are split samples coding with 0 ... r-2 split bits and LZ77 coding. The numbers of values compressed into a DDS Basic Group using these compressors are shown below

together with the percentage of the number of blocks compressed using split samples (ss) to the number of blocks compressed using LZ77:

Case	no. of options	no. of blocks between syncs	no. of values compressed	Percentage of use of ss to use of LZ77	Compressed Size (bits per value)
I	3	8	222138	49:51	4.5608
II	4	8	225750	50:50	4.4872
III	5	8	223557	51:49	4.5312
IV	8	8	200208	81:19	5.0600
V	4	16	255715	51:49	3.9616
VI	4	16	253659	51:49	3.9936

In case VI the dictionary is preloaded with the least frequently occurring symbol which arises 0.00005% of the time, as against zero, which is the most frequently symbol, occurring 11.54% of the time, in cases I to V. This makes it clear that no prior knowledge of the data is required for the attainment of powerful compressor performance. A compressor with four options and sixteen blocks between synchronizations provides the best performance of the compressors tested. The difference between cases II and V is due to the improved performance of LZ77 when the dictionary is able to become longer between synchronizations. The differences in performance between cases I and II and also between cases III and IV arise solely from variations in individual compressor option performances since the option ID overhead is of the same length in the respective cases. There are blocks of data which are compressed by LZ77 in case I but compressed by split samples with two split bits in case II. Recall that split samples with k split bits works best for entropies of around $k+2$ bits per sample. The poorer performance in case I suggests that split samples outperforms LZ77 at entropies of around four bits per sample. In case III, LZ77 compression is used on data which case IV compresses by

split samples with from four to six split bits. The improved performance of case III indicates that LZ77 performs better than split samples from entropies of around six bits per sample through to eight bits per sample.

4.4.3 Random Errors

An investigation of the performance of this strategy in the presence of random errors as in **Section 4.3.3** was performed in cases II and IV above. The same form of relationship of the previous section emerged again between the corrected byte error rate and the proportion of values decompressed correctly i.e. if R represents the proportion of values decompressed correctly and b_e represents the corrected byte error rate then:

$$R = (1 - b_e)^N$$

In case II the value of N was calculated to be 26.626 and in case IV the value of N was 32.529. So, for example, in case II, with a corrected byte error rate of 0.00037, the proportion of values decompressed correctly is 0.9902.

4.4.4 Burst Errors

The data bytes within a data corruption the length of a coded Basic Group which remain uncorrected will be spread throughout the Basic Group on decoding. The error control procedures of DDS are able to correct a data corruption up to 0.3 mm wide. Widths are

increased in steps of $^1/_{80}$ mm in the simulation. Data corruptions the length of a Basic Group or more, cannot be tolerated in cases IV and V since the length of the code between synchronizations is too long to appear between deinterleaved errors. Data recovery in case I is illustrated below:

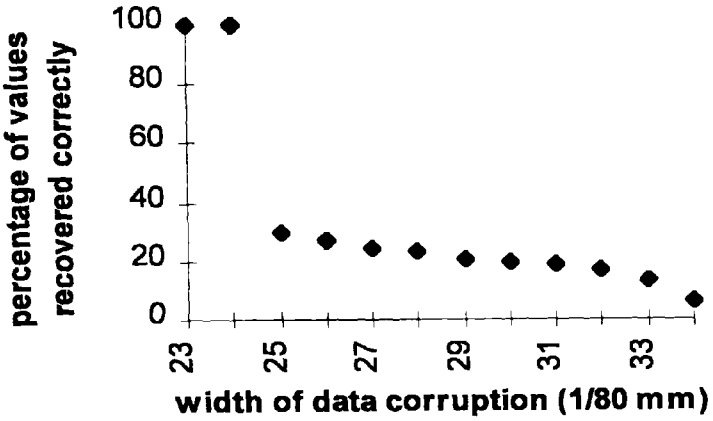


Fig. 4.4 Data loss within a DDS Basic Group

The pattern of data recovery is similar to that in the case of pure correlated physical data. The errors do not propagate beyond the end of the Basic Group.

4.5 Error Tolerant Compression of Text Data

Dictionary-based compressors offer the most effective compression of text data. Performance figures for the compression of text with several different compressors may be found in the literature. The book of Nelson [22], for example, quotes figures showing that LZSS compresses certain text files to around 40% of their original size while Huffman coding and

arithmetic coding compress the same files to around 60% of their original size. The use of dictionary-based compressors with the piecewise compression strategy, however, will attenuate compression performance since the dictionary will never be of optimum size. Of the two Ziv Lempel compressors an LZ77-based compressor is favoured over an LZ78-based compressor since the former begins to perform compression almost immediately after start-up. The LZSS compressor has been described earlier [**§3.4**] together with the improvements made by LZB.

4.5.1 LZB

The advantages of using LZB when performing piecewise compression may be appreciated through the examination of a simple example. Consider the coding of the following sequence in isolation using both LZSS and LZB.

Position	0	1	2	3	4	5	6	7
Character	**a**	**b**	**a**	**b**	**b**	**a**	**b**	**c**

Consider the use by LZSS of six bits for the displacement and four bits for the match length so that the smallest match length coded by a pointer is $\lceil(1 + 6 + 4)/(1 + 8)\rceil = 2$. In LZSS the match length is coded as match length minus 2. For this example, the minimum match length that is coded by LZB, p, is fixed at 2.

Characters coded	LZSS	LZB
a	1 'a'	1 'a'
b	1 'b'	1 'b'
ab	0 000000 0000	0 0 1
bab	0 000001 0001	0 01 010
c	1 'c'	1 'c'

The advantages of using LZB rather than LZSS are strikingly apparent. The code for the displacement into the dictionary and the code for the match length are both shorter in this example when LZB is used. During piecewise compression such space saving advantages accumulate since there many such start-ups. The 'continuous' compression of large files would not demonstrate such a great advantage since the displacement codes will be the same length once the sliding window has filled up. Moreover, the variable length coding of the match length may begin to claw back some of the advantage gained when long matches are found.

4.5.2 LZB Piecewise Compression Performance

The LZB algorithm was programmed in PASCAL to perform piecewise compression. The piecewise LZB compression of 'Book1' of the Calgary data compression corpus [**41**] was performed. The parameter 'p' of LZB is fixed at two in all cases save for compression using a piece size of 16K. The best compression performance with a piece size of 16K is achieved when p is increased from two to three after 8K (2^{13}) values have been compressed within each piece. Compression revealed the following frequency distribution of match lengths:

Match Length	**Piece size (bytes)**					**Variable p**	
	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14} <=2^{13}	>2^{13}
2	127885	123605	108069	86114	63187	31588	13890 *
3	39734	49170	55521	57631	54726	27270	21534
4	16355	21665	26590	30638	33308	16734	17681
5	7325	10160	13419	17087	20543	10313	13197
6	3178	4835	6942	9332	11763	5962	8583
7	1516	2428	3564	5006	6786	3413	5227
8	828	1328	1949	2766	3835	1942	3174
9	485	805	1220	1680	2294	1169	1941
10	273	446	652	933	1308	662	1104
11	158	246	367	526	720	356	658
12	93	162	243	332	451	198	418
13	67	97	133	187	233	116	226
14	53	70	97	129	185	80	150
15	29	50	76	101	116	62	107
16	29	33	37	53	66	31	62
17	78	102	135	168	207	104	142

* the number of matches of length two in this case is shown for information only since they are ignored by the compressor. It is evident from the above table that as the piece size increases so the distribution of match lengths skews so that more long matches are found. The performance, in bits per sample, of piecewise LZB on three files of eight bit data from the Calgary data compression corpus for varying piece sizes is shown below:

File name		**Book1**	**Book2**	**News**
Number of individual characters		82	96	98
Entropy (bits per sample)		4.53	4.79	5.19
Performance at piece size				
measured in K -	**½**	5.72	5.35	5.76
	1	5.30	4.86	5.31
	2	4.98	4.48	4.87
	4	4.73	4.18	4.48
	8	4.53	3.94	4.25
	16	4.22	3.66	3.97
LZB (Fenwick [42])		3.86	3.28	3.55

The entropy acts as a lower bound on the performance of a Huffman code. A piece size of 2K allows LZB to outperform Huffman coding in two of the three cases above. In reality, the performance differential is greater still, since Huffman coding has the encumbrance of a probability table. Generally speaking, it may thought that, at the very least, text should be compressed to around half of its original size before such a performance may be regarded as satisfactory. Some text may not allow such compression to be achieved. Each of the three texts examined above do allow such a performance with 'continuous compression'. The piecewise compression of files Book2 and News allow compression ratios of nearly 0.5 with pieces of size 8K and 16K respectively. Although the piecewise compression of Book1 did not exhibit the same performance with any of the piece sizes tested, the performance for this

file did approach that of 'continuous' LZB most closely. The compressed size with a piece size of 16K was 9.3% greater than that of 'continuous' compression for Book1 while the same figures for Book2 and News were 11.6% and 11.8% respectively.

4.5.3 Error Propagation in LZB

The LZB code comprises a mixture of flag-pointer and flag-character pairs. The effects of errors in the various components are summarized below:

1	erroneous flag	loss of compressed code synchronization leading to total scrambling of subsequent data
2	erroneous character	may propagate if referenced by later pointers
3	erroneous offset	- may propagate if referenced by later pointers - may reference non-existent locations in dictionary
4	erroneous length	- loss of data synchronization - slippage - loss of code synchronization if leading zeroes corrupted - loss of data synchronization followed by later loss of code synchronization since interpretation of displacement dependent upon number of values compressed.

Most of the effects of the above errors are not easily demonstrated since a lot of scrambled rubbish is produced. However, an example of propagation due to an erroneous offset is shown below:

Original Text

The Canterville Ghost
A Hylo-Idealistic Romance
WHEN Mr. Hiram B. Otis, the American Minister, bought Canterville Chase, every one told him he was doing a very foolish thing, as there was no doubt at all that the place was haunted. Indeed, Lord Canterville himself, who was a man of the most punctilious honour, had felt it his duty to mention the fact to Mr. Otis when they came to discuss terms.
'We have not cared to live in the place ourselves,' said Lord Canterville, 'since my grand-aunt, the Dowager Duchess of Bolton, was frightened into a fit, from which she never really recovered, by two skeleton hands being placed on her shoulders as she was dressing for dinner, and I feel bound to tell you, Mr. Otis, that the ghost has been seen by several living members of my family, as well as by the rector of the parish, the Rev. Augustus Dampier, who is a Fellow of King's College, Cambridge. After the unfortunate accident to the Duchess, none of our younger servants would stay with us, and Lady Canterville often got very little sleep at night, in consequence of the mysterious noises that came from the corridor and the library.'

Corrupted Text

The Canterville Ghost
A Hylo-Idealistic Romance
WHEN Mr. Hiram B. Otis, te CAmerican Minister, bought Canterville Chase, every one told him e Cwas doing a very foolish teing, as te r Cwas no doubt at all teat te CplaceCwas eaunted. Indeed, Lord Canterville himself, who was a man of te CmostCpunctilious eonour, ead feltCitChis duty toCmention te Cfact toCMr. Otis when te y came toCdiscuss terms.
'We have not car d toClive in te CplaceCourselves,' said Lord Canterville, 'sinceCmy grand-aunt, te CDowagerCDuce ss of Bolton, was frighten d intoCa

fit, from which she never r ally r covered, by twoCskeleton hands being placed on her shoulders as she was dressing for dinner, and I feel bound to tell you, Mr. Otis, teat te CghostChas been seen by severalCliving members of my family, as well as by te Cr ctor of te Cparish, te CRev. Augustus Dampier, who is a Fellow of King's College, Cambridge. After te unfortunat Caccident toCte CDuce ss, none of our younger servants would stay with us, and Lady Canterville oftenCgot very little sleep at night, in consequenceCof te Cmysterious nois s teat came from te Ccorridor and te library.'

The '*he* ' within '*the Americ*an' is referenced to the same phrase in the title. A single bit error has shifted the reference forwards by one position so that '*e C*' is now copied on decompression.

4.5.4 The Error Tolerant Compressor

The core of the compressor is the algorithm LZB. A second form of variable length coding for the match lengths is included as an option. The inclusion of a second option for the coding of the match lengths is suggested by the distribution of match lengths found in **Section 4.5.2**. The distribution of match lengths, with smaller piece sizes, approximates a Laplacian distribution which the Rice algorithm is able to exploit with the fundamental sequence code. This can be seen clearly in the following example where the match lengths of the piecewise LZB coding of Book1 with a 1K piece have been coded:

Match Length	Frequency	Huffman Code	Fundamental Sequence	The γ Code
2	123605	1	1	1
3	49170	01	01	010
4	21665	001	001	011
5	10160	0001	0001	00100
6	4835	00001	00001	00101
7	2428	000001	000001	00110
8	1328	0000001	0000001	00111
9	805	00000001	00000001	0001000
10	446	000000001	000000001	0001001
11	246	00000000011	0000000001	0001010
12	162	00000000001	00000000001	0001011
13	97	000000000101	000000000001	0001100
14	70	000000000001	0000000000001	0001101
15	50	0000000000001	00000000000001	0001110
16	33	0000000000000	000000000000001	0001111
17	102	000000000100	0000000000000001	000010000
Total length (bits)		**394868**	**395216**	**444146**

Clearly, the fundamental sequence code is almost as good as Huffman and the γ code is not competitive here. The compressor performs LZB compression on each piece in the normal way and writes the compressed code to a buffer. In parallel, and at no extra run-time penalty as far as searching for matches is concerned, the fundamental sequence coding of the Rice algorithm is applied to the adjusted match lengths and the compressed code is written to a second buffer. The compressor then chooses the shorter code for this piece and inserts it into the main compressed data stream preceded by a single-bit flag to identify the variable length code employed.

In order to achieve robustness in a similar way to that of the earlier compressors the compressed data stream takes the form

... sync flag-bit lz-code count count count count sync ...

The synchronizing sequence has been increased in length for the error resistant compressor from the eight bits of the two prior compressors to fourteen bits - 10011101010110 for the following reason. The compressed code between synchronizing sequences is a great deal longer than in the compressors for correlated physical data and mixed correlated physical data and text. If synchronization is lost the eight bit pattern would be detected far more frequently than in the prior compressors and, in triggering the resynchronizing scheme each time synchronization is lost, the decompressor would be slowed markedly. The cyclic count and the form of its coding is retained.

4.5.4.1 Random Errors

The table in **Section 4.5.2** reveals that the compression performance of LZB on the file News with a piece size of 2K is very similar to its performance on file Book2 with a piece size of 1K. Compression of Book1 with a piece size of 3K exhibits a similar performance. The use of the synchronizing scheme in each of these three cases shows directly how piece size affects data recovery. It turns out that the same expression for data recovery in the presence of random errors which applied to correlated physical data and to mixed correlated physical data and text also applies in the case of pure text data; i.e. if R is the proportion of values decompressed correctly and b_c is the corrected byte error rate then

$$R = (1 - b_e)^N$$

The exponent, N, is found by taking logarithms and performing regression analysis. The results of this exercise are shown below.

	Book1@3K	News@2K	Book2@1K
Values compressed into Basic Group	208896	202752	205824
Compressed size(bits per value)	4.85	5.00	4.92
Ratio of use of FS code to gamma code	100:0	44:56	73:27
Exponent	386	286	145
Predicted proportion of values decompressed correctly at byte error rate of 0.00037	0.8669	0.8996	0.9477

It can be seen that the fundamental sequence coding of the match lengths proved to be the shorter option for the majority of pieces compressed. The option to use fundamental sequence coding allows a small improvement in performance over 'straight' LZB. The improvement has not been measured.

4.5.4.2 Burst Errors

The codes between synchronizations in the case of pure text data are necessarily long to allow acceptable compression performance. This precludes any data recovery from within Frames affected by longitudinal scratches since the codes are too long to appear between deinterleaved errors. The long codes do allow strong synchronization to be maintained when the tapes is

affected by quite large burst errors. For example, the compression of Book1 with a piece size of 3K shows that $(208896/(3\times2^{10})) = 68$ blocks may be compressed into a Basic Group. Recalling that the count period is 256, it can be seen that strong synchronization can be maintained when three whole Basic Groups are corrupted by errors.

4.6 Broad Mathematical Model of Data Recovery for Strong Synchronization

There are around 10,000 Basic Groups on a 60m tape. A compressed data file is stored on the tape as an Entity. The number of Basic Groups which make up an Entity, of course, depends upon the size of the data file. If an error occurs on the tape, errors will only propagate to the end of the Entity and not to the end of the tape.

Let Entities comprise an average of N Basic Groups. Suppose an error corrupts an entire Basic Group so that no data is recovered from the Basic Group in which the error occurs. Let the probability of a Basic Group being uncorrupted be p.

Case 1 - Synchronization and a cyclic count.

Errors will not propagate beyond the 'end' of a Basic Group. There may be some 'edge effects' in the following Basic Group where data recovery will start from the first valid synchronizing sequence.

The number of Basic Groups recovered is approximately Np.

Case 2 - No synchronization

If an error occurs in a Basic Group all subsequent Basic Groups in the Entity will be lost.

$$E(recovery) = Np^N + \sum_{i=1}^{N-1} ip^i(1-p)$$

$$E(recovery) = \frac{p-p^{N+1}}{1-p}$$

If there are 100 Basic Groups in an Entity and the probability of a Basic Group's being uncorrupted is 0.9 then case 1 yields 90 Basic Groups and case 2 yields 9 Basic Groups.

Chapter Five

DDS-3 Revisited

5.1 Preamble

The tape format DDS-3 was described briefly in **Chapter One**. This chapter lays the foundation for a possible future simulation, based upon that of DDS-1, of this latest format by providing the parity byte equations and providing an indication of the impact of uncorrected burst errors on data compressed using the present strategy.

5.2 The Parity Byte Equations

The C1 parity check matrix is [4]

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \ldots\ldots & 1 & 1 & 1 \\ \alpha^{61} & \alpha^{60} & \alpha^{59} & \alpha^{58} & \ldots\ldots & \alpha^{2} & \alpha & 1 \\ \alpha^{122} & \alpha^{120} & \alpha^{118} & \alpha^{116} & \ldots\ldots & \alpha^{4} & \alpha^{2} & 1 \\ \alpha^{183} & \alpha^{180} & \alpha^{177} & \alpha^{174} & \ldots\ldots & \alpha^{6} & \alpha^{3} & 1 \\ \alpha^{244} & \alpha^{240} & \alpha^{236} & \alpha^{232} & \ldots\ldots & \alpha^{8} & \alpha^{4} & 1 \\ \alpha^{50} & \alpha^{45} & \alpha^{40} & \alpha^{35} & \ldots\ldots & \alpha^{10} & \alpha^{5} & 1 \end{bmatrix} \qquad (5.1)$$

and V^T is

$$[V1 \;\; V2 \;\; V3 \;\; \ldots \;\; V56 \;\; P1 \;\; P2 \;\; P3 \;\; \ldots \;\; P6] \qquad \textbf{(5.2)}$$

A computer program has been written to solve the equation H*V = 0 (here * means matrix multiplication again). The program has been shown to be running correctly by reproducing the equations in **Chapter Two** which were previously produced by hand. The DDS-3 C1 parity bytes are:

$$\begin{aligned}
P_1 = {} & \alpha^{54}V_1 + \alpha^{254}V_2 + \alpha^{185}V_3 + \alpha^{56}V_4 + \alpha^{96}V_5 + \alpha^{154}V_6 + \alpha^{6}V_7 + \alpha^{200}V_8 + \alpha^{189}V_9 + \alpha^{40}V_{10} + \\
& \alpha^{0}V_{11} + \alpha^{154}V_{12} + \alpha^{183}V_{13} + \alpha^{201}V_{14} + \alpha^{69}V_{15} + \alpha^{243}V_{16} + \alpha^{8}V_{17} + \alpha^{61}V_{18} + \alpha^{207}V_{19} + \\
& \alpha^{15}V_{20} + \alpha^{174}V_{21} + \alpha^{242}V_{22} + \alpha^{190}V_{23} + \alpha^{220}V_{24} + \alpha^{51}V_{25} + \alpha^{162}V_{26} + \alpha^{237}V_{27} + \\
& \alpha^{16}V_{28} + \alpha^{61}V_{29} + \alpha^{239}V_{30} + \alpha^{53}V_{31} + \alpha^{206}V_{32} + \alpha^{141}V_{33} + \alpha^{214}V_{34} + \alpha^{217}V_{35} + \\
& \alpha^{93}V_{36} + \alpha^{160}V_{37} + \alpha^{201}V_{38} + \alpha^{39}V_{39} + \alpha^{83}V_{40} + \alpha^{171}V_{41} + \alpha^{51}V_{42} + \alpha^{42}V_{43} + \\
& \alpha^{174}V_{44} + \alpha^{33}V_{45} + \alpha^{92}V_{46} + \alpha^{192}V_{47} + \alpha^{180}V_{48} + \alpha^{76}V_{49} + \alpha^{177}V_{50} + \alpha^{162}V_{51} + \\
& \alpha^{108}V_{52} + \alpha^{225}V_{53} + \alpha^{205}V_{54} + \alpha^{228}V_{55} + \alpha^{166}V_{56}
\end{aligned} \qquad \textbf{(5.3)}$$

$$P_2=\alpha^{216}V_1+\alpha^{45}V_2+\alpha^{189}V_3+\alpha^{109}V_4+\alpha^{204}V_5+\alpha^{110}V_6+\alpha^{139}V_7+\alpha^{254}V_8+\alpha^{104}V_9+\alpha^{2}V_{10}+$$

$$\alpha^{150}V_{11}+\alpha^{162}V_{12}+\alpha^{226}V_{13}+\alpha^{221}V_{14}+\alpha^{82}V_{15}+\alpha^{239}V_{16}+\alpha^{49}V_{17}+\alpha^{197}V_{18}+\alpha^{144}V_{19}+$$

$$\alpha^{124}V_{20}+\alpha^{79}V_{21}+\alpha^{163}V_{22}+\alpha^{153}V_{23}+\alpha^{213}V_{24}+\alpha^{185}V_{25}+\alpha^{31}V_{26}+\alpha^{117}V_{27}+$$

$$\alpha^{245}V_{28}+\alpha^{187}V_{29}+\alpha^{9}V_{30}+\alpha^{6}V_{31}+\alpha^{123}V_{32}+\alpha^{253}V_{33}+\alpha^{15}V_{34}+\alpha^{115}V_{35}+$$

$$\alpha^{237}V_{36}+\alpha^{51}V_{37}+\alpha^{110}V_{38}+\alpha^{46}V_{39}+\alpha^{25}V_{40}+\alpha^{107}V_{41}+\alpha^{53}V_{42}+\alpha^{92}V_{43}+$$

$$\alpha^{163}V_{44}+\alpha^{175}V_{45}+\alpha^{69}V_{46}+\alpha^{82}V_{47}+\alpha^{138}V_{48}+\alpha^{15}V_{49}+\alpha^{203}V_{50}+\alpha^{100}V_{51}+$$

$$\alpha^{169}V_{52}+\alpha^{46}V_{53}+\alpha^{187}V_{54}+\alpha^{169}V_{55}+\alpha^{0}V_{56} \qquad \textbf{(5.4)}$$

$$P_3=\alpha^{244}V_1+\alpha^{145}V_2+\alpha^{173}V_3+\alpha^{51}V_4+\alpha^{195}V_5+\alpha^{156}V_6+\alpha^{33}V_7+\alpha^{70}V_8+\alpha^{96}V_9+\alpha^{110}V_{10}+$$

$$\alpha^{50}V_{11}+\alpha^{250}V_{12}+\alpha^{172}V_{13}+\alpha^{202}V_{14}+\alpha^{40}V_{15}+\alpha^{190}V_{16}+\alpha^{238}V_{17}+\alpha^{176}V_{18}+\alpha^{218}V_{19}+$$

$$\alpha^{254}V_{20}+\alpha^{126}V_{21}+\alpha^{6}V_{22}+\alpha^{12}V_{23}+\alpha^{114}V_{24}+\alpha^{116}V_{25}+\alpha^{103}V_{26}+\alpha^{179}V_{27}+$$

$$\alpha^{63}V_{28}+\alpha^{99}V_{29}+\alpha^{73}V_{30}+\alpha^{224}V_{31}+\alpha^{14}V_{32}+\alpha^{108}V_{33}+\alpha^{65}V_{34}+\alpha^{109}V_{35}+$$

$$\alpha^{73}V_{36}+\alpha^{133}V_{37}+\alpha^{194}V_{38}+\alpha^{148}V_{39}+\alpha^{225}V_{40}+\alpha^{242}V_{41}+\alpha^{182}V_{42}+\alpha^{32}V_{43}+$$

$$\alpha^{151}V_{44}+\alpha^{102}V_{45}+\alpha^{149}V_{46}+\alpha^{252}V_{47}+\alpha^{221}V_{48}+\alpha^{166}V_{49}+\alpha^{80}V_{50}+\alpha^{64}V_{51}+$$

$$\alpha^{45}V_{52}+\alpha^{45}V_{53}+\alpha^{201}V_{54}+\alpha^{89}V_{55}+\alpha^{134}V_{56} \qquad \textbf{(5.5)}$$

$$P_4 = \alpha^{17} V_1 + \alpha^{124} V_2 + \alpha^{224} V_3 + \alpha^{241} V_4 + \alpha^{88} V_5 + \alpha^{98} V_6 + \alpha^{30} V_7 + \alpha^{170} V_8 + \alpha^{118} V_9 + \alpha^{53} V_{10} +$$

$$\alpha^{109} V_{11} + \alpha^{101} V_{12} + \alpha^{211} V_{13} + \alpha^{99} V_{14} + \alpha^{227} V_{15} + \alpha^{99} V_{16} + \alpha^{140} V_{17} + \alpha^{61} V_{18} + \alpha^{148} V_{19} +$$

$$\alpha^{24} V_{20} + \alpha^{207} V_{21} + \alpha^{4} V_{22} + \alpha^{61} V_{23} + \alpha^{179} V_{24} + \alpha^{223} V_{25} + \alpha^{240} V_{26} + \alpha^{202} V_{27} +$$

$$\alpha^{76} V_{28} + \alpha^{123} V_{29} + \alpha^{191} V_{30} + \alpha^{239} V_{31} + \alpha^{183} V_{32} + \alpha^{205} V_{33} + \alpha^{126} V_{34} + \alpha^{110} V_{35} +$$

$$\alpha^{18} V_{36} + \alpha^{175} V_{37} + \alpha^{227} V_{38} + \alpha^{183} V_{39} + \alpha^{23} V_{40} + \alpha^{138} V_{41} + \alpha^{13} V_{42} + \alpha^{112} V_{43} +$$

$$\alpha^{42} V_{44} + \alpha^{41} V_{45} + \alpha^{27} V_{46} + \alpha^{28} V_{47} + \alpha^{87} V_{48} + \alpha^{200} V_{49} + \alpha^{182} V_{50} + \alpha^{147} V_{51} +$$

$$\alpha^{215} V_{52} + \alpha^{127} V_{53} + \alpha^{151} V_{54} + \alpha^{54} V_{55} + \alpha^{5} V_{56} \qquad \textbf{(5.6)}$$

$$P_5 = \alpha^{176} V_1 + \alpha^{103} V_2 + \alpha^{154} V_3 + \alpha^{243} V_4 + \alpha^{229} V_5 + \alpha^{197} V_6 + \alpha^{178} V_7 + \alpha^{118} V_8 + \alpha^{169} V_9 + \alpha^{26} V_{10} +$$

$$\alpha^{3} V_{11} + \alpha^{111} V_{12} + \alpha^{13} V_{13} + \alpha^{89} V_{14} + \alpha^{75} V_{15} + \alpha^{237} V_{16} + \alpha^{0} V_{17} + \alpha^{169} V_{18} + \alpha^{239} V_{19} +$$

$$\alpha^{160} V_{20} + \alpha^{183} V_{21} + \alpha^{36} V_{22} + \alpha^{10} V_{23} + \alpha^{179} V_{24} + \alpha^{239} V_{25} + \alpha^{43} V_{26} + \alpha^{35} V_{27} +$$

$$\alpha^{50} V_{28} + \alpha^{87} V_{29} + \alpha^{166} V_{30} + \alpha^{53} V_{31} + \alpha^{149} V_{32} + \alpha^{70} V_{33} + \alpha^{174} V_{34} + \alpha^{122} V_{35} +$$

$$\alpha^{225} V_{36} + \alpha^{71} V_{37} + \alpha^{220} V_{38} + \alpha^{167} V_{39} + \alpha^{9} V_{40} + \alpha^{142} V_{41} + \alpha^{115} V_{42} + \alpha^{149} V_{43} +$$

$$\alpha^{73} V_{44} + \alpha^{138} V_{45} + \alpha^{172} V_{46} + \alpha^{112} V_{47} + \alpha^{69} V_{48} + \alpha^{17} V_{49} + \alpha^{167} V_{50} + \alpha^{200} V_{51} +$$

$$\alpha^{249} V_{52} + \alpha^{248} V_{53} + \alpha^{184} V_{54} + \alpha^{210} V_{55} + \alpha^{176} V_{56} \qquad \textbf{(5.7)}$$

$$P_6=\alpha^{14}V_1+\alpha^{200}V_2+\alpha^{71}V_3+\alpha^{111}V_4+\alpha^{169}V_5+\alpha^{21}V_6+\alpha^{215}V_7+\alpha^{204}V_8+\alpha^{55}V_9+\alpha^{15}V_{10}+$$

$$\alpha^{169}V_{11}+\alpha^{198}V_{12}+\alpha^{216}V_{13}+\alpha^{84}V_{14}+\alpha^{3}V_{15}+\alpha^{23}V_{16}+\alpha^{76}V_{17}+\alpha^{222}V_{18}+\alpha^{30}V_{19}+$$

$$\alpha^{189}V_{20}+\alpha^{2}V_{21}+\alpha^{205}V_{22}+\alpha^{235}V_{23}+\alpha^{66}V_{24}+\alpha^{177}V_{25}+\alpha^{252}V_{26}+\alpha^{31}V_{27}+$$

$$\alpha^{76}V_{28}+\alpha^{254}V_{29}+\alpha^{68}V_{30}+\alpha^{221}V_{31}+\alpha^{156}V_{32}+\alpha^{229}V_{33}+\alpha^{232}V_{34}+\alpha^{108}V_{35}+$$

$$\alpha^{175}V_{36}+\alpha^{216}V_{37}+\alpha^{54}V_{38}+\alpha^{98}V_{39}+\alpha^{186}V_{40}+\alpha^{66}V_{41}+\alpha^{57}V_{42}+\alpha^{189}V_{43}+$$

$$\alpha^{48}V_{44}+\alpha^{107}V_{45}+\alpha^{207}V_{46}+\alpha^{195}V_{47}+\alpha^{91}V_{48}+\alpha^{192}V_{49}+\alpha^{177}V_{50}+\alpha^{123}V_{51}+$$

$$\alpha^{240}V_{52}+\alpha^{220}V_{53}+\alpha^{243}V_{54}+\alpha^{181}V_{55}+\alpha^{15}V_{56} \qquad \textbf{(5.8)}$$

The above equations have been tested by the calculation of syndromes for error-free codewords. The production of all-zero syndromes verifies the above equations. The C1 codeword of DDS-3 and the C2 codeword of DDS-1 share a common characteristic - the six parity bytes in the respective codewords occupy consecutive positions. This characteristic has produced similarities in the parity byte equations. The coefficients of the first thirteen data bytes in the parity byte equations for the C2 codeword of DDS-1 are the same as those of the last thirteen data bytes in the parity byte equations for the C1 code of DDS-3.

The C2 parity check matrix is [4]

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \ldots\ldots & 1 & 1 & 1 \\ \alpha^{31} & \alpha^{30} & \alpha^{29} & \alpha^{28} & \ldots\ldots & \alpha^{2} & \alpha & 1 \\ \alpha^{62} & \alpha^{60} & \alpha^{58} & \alpha^{56} & \ldots\ldots & \alpha^{4} & \alpha^{2} & 1 \\ \alpha^{93} & \alpha^{90} & \alpha^{87} & \alpha^{84} & \ldots\ldots & \alpha^{6} & \alpha^{3} & 1 \\ \alpha^{124} & \alpha^{120} & \alpha^{116} & \alpha^{112} & \ldots\ldots & \alpha^{8} & \alpha^{4} & 1 \\ \alpha^{155} & \alpha^{150} & \alpha^{145} & \alpha^{140} & \ldots\ldots & \alpha^{10} & \alpha^{5} & 1 \end{bmatrix} \quad \textbf{(5.9)}$$

and V^T is

$$[Q1\ \ Q2\ \ Q3\ \ V4\ \ V5\ \ V6\ \ \ldots\ \ V29\ \ Q4\ \ Q5\ \ Q6] \quad \textbf{(5.10)}$$

The C2 parity byte equations are:

$$Q_1 = \alpha^{200}V_4 + \alpha^{204}V_5 + \alpha^{147}V_6 + \alpha^{58}V_7 + \alpha^{126}V_8 + \alpha^{109}V_9 + \alpha^{217}V_{10} + \alpha^{80}V_{11} + \alpha^{129}V_{12} + \alpha^{23}V_{13} + \alpha^{180}V_{14} + \alpha^{80}V_{15} + \alpha^{76}V_{16} + \alpha^{217}V_{17} + \alpha^{230}V_{18} + \alpha^{29}V_{19} + \alpha^{44}V_{20} + \alpha^{59}V_{21} + \alpha^{69}V_{22} + \alpha^{76}V_{23} + \alpha^{83}V_{24} + \alpha^{205}V_{25} + \alpha^{210}V_{26} + \alpha^{45}V_{27} + \alpha^{154}V_{28} + \alpha^{180}V_{29} \quad \textbf{(5.11)}$$

$$Q_2 = \alpha^{250}V_4 + \alpha^{213}V_5 + \alpha^{62}V_6 + \alpha^{243}V_7 + \alpha^{179}V_8 + \alpha^{74}V_9 + \alpha^{14}V_{10} + \alpha^{113}V_{11} + \alpha^{230}V_{12} + \alpha^{37}V_{13} + \alpha^{29}V_{14} + \alpha^{82}V_{15} + \alpha^{17}V_{16} + \alpha^{206}V_{17} + \alpha^{30}V_{18} + \alpha^{78}V_{19} + \alpha^{28}V_{20} + \alpha^{141}V_{21} + \alpha^{169}V_{22} + \alpha^{178}V_{23} + \alpha^{176}V_{24} + \alpha^{140}V_{25} + \alpha^{89}V_{26} + \alpha^{119}V_{27} + \alpha^{192}V_{28} + \alpha^{146}V_{29} \quad \textbf{(5.12)}$$

$$Q_3 = \alpha^{213} V_4 + \alpha^{69} V_5 + \alpha^{132} V_6 + \alpha^{219} V_7 + \alpha^{170} V_8 + \alpha^{188} V_9 + \alpha^{40} V_{10} + \alpha^{226} V_{11} + \alpha^{69} V_{12} + \alpha^{199} V_{13} +$$
$$\alpha^{104} V_{14} + \alpha^{247} V_{15} + \alpha^{80} V_{16} + \alpha^{208} V_{17} + \alpha^{80} V_{18} + \alpha^{194} V_{19} + \alpha^{138} V_{20} + \alpha^{186} V_{21} + \alpha^{57} V_{22} +$$
$$\alpha^{84} V_{23} + \alpha^{84} V_{24} + \alpha^{39} V_{25} + \alpha^{85} V_{26} + \alpha^{59} V_{27} + \alpha^{72} V_{28} + \alpha^{245} V_{29} \quad \textbf{(5.13)}$$

$$Q_4 = \alpha^{120} V_4 + \alpha^{197} V_5 + \alpha^{179} V_6 + \alpha^{200} V_7 + \alpha^{149} V_8 + \alpha^{189} V_9 + \alpha^{184} V_{10} + \alpha^{152} V_{11} + \alpha^{21} V_{12} + \alpha^{223} V_{13}$$
$$\alpha^{19} V_{14} + \alpha^{155} V_{15} + \alpha^{23} V_{16} + \alpha^{145} V_{17} + \alpha^{52} V_{18} + \alpha^{159} V_{19} + \alpha^{249} V_{20} + \alpha^{114} V_{21} + \alpha^{11} V_{22} +$$
$$\alpha^{75} V_{23} + \alpha^{218} V_{24} + \alpha^{195} V_{25} + \alpha^{239} V_{26} + \alpha^{147} V_{27} + \alpha^{79} V_{28} + \alpha^{218} V_{29} \quad \textbf{(5.14)}$$

$$Q_5 = \alpha^{26} V_4 + \alpha^{67} V_5 + \alpha^{244} V_6 + \alpha^{209} V_7 + \alpha^{0} V_8 + \alpha^{31} V_9 + \alpha^{28} V_{10} + \alpha^{14} V_{11} + \alpha^{236} V_{12} + \alpha^{118} V_{13} +$$
$$\alpha^{163} V_{14} + \alpha^{110} V_{15} + \alpha^{26} V_{16} + \alpha^{87} V_{17} + \alpha^{147} V_{18} + \alpha^{89} V_{19} + \alpha^{92} V_{20} + \alpha^{25} V_{21} + \alpha^{158} V_{22} +$$
$$\alpha^{54} V_{23} + \alpha^{109} V_{24} + \alpha^{209} V_{25} + \alpha^{13} V_{26} + \alpha^{82} V_{27} + \alpha^{228} V_{28} + \alpha^{5} V_{29} \quad \textbf{(5.15)}$$

$$Q_6 = \alpha^{65} V_4 + \alpha^{34} V_5 + \alpha^{175} V_6 + \alpha^{80} V_7 + \alpha^{70} V_8 + \alpha^{198} V_9 + \alpha^{186} V_{10} + \alpha^{174} V_{11} + \alpha^{159} V_{12} + \alpha^{139} V_{13} +$$
$$\alpha^{119} V_{14} + \alpha^{60} V_{15} + \alpha^{42} V_{16} + \alpha^{151} V_{17} + \alpha^{150} V_{18} + \alpha^{245} V_{19} + \alpha^{83} V_{20} + \alpha^{184} V_{21} + \alpha^{130} V_{22} +$$
$$\alpha^{7} V_{23} + \alpha^{149} V_{24} + \alpha^{161} V_{25} + \alpha^{88} V_{26} + \alpha^{172} V_{27} + \alpha^{224} V_{28} + \alpha^{215} V_{29} \quad \textbf{(5.16)}$$

The C2 parity byte equations have also been verified by the production of all-zero syndromes for error-free codewords.

5.3 Uncorrected Burst Error Patterns

It is possible to obtain the patterns of erroneous bytes in decoded data in some simple cases without a full simulation of the error control procedures of DDS-3. It will be assumed that the optional C3 code is not used or else is ineffective due to the number of tracks affected. The interleave depth of the C2 code is three Fragments (recall from **Chapter One** that a Fragment is the DDS-3 equivalent of the Block of DDS-1) so that the six C2 parity bytes will allow correction of a burst of eighteen Fragments; this corresponds to a longitudinal corruption $^{4}/_{9}$ mm wide. If the minimum size of burst considered is twenty four Fragments then there can be no C2 error correction in any of the codewords affected (C1 is assumed to be overwhelmed and so passes on flags to C2). The interleaving of DDS-3 has been simulated computationally and the effects of burst errors on a Frame have been calculated by de-interleaving the erroneous Fragments. The results are shown below. In all cases there are 112 error-free runs of the length stated and 110 runs of erroneous bytes of the length stated plus two runs of erroneous bytes with a length two bytes less than that stated.

Burst size (Fragments)	Length of error-free runs (bytes)	Length of runs of erroneous bytes
24	108	48
30	96	60
36	84	72
42	72	84

There are 17468 data bytes in a DDS-3 Frame so the above figures show that in the case of a burst one third larger than the maximum correctable around 69% of the recorded data bytes are recovered error-free. This contrasts markedly with DDS-1 where a burst only $1/_{24}^{th}$ larger than the maximum correctable allowed error-free recovery of a little over half of the recorded data bytes in usable runs. This contrast must be due to the less complex interleaving employed in DDS-3 (for example, unlike DDS-1, there is no interleaving between tracks) so that uncorrected errors are not dispersed so widely.

5.4 Implications for the Present Robust Data Compression Strategy

The results of the last section demonstrate that a greater proportion of compressed data per Frame will be recovered in the case of DDS-3 than is the case with DDS-1 when the present strategy is employed with the same compression and synchronization parameters. It is possible, therefore, to maintain strong synchronization in the present compression strategy when using DDS-3 without changing the parameters of the strategy used with DDS-1.

Chapter Six

Conclusions

6.1 Review of the Thesis

Chapter One saw the introduction to the problem of error propagation in compressed data recorded on DDS tapes operating in hostile environments. The error control procedures of DDS were reviewed. **Chapter Two** described the Reed-Solomon error control coding of DDS and the decoding of the most complex of the three codes employed was detailed. The error correction capability of the simulation of DDS was found to correspond to that claimed for the format in the literature. Severe errors were input into the simulation and the uncorrected errors were calculated. In **Chapter Three** some of the better known data compression algorithms were described. The little known Rice algorithm was introduced as it was to find an important role in this work. **Chapter Four** described a compression strategy which limited the propagation of errors and prevented slippage in the decompressed data and which was independent of the compression algorithm. Compression algorithms were described which perform efficient piecewise compression of correlated physical data, mixed correlated physical and text data and English text. In each of these three cases the strategy was tested in the presence of both random and burst errors. Although unusually harsh, the random error tests demonstrated the ability of the strategy to establish strong resynchronization on a continuous basis. The effects of more realistic burst errors were investigated and the data recovery determined. **Chapter Five** paved the way for a simulation of the latest format DDS-3.

6.2 New Results

The use of conventional data compression in situations where the error control coding of the channel may let through uncorrected errors always risks the loss of all the data following such events. This thesis has demonstrated a viable alternative in which the effects of errors are confined to only a relatively small quantity of data following the error. Not only has error propagation been prevented but slippage in the decompressed data has also been eliminated. The DDS format is able to do this only on an entity by entity basis. The present strategy can prevent slippage *within* entities. This strategy has been rigorously tested for an actual channel and for realistic error patterns in the end-to-end simulation. The end-to-end simulation has brought together some quite distinct and diverse algorithms and methods. There are no reports of such simulations in the literature. In the course of the development of the computer model of the error control procedures of DDS the behaviour of the format in the presence of high random error rates was investigated. These results, again, are seemingly absent from the literature. The compressors employed within the strategy are also noteworthy. The compressor designed for correlated physical data is an implementation of the Rice algorithm - with synchronization. This implementation allows a completely free choice of block length. Hardware implementations tend to be more restrictive as far as the choice of block length is concerned. A value of sixteen bytes seems to be typical. The compressor designed for mixed correlated physical and text data is unusual as it switches between two quite distinct algorithms as the data type varies. The LZB text compressor has been shown to be able to perform surprisingly efficient compression of relatively small pieces of English text. LZB has been modified to perform even more efficiently, albeit very slightly.

In summary, this work has involved:

- complete simulation of an industry standard medium
- devising a scheme to resynchronize strongly i.e. at correct position in output data stream
- producing an overall strategy which limits error propagation at little cost in terms of compression performance
- devising a compression strategy which is both robust and effective for hybrid data and testing on a sample of real industrial data
- exploring the parameters of the strategy with regard to compression ratio and synchronization
- linking the simulations of the robust compression strategies directly to the DDS simulation for end-to-end simulations where the final output data is compared to the original.

6.3 Users and Data

Any compression strategy which seeks to limit error propagation will lose some data prior to resynchronization. Self-synchronizing Huffman codes produce an unpredictable number of erroneous symbols following a single bit error. This is a form of data loss. The adaptive compression of the present work led, almost axiomatically, to the piecewise approach. Errors affecting data compressed using this strategy may cause the loss of entire pieces, although slippage can be avoided.

Terrain elevation maps are one example of correlated physical data and so piece sizes might vary widely as compression performance *per se* is largely independent of piece size. Errors might arise in the initial measurement and recording in a hostile environment or the errors

might arise in the reading in a hostile environment. Lack of slippage in this data would be of paramount importance if measurements were made on a predetermined grid. If the data were recorded one dimensionally on a row by row basis on a rectangular grid then loss of part or all of a line of data without slippage could be countered through interpolation, since the data is correlated. However, if the resolution of the grid were designed to pick up even the sharpest possible change in elevation and if such changes fell within the line lost then these changes could not be approximated through interpolation.

Computer programs written in high level computer languages such as FORTRAN or PASCAL are clearly text. Absolutely no loss of text data in this application can be tolerated since the programs would not even compile properly let alone run correctly. On the other hand some loss of data can be tolerated in computer graphics data although here, again, prevention of slippage is vital. Graphics that did not consist of digitized photographs but instead were artificially generated images could be considered in the same way as text. If, for example, a road map represented a motorway with a black line on a white background then adjacent black and white pixels on a line would be a common phrase. The significance of the loss of data here is a judgement based upon surrounding patterns. A 'gap' appearing in a road through data loss should not be critical.

In some applications slippage might be regarded as unimportant. For example, the data might be correlated physical data recorded as x-y-z triples on an irregular grid or English text. It is possible to identify leaks and weaknesses in a pipeline by sending through mechanical devices to measure magnetically any variations in the thickness of the pipe wall. The output generated is the correlated physical data of the measurements and data identifying the position of the measurement. Avoidance of slippage *could* be vital in this application. Circumferential measurements are one dimensional but any weakness is likely to leave a two-dimensional

signature. The resolution of the measurement grid is, again, important and only consideration of that can determine if data loss is tolerable.

6.4 The Parameters of the Strategy

The design of a strategy needs to consider two parameters - the quantity of data to be compressed between synchronization sequences and the period of the cyclic count. The period of the count used in generating the results presented in this thesis has been fixed at 256 - the period of a count corresponds to the eight bits of a byte and might be regarded as 'standard'. The choices for these parameters will depend upon the environment in which the strategy is to operate. Operation in an environment generating frequent short bursts, say two or three corruptions within Frames per Basic Group, would be best served with a strategy with relatively small quantities of data between synchronizations and a 'standard' count period of 256. Compressed data would be recovered between deinterleaved errors and the count period would be adequate to maintain strong synchronization. Operation in an environment liable to cause loss of many consecutive Frames would suggest use of larger quantities of data between synchronizations since there is little scope for the recovery of data between deinterleaved errors. This would allow better overall compression performance since the overhead is lessened and, as far as text is concerned, compression performance is improved. Increasing the quantity of data between synchronizations increases the number of Frames spanned by the 'standard' cyclic count period.

6.5 Practicalities of Integrating the Strategy with DDS

There is no standard data compression algorithm for use with the DDS format so that the strategy presented here may be implemented in conjunction with DDS by anyone who may care to do so. Even if an algorithm such as DCLZ (Data Compression Lempel Ziv)[**43**], for example, were to become the industry standard the present strategy could still be employed if data compression were not made automatic. The practitioner could implement the present compression strategy in software rather than hardware as with DCLZ. The present strategy combined with the DDS format makes a versatile recording tool. A 60 metre DDS tape will store 1.3 Gbyte of user data. The compression of data to half its original size, which has been shown to be achievable by the present strategy for the data tested, allows the recording of 2.6 Gbyte of data on the tape. The immediate descendant of the DDS format, DDS-2, employs the same error correction and interleaving. The principal difference is the greater recording density of the newer format. A DDS-2 tape can store 4 Gbyte of uncompressed user data (this is clearly preferable to the possible 2.6 Gbyte of compressed data of DDS). It follows that the recording manifestation of a blemish of a particular size on the tape is of greater consequence, particularly as far as error propagation in compressed data is concerned, with format DDS-2. In an environment where blemishes of a particular size are liable to occur, the switch to DDS-2 should be accompanied by a change in the parameters of the compression strategy used with DDS. For example, the period of the cyclic count could be increased to ensure that strong synchronization is maintained. Compression of data to half its original size can result in the recording of 8 Gbyte of user data on a DDS-2 tape.

The present strategy for robust data compression is sufficiently developed to be usable now and in **Chapter Five** it was shown that the present strategy could also be implemented with DDS-3 without change. It would be preferable to implement the strategy in hardware for

reasons of speed and convenience but if required the developed software could be used to compress data prior to recording and decompress data when the tape is read.

6.6 Prevention of Slippage in Static Binary Prefix Codes

In the literature much of the work reported on the synchronization of compressed data has concentrated on Huffman codes. The occurrence of symbol slippage in self-synchronizing Huffman codes decoded following an error has been demonstrated in this thesis. An investigation into the prevention of such symbol slippage in binary prefix codes has recently been reported in the literature [**44**]. That work proposes the use of clear *extended synchronizing codewords* (ESC's) to detect as well as correct loss of code synchronization. As the term would suggest, construction of an ESC incurs an overhead penalty. The self-synchronization property of some Huffman codes cannot allow the occurrence of an error to be detected. The detection of the occurrence of the loss of code synchronization enables checks to be made on the state of symbol synchronization if the ESC is used as a marker and placed at regular positions in the bit stream. If an error occurs in the ESC or if errors cause the appearance of an ESC elsewhere in the bit stream then this paper suggests using a fixed length codeword as a counter of the occurrence of ESC's, inserted into the bit stream after the ESC.

6.7 Future Developments

The most obvious next step would be to assess the use of the developed strategy for the compression of data in real hostile environments. It is envisaged that the facilities so to do would only be readily available to interested parties within industry.

There are three separate error tolerant compressors, one for each of the data types investigated. Ideally, all three would be combined into a single 'intelligent compressor'. The error tolerant compressor designed for mixed data could be used for wholly correlated physical data as it stands. The difficulty lies in allowing English text to be tackled with a compressor which groups the data into blocks of perhaps as few as the sixteen or so values of the Rice algorithm. The look ahead buffer of a sliding window text compressor is typically around sixteen bytes in length. So, rather than using sliding window compression to compress a block in isolation it should be possible to scan ahead for the next block of text and append that to the current block so that the look ahead buffer is always around optimum size. If the data comprised only English text then that would present no difficulty.

There may be other modifications that can be made to improve compression performance. The extra complexity and run-time required are not often justified by the gains in performance when trying to improve upon already efficient algorithms such as LZB.

References

1. European Computer Manufacturers Association (ECMA)(June 1990). 3.81mm Wide Magnetic Tape Cartridge For Information Interchange - Helical Scan Recording - DDS Format. *Standard ECMA-139.*

2. Watkinson, John (1994). *The Art of Digital Audio (2nd edn).* Focal Press, Oxford.

3. European Computer Manufacturers Association (ECMA)(June 1995). 3.81mm Wide Magnetic Tape Cartridge For Information Interchange - Helical Scan Recording - DDS-2 Format Using 120m Length Tapes. *Standard ECMA-198, 2nd Edition.*

4. European Computer Manufacturers Association (ECMA)(March 18th 1996). Standard ECMA-XXX 3.81mm Wide Magnetic Tape Cartridge For Information Interchange - Helical Scan Recording - DDS-3 Format Using 125m Length Tapes. *Working Document TC17/96/13.*

5. Hewlett-Packard Company Press Release (1995). HP First To Market With 24GB DDS-3 DAT Drive. *http://www.hp.com:80/pressrel/oct95/23oct95j.html*

6. European Computer Manufacturers Association (ECMA)(June 1992). 3.81mm Wide Magnetic Tape Cartridge For Information Interchange - Helical Scan Recording - DDS-DC Format Using 60m And 90m Length Tapes. *Standard ECMA-150, 2nd Edition.*

7. Simpson, David (1992). DAT's all, folks. *Systems Integration, Vol.25, pp. 40-51.*

8. Sony Electronics Inc. (1996). Tape Streamer SDK-5000/M. *http://www.sel.sony.com/SEL/ccpg/storage/tape/t5000.html*

9. Hewlett-Packard Company (1995). HP 35480A DAT Drive. *http://www.dmo.hp.com/tape/35480a.htm*

10. Rice, R.F., Yeh, P.S. and Miller, W. (1991). Algorithms for a Very High Speed Noiseless Coding Module. *JPL Publication 91-1.*

11. Ferguson, Thomas J. and Rabinowitz, Joshua H. (1984). Self-Synchronizing Huffman Codes. *IEEE Transactions on Information Theory, Vol. IT-30, No. 4, pp. 687-693.*

12. Roberts, J.D. (1996). The Limiting Error Correction Capabilities of the CDROM. *PhD Thesis*. The University of Glamorgan.

13. Patel, Arvind M. (1986). On-the-fly decoder for multiple byte errors. *IBM J. Res. Develop., Vol. 30, No.3, pp. 259-269.*

14. Hayashi, K., Arai, T., Noguchi, T., Okamoto, H. and Kobayashi, M. (1986). Error Correction Method for R-Dat and its Evaluation. *ICASSP 86, Tokyo, pp. 9-12.*

15. Ko, C.C. and Tjhung, T.T. (1989). Simple Programmable Processor for Decoding Reed-Solomon Codes in Compact Disc Devices at High Speed. *International Journal of Electronics, 67(1), pp. 15-25.*

16. Hoffman, D.G., Leonard, D.A., Linder, C.C., Phelps, K.T., Rodger, C.A. and Wall, J.R. (1991). *Coding Theory - The Essentials (1st edn)*. Marcel Dekker, New York.

17. DDS Manufacturers Group (1988). DDS Format Description. A1-B6.

18. Van Gelder, T. (1989). Data Storage Using DAT. *Electronika, 37(18), pp. 21-31.*

19. Laeser, R.P., McLaughlin, W.I. and Wolff, D.M. (1986). Engineering Voyager 2's Encounter with Uranus. *Scientific American 255(5), pp. 34-43.*

20. Ziv, Jacob and Lempel, Abraham (1977). A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory, Vol. IT-23, No. 3, pp. 337-343.*

21. Bell, Timothy C., Cleary, John G. and Witten, Ian H. (1990). *Text Compression.* Prentice Hall, New Jersey.

22. Nelson, Mark (1991). *The Data Compression Book.* Prentice Hall and M&T Books.

23. Ziv, Jacob and Lempel, Abraham (1978). Compression of Individual Sequences via Variable-Rate Coding. *IEEE Transactions on Information Theory. Vol. IT-24, No. 5, pp. 530-536.*

24. Rice, R.F. and Plaunt, J.R. (1971). Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data. *IEEE Transactions on Communications Technology, COM-19(6), pp. 889-897.*

25. Rice, Robert F. (1972). Channel Coding/Decoding Alternatives for Compressed TV Data on Advanced Planetary Missions. *Proc. of the 5th Inter. Conf. on Sys. Sci., pp. 60-62.*

26. Rice, Robert F. (1974). Channel Coding and Data Compression System Considerations for Efficient Communication of Planetary Imaging Data. *JPL Technical Memorandum 33-695.*

27. Rice, Robert F. (1979). Practical universal noiseless coding. *SPIE Vol. 207 Applications of Digital Image Processing III, pp. 247-267.*

28. Rice, Robert F., Hibert E., Lee J.J. and Schlutsmeyer A. (1979). Block Adaptive Rate Controlled Image Data Compression. *Proceedings of 1979 National Telecommunications Conference, Washington, pp. 53.5.1-53.5.6.*

29. Yeh Pen-Shu, Rice Robert and Miller Warner (1991). On the Optimality of Code Options for a Universal Noiseless Coder. *JPL Publication 91-2.*

30. Rice, Robert F. (1979). Some Practical Universal Noiseless Coding Techniques. *JPL Publication 79-22.*

31. Rice, Robert F. (1982). End-to-End Imaging Information Rate Advantages of Various Alternative Communication Systems. *JPL Publication 82-61.*

32. Rice, Robert F. (1991). Some Practical Universal Noiseless Coding Techniques, PART III, Module PSI14, K+. *JPL Publication 91-3.*

33. Venbrux Jack, Yeh Pen-Shu and Liu Muye N. (1992). A VLSI Chip Set for High-Speed Lossless Data Compression. *IEEE Transactions on Circuits and Systems for Video Technology, Vol. 2. No. 4, pp. 381-391.*

34. Lelewer, Debra A. and Hirschberg, Daniel S. (1987). Data Compression. *ACM Computing Surveys, Vol. 19, No. 3, pp. 261-296.*

35. Escott, Adrian E. and Perkins, Stephanie (1995). Constructing good binary synchronous Huffman codes. *Proceedings 1995 International Symposium on Synchronization, Essen, Germany, pp.105-110 .*

36. Teuhola J. and Raita T. (1991). Piecewise Arithmetic Coding. *Proceedings of the First International Data Compression Conference (IEEE DCC '91), pp. 33-42.*

37. Kobler, Ben (1991). Techniques for Containing Error Propagation in Compression/decompression Schemes. *NASA Space and Earth Science Data Compression Workshop, NASA Conference Publication 3130, pp. 73-74.*

38. Woolley S.I. (1994). Error Statistics and Data Compression in Digital Instrumentation Recording Systems. *PhD Thesis.* University of Manchester.

39. Rice, Robert F. and Lee, Jun-Ji (1983). Some Practical Universal Noiseless Coding Techniques, Part II. *JPL Publication 83-17.*

40. Lei, Shaw-Min (1991). The construction of efficient variable-length codes with clear synchronizing codewords for digital video applications. *SPIE Vol. 1605 Visual Communications and Image Processing '91: Visual Communication, pp. 863-873.*

41. Calgary Data Compression Corpus.
ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/

42. Fenwick, Peter M. (1993). Ziv-Lempel Encoding with Multi-bit Flags. *Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, pp. 138-147.*

43. European Computer Manufacturers Association (ECMA)(June 1991). Data Compession for Information Interchange - Adaptive Coding with Embedded Dictionary - DCLZ Algorithm. *Standard ECMA-151.*

44. Lam, Wai-Man and Kulkarni, Sanjeev R. (1996). Extended Synchronizing Codewords for Binary Prefix Codes. *IEEE Transactions on Information Theory, Vol. 42, No. 3, pp. 984-987.*

Copy of Publication

'Synchronized Lossless Data Compression for DDS Tape Storage'.

Proceedings 1995 International Symposium on Synchronization, December 14-15, 1995, Essen, Germany, pp. 12-15. ISBN 90-74249-07-8

Synchronized Lossless Data Compression for DDS Tape Storage

O.D.J. Thomas, D.H. Smith, A. Ryley

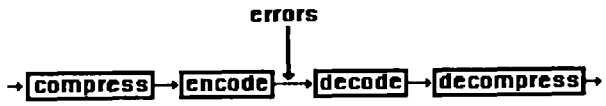
Division of Mathematics and Computing, University of Glamorgan,
Pontypridd, Mid Glamorgan, CF37 1DL, Wales, United Kingdom.
Tel. 0044 1443 482250 Fax. 0044 1443 482711
email <ODJTHOMA>@glam.ac.uk

Abstract

A data compression strategy with frequent resynchronization which limits the propagation of errors is presented. The strategy is applied to RDAT-DDS digital tape storage devices. The decompressed data is compared to the original when errors in the channel are uncorrected.

1 Introduction

Proprietary storage devices find application in remote data acquisition, often in hostile environments. Residual channel errors can have catastrophic consequences if data compression is used. The challenge is to accept the limitations of the channel in hostile environments and develop a data compression strategy that checks the propagation of errors. The situation under investigation is summarized below:



The development of a deterministic computer model of the error control procedures of RDAT-DDS digital tape storage (DDS) enables the uncorrected error patterns to be calculated for given input errors that reflect realistically the effects of the environment. This provides a truer test of the compression strategy than idealized bit errors. This paper describes the integration of a robust data compression strategy incorporating frequent resynchronization with a model of the error control procedures of an actual channel.
The encoding procedures of DDS are prescribed in an international standard [1]. The decoding procedures are not so prescribed but are documented in the literature [2,3,4].

2 The Rice Algorithm

The propagation of errors in decompressed data can be prevented by performing piecewise compression. An adaptive scheme is needed since information such as a probability table would require significant overhead for its protection and is in any case a significant overhead itself for the piece sizes contemplated. Piecewise compression reduces the performance of some of the well known adaptive algorithms. The solution is to use an algorithm that is designed to compress effectively very small blocks of data and the Rice algorithm [5,6,7] does exactly that. This algorithm performs data compression by reducing the statistical redundancy of the data and works adaptively by selecting the optimal of several Huffman-equivalent codes on blocks of, for example, sixteen values. The algorithm works in two stages. The first of these is termed preprocessing and is designed to remove the correlation between the analogue samples by calculating the differences between adjacent values.

The second stage compresses the processed data in blocks choosing for each block the optimum code. The bit stream for the compressed data takes the form

ref id code id code...

where *ref* is the reference starting values (to enable the original data to be reconstructed from the coded differences), *id* is the identifier of the code option used and *code* is the complete code for the data block.

3 Synchronization of the Decompressor and the Output Data Stream

The Rice Algorithm produces a variable length code. As with other variable length codes the process of locating a valid coding after the occurrence of errors can be difficult. This process, here, shall be termed *weak synchronization. Strong synchronization* shall be the term used to describe the occurrence of weak synchronization *without* slippage in the output data stream. Each form of synchronization may be illustrated with the use of a self-synchronizing Huffman code. The code is:

a	1
b	01
c	001
d	000

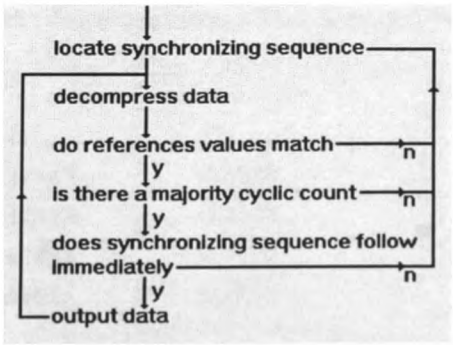
The sequence a.d.a.c.a.b, using the above, is coded as 1.000.1.001.1.01. If the first 1 becomes a 0 then decoding gives 000.01.001.1.01 or d.b.c.a.b. Weak synchronization is said to have occurred since there is slippage in the decoded symbols. If the first 0 becomes a 1 then decoding gives 1.1.001.001.1.01 or a.a.c.c.a.b. Strong synchronization is said to have occurred since the number of decoded symbols matches the number of encoded symbols.

The piecewise compression strategy requires that synchronization be forced rather than awaited as with self-synchronizing Huffman codes. This is achieved by the addition to the compressed data stream of redundancy in the form of a synchronizing sequence. The term sequence is used quite deliberately since there is no attempt to reserve a code for synchronization. The synchronizing sequences alone will only provide weak synchronization. The establishment of strong synchronization requires the addition of more redundancy. A cyclic count is introduced to number the groups of data between synchronizing sequences. If an error occurs and data is lost then the count allows the data that is decompressed subsequently to be placed correctly into the output data stream by shifting along by the number of values lost.

The complete compressed data stream used in producing the results presented in this paper is

... sync ref id code....id code ref count count count count sync...

The synchronization scheme is represented as:



The synchronization sequence used is relatively short at only eight bits in length. A longer sequence would add unnecessary redundancy but a shorter sequence would increase the number of false synchronizations after an error, which would slow decompression; there is no hard-and-fast rule governing the length of the sequence. The reference value is repeated since it is of fixed length and synchronization could be established if it were affected by errors. The cyclic count is of a fixed length of eight bits and so requires repetition also. The Reed-Solomon symbols of the error correction codes are of eight bits so that a single symbol error might affect up to two counts. The count, therefore, is written four times to make the reading of a valid count much more likely.
It is clear from the structure of the compressed data stream that the priority of the decompression scheme is to maintain the integrity of the cyclic count and, hence, strong synchronization. Data that is decompressed will be discarded if there is no valid count.

4 Results

The test data used in the production of the results presented here was part of a database of terrain modelling data and consists of eight-bit elevations on a rectangular grid.

4.1 Random Errors

The DDS format is able to correct the bit errors that occur in practice with extremely high reliability. Abnormally high bit error rates were input into the simulation to test the performance of the compression strategy. The graphs in the literature do not cover random bit error rates beyond around 0.003. The results shown below were produced using a Rice block length of 12 and there are 10 blocks between synchronizations. The compression ratio was 0.5178.

bit random error rate	corrected symbol error rate	% values recovered correctly
0.01	0.023	38.18
0.009	0.013	58.27
0.008	0.0059	77.65
0.007	0.0025	90.19
0.006	0.0013	95.28
0.005	0.00037	98.60
0.004	0.000015	99.94

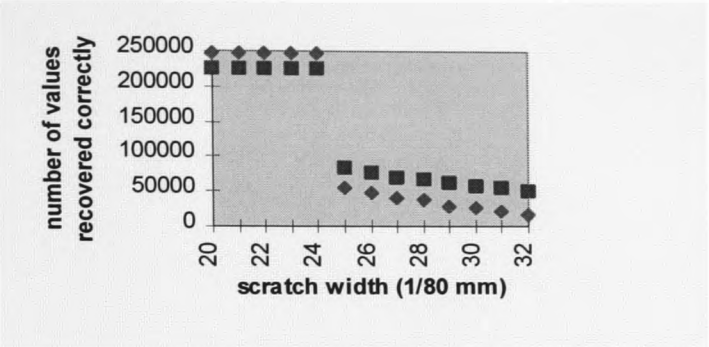
4.2 Burst Errors

The results for burst errors were produced using a more usual block length of sixteen values and the number of blocks between synchronizations (Nb) considered were 4,8,16 and 24. DDS structures the data into Basic Groups of 126632 bytes. The number of values compressed into a Basic Group (Nv) and the compression ratio (Cr) are shown below:

Nb	Nv	Cr
4	227110	0.5576
8	247422	0.5118
16	261883	0.4835
24	266035	0.4769

The error control procedures of DDS are able to correct a longitudinal scratch up to 0.3mm wide [8]. Scratch

widths are increased in increments of 1/80 mm in the simulation as this is the minimum that can be simulated. The strategy was ineffective in dealing with uncorrectable scratches when Nb was 16 or 24. The numbers of values recovered correctly when Nb is 4 and 8 are shown below:



The coding between synchronizations is small enough to appear between deinterleaved errors when Nb is 4 or 8 and so allows a significant proportion of values to be recovered correctly.
The performance of the synchronized lossless compression described here must be placed in context. The use of a conventional compression strategy in a hostile environment might result in the loss of almost all the data through error propagation. The scheme presented here can ensure that only the compressed data that is directly affected by errors is lost.

Bibliography

1. European Computer Manufacturers Association (ECMA) (1992). 3.81mm Wide Magnetic Tape Cartridge For Information Interchange - Helical Scan Recording - DDS-DC Format Using 60m And 90m Length Tapes.

2. Ko, C.C. and Tjhung, T.T. (1989). Simple Programmable Processor for Decoding Reed-Solomon Codes in Compact Disc Devices at High Speed. *International Journal of Electronics, 67(1), pp. 15-25.*

3. Hoffman, D.G., Leonard, D.A., Linder, C.C., Phelps, K.T., Rodger, C.A. and Wall, J.R. (1991). *Coding Theory - The Essentials (1st edn).* Marcel Dekker, New York.

4. Watkinson, John (1994). *The Art of Digital Audio (2nd edn).* Focal Press, Oxford.

5. Rice, R.F. and Plaunt, J.R. (1971). Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data. *IEEE Transactions on Communication Technology, COM-19(6), pp. 889-897.*

6. Rice, R.F., Yeh, P.S. and Miller, W. (1991). Algorithms for a Very High Speed Noiseless Coding Module. *JPL Publication 91-1.*

7. Venbrux Jack, Yeh Pen-Shu and Liu Muye N. (1992). A VLSI Chip Set for High-Speed Lossless Data Compression. *IEEE Transactions on Circuits and Systems for Video Technology, Vol. 2, No.4, pp. 381-391.*

8. Van Gelder, T. (1989). Data Storage Using DAT. *Electronika, 37(18), pp. 21-31.*