

LKC
QA
268
.M67
1983

C.2

IC

21
MORE POWERFUL ERROR-CORRECTION

SCHEME FOR THE BROADCAST

TELIDON SYSTEM

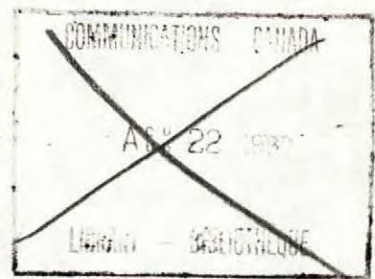
3 final reports

1/ Brian Mortimer

Michael Moore

Industry Canada
Library Queen
AVR 28 1998
APR 28 1998
Industrie Canada
Bibliothèque Queen

Department of Mathematics and Statistics



Carleton University
Ottawa, Canada

MORE POWERFUL ERROR-CORRECTION

SCHEME FOR THE BROADCAST

TELIDON SYSTEM

Brian Mortimer

Michael Moore

FINAL REPORT

prepared for the Department of
Communications, Ottawa under
DSS Contract No. OSU82-00164

Principal Investigator: Dr. B.C. Mortimer, Carleton University

Scientific Authority: Dr. M. Sablatash, Communications
Research Centre, Dept. of Communications

March, 1983

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	1
STATEMENT OF PRINCIPAL RESULTS	2
CHAPTER 1 INTRODUCTION	4
1.1 Error Correcting Codes in Broadcast Teletext Systems	4
1.2 Performance of the Coding System with Independent Errors	8
1.3 Options and Recommendations	13
CHAPTER 2 THE CODES	21
2.1 The Hamming and Product Codes	21
2.2 The Code C	23
2.3 Encoding	26
2.4 Decoding Code C (in theory)	27
2.5 Bundle Codes	28
2.6 Encoding and Decoding Bundle Codes	30
CHAPTER 3 PERFORMANCE OF THE BUNDLE CODE SYSTEM	32
3.1 System Performance with Independent Errors	32
3.2 The Probability of Correct Decoding for a t-Error Correcting Code	34
3.3 The Vertical and Horizontal Codewords and Pernicious Error Patterns	35
3.4 Putting the Pieces Together	50
3.5 Bundle Length Effects	52
3.6 Burst Errors	53

CHAPTER 4	OTHER DECODING STRATEGIES	56
4.1	Introduction	56
4.2	No Bundle Coding	56
4.3	Erasur e Correction of One Data Block	58
4.4	Single-Byte-Correction of the Vertical Codewords	70
4.5	Full Decoding - the other cases	62
CHAPTER 5	TWO-BYTE EXTENSIONS OF THE PRODUCT CODE	63
5.1	Extensions in General	63
5.2	Some Extension Ideas that Don't Work	64
APPENDIX A	Special Weight 6 Codewords	67
APPENDIX B	Extensions of the Product Code: Weight Six Codewords	70
APPENDIX C	Program Listings	
REFERENCES		73

ACKNOWLEDGEMENTS

We are happy to express our thanks for the assistance of a number of students we have helped out on various parts of the project: James Currie, Paula Gray, Lee Oattes and Brian Leroux. We are also appreciative of the quality and efficiency of the work of our typist Susan Jameson.

Brian Mortimer
NSERC University Research Fellow,
Department of Mathematics and Statistics,
Carleton University, Ottawa, K1S 5B6.

Michael Moore
Assistant Professor,
Department of Mathematics and Statistics,
Carleton University, Ottawa, K1S 5B6.

More Powerful Error-Correction Schemes for the Broadcast Telidon
System

DSS Contract No. OSU82-00164

Statement of Principal Results

1. A particular Reed-Solomon code based on symbols of 8 bits was defined, analyzed and proposed as a "two-byte" data block code for the broadcast Telidon System. This is a code of variable length which uses two check symbols per codeword.
2. A proposal was made to encode collections (tentatively called bundles) of data packets by interleaving a single-byte correcting code across the data blocks. The Reed-Solomon code of Item 1 above was proposed as a suitable code to implement this bundle code.
3. It was shown that in the case of independent errors and a proper decoding procedure there is a significant improvement in using bundle coding over using single or double bit error correction on the data blocks alone. It was further shown that bundle coding without any error correcting on the data blocks gives better performance than double bit error correction of the data blocks without bundle coding.

4. It was shown that the Reed-Solomon code of Item 1 could be efficiently decoded by a microprocessor based decoder.

Reports Delivered Under the Terms of the Contract

"Two-Byte Data Block and Bundle Codes for the Broadcast Telidon System", Progress Report, November, 1982.

"More Powerful Error-Correction Schemes for the Broadcast Telidon System", Final Report, March, 1983.

March 31, 1983

Brian Mortimer, Ph.D.

NSERC University Research Fellow

Carleton University

Principal Investigator

INTRODUCTION

1.1 Error Correcting Codes in Broadcast Teletext System

Broadcast teletext systems are designed to be used in a wide variety of situations. They must meet the needs of a spectrum of users and tolerate a range of communication channels. Error correcting codes can be used effectively in broadcast teletext systems to enhance the performance observed by the end user. This report describes certain aspects of this application of error-correcting codes.

The model on which the work described in this report is based is the Canadian Broadcast Telidon System as outlined in the Broadcast Specification No. 14 (Provisional, June 1982)^[1]. The data stream in this system is broken up into data lines. Each data line begins with some synchronization information. This is followed by the 33 bytes of a data packet; a five byte prefix is followed by a 28-byte data block.

The prefix bytes consist of three address bytes which identify the source of the data packet, then a single continuity count byte which counts, in cycles of 16, the data packets of a given address as in the order in which they were sent down the channel and finally

a Packet Structure byte which contains information about the format of the data block. All of these bytes are encoded as Hamming [8,4]-codewords (with odd parity). This means that single errors are corrected and all double errors are detected if they corrupt one of these bytes.

The data block may contain information about the organization of the data in the channel or it may consist of the actual data (PDI's) that the system is designed to transmit from source to user. In the former case the bytes are encoded as Hamming [8,4] codewords as in the prefix. We will ignore these data blocks on the hypothesis that they represent a very minor fraction of all data blocks transmitted. If the data block contains data then a certain number, say s , of its 28 bytes are committed to providing the redundant information that allows an error correcting code to correct errors in the data block. The value of s is signalled by two of the bits in the Packet Structure Byte of the prefix. Thus s is limited to 4 values and we will assume that three of them are $s = 0, 1$ or 2 .

The data bytes all have odd parity so the value $s = 0$ represents an error detecting code. This code detects all error patterns which have an odd number of errors in at least one byte of the data block. No correction is possible so the receiver must await retransmission when an error is detected.

If $s > 0$ then an error correcting code is being used to repair damage caused by noise in the channel. The choice of error correcting code has been the subject of much work and discussion [2]-[8], and we will make further proposals in this report. The codes used in the data blocks are called data block (and later horizontal) codes.

We are proposing in this report that a third level of coding (after prefix and data block) be available to improve the performance of the system as a whole. The idea is to gather together bundles of data blocks (from the same teletext page) and encode them together. The new codewords are vertical. This means that these codewords consist of two bytes from each data block of the bundle. For example one codeword would consist of the first and fourteenth bytes of each data block. The last data block of the bundle consists entirely of check bytes for the vertical codewords, see Figure 1.1. In order for the vertical code to be useful it must use at least two check bytes in each codeword. This is why we have specified two bytes from each data block. (If we increase the redundancy by using two data blocks of vertical checks then we can correct more errors and in particular we can get away with having only one byte from each data block in each vertical codeword.)

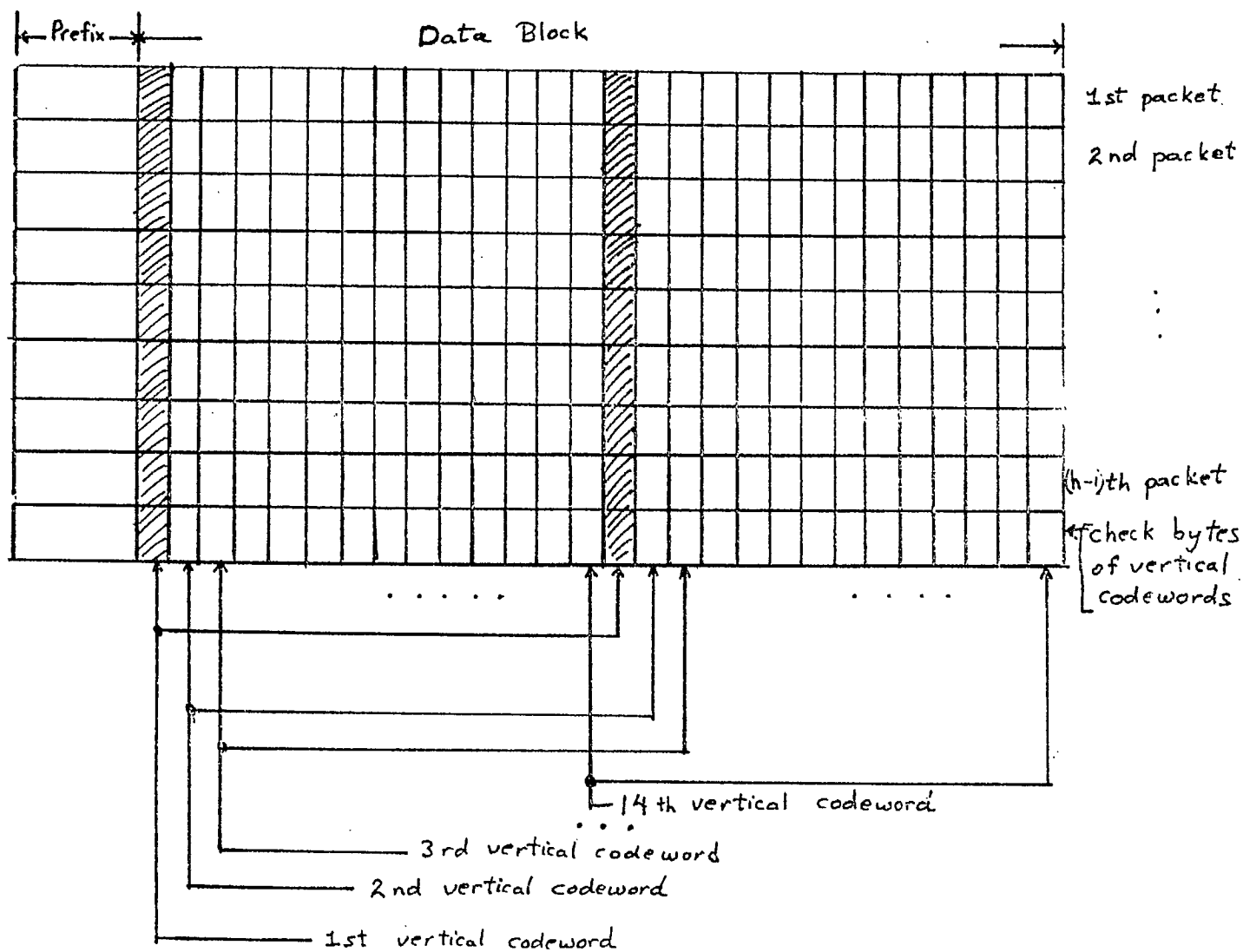


Figure 1.1 Arrangement of the 14 Vertical Codewords of the Bundle Code

1.2 Performance of the Coding System with Independent Errors

When we refer to "the coding system", we are referring to all of the levels of coding acting together. We will use as measures of performance the expected number of bundles before an incorrectly decoded bundle occurs and the expected number of information bits before such an event. The point is that longer bundles contain a larger fraction of information bits but do not protect it as well.

The coding of a bundle comes in 3 layers. The first is the prefix code which consists of 5 bytes each encoded with the Hamming (8,4) code. Due to the continuity count byte (the 4th byte) and the interpretation of the packet structure byte (the 5th byte) it will be a rare event that a decoding error in the prefix actually goes unnoticed by the system. The most common error effect is that a packet is lost or appears to be lost from the broadcast stream. This might also occur if there are synchronization problems when the line arrives.

The second layer is the data block code. We consider three options:

Byte-Parity (Parity)

The only coding is the parity bit in each data byte.

Single Bit Error Correction (SEC)

The 28th byte of the block is a check byte which allows the correction of a single bit error. With only one check byte it is not possible to correct all double errors.

Double Bit Error Correction (DEC)

The 27th and 28th bytes of the block are check bytes. Any double bit error can be corrected.

The third layer of coding covers all of the data blocks together (but excludes the prefix bytes). We consider four options. The first is to add no further coding. The other three are simply alternative decoding strategies for the same encoding. For these we interleave a number of vertical byte-correcting codes across the set of h data blocks adding 28 check bytes as the h -th data block (the layout of the interleaving is given in Figure 1.1). We assume that the basic code which is then interleaved is capable of correcting any error pattern confined to a single byte or any two byte-errors when the bytes are known (erasure decoding). We will propose such a code below (Chapter 2). We summarize the four strategies for coding the bundle as follows.

Nocode:

The $(h-1)$ data blocks are sent as they are with no further protection.

Erasure correction:

The bundle code is used to replace a lost data block by erasure correction. One data block can be retrieved and no other corrections are attempted on the bundle. This technique can overcome a failure in a single prefix or a single synchronization error.

14 X 1-Byte-error correction:

A single byte error in each of the 14 interleaved codewords is corrected. If there are two faulty bytes in any one codeword then the errors are not correctable.

Full Decoding:

In each of the interleaved codewords we correct either a single erroneous byte or two bytes with parity failures or two bytes which are absent entirely. In this scheme a single missing data block is replaced if there are no other errors. If all the data blocks are present we correct either one byte error or two byte erasures in each of the 14 vertical codewords.

In order to calculate the mean number of pages before a decoding fault (not a correct decoding) we must first choose a coding strategy for the second and third levels and then make an assumption about the patterns and frequency of the errors. As to the latter we will assume at this stage that the errors arise independently and defer

the question of burst errors to later consideration. We are then able to present (Figure 1.1) for each combination of coding strategies and a range of bit error rates the expected number of bundles until an incorrect decoding, for a bundle length $h \geq 9$.

The data of Figure 1.1 show clearly that performance can be greatly improved if a bundle code is used, at least, to correct erased data blocks and then to correct more if it can. This is especially true if a double bit error correcting code is used on the packets first. The importance of the double bit error corrections is made clear by calculating the expected number of bit errors in the bundle. We have included these numbers in Figure 1.1 as well. At a bit error rate of 10^{-3} we expect slightly more than 2 errors. The probability that two bit errors are in the same block is just $1/9$. Thus a single bit error correcting code has a lot of trouble. Also without erasure decoding we cannot escape the failures from the prefix so these act as a limiting factor. We can easily calculate for example that the expected number of bundles until at least one prefix fails is 800 at a BER of 10^{-3} but we expect $1.4E6$ bundles before at least two prefixes fail at the same BER.

We now take a look at Full Decoding in more detail. We have two parameters that we can vary: the bit error rate and the bundle

TABLE 1.1 Expected Number of Bundles Of Length 9 Until Decoding Fault

Coding Strategy		Bit Error Rate						
Bundle	Data Block	10^{-3}	8×10^{-4}	6×10^{-4}	4×10^{-4}	2×10^{-4}	10^{-4}	10^{-5}
No code	Parity	1.2	1.3	1.5	2.0	3.3	6.1	56
	SEC	6.2	9.2	16	33	129	505	4.9E4
	DEC	74	137	304	940	6280	3.9E4	1.6E6
Erasure cor- rection of 1 data block	Parity	1.2	1.3	1.4	1.8	3.0	5.9	50
	SEC	65	145	421	1969	2.9E4	4.5E5	4.4E9
	DEC	9615	3.3E4	1.6E5	1.6E6	7.0E7	2.7E9	1.3E14
Single byte correction in 14 v.c.	Parity	8.4	13	22	47	185	730	7.2E4
	SEC	71	127	252	621	2630	1.0E4	1.0E6
	DEC	352	615	1241	3247	1.6E4	7.0E4	7.9E6
1-byte error	Parity	140	263	596	1920	1.5E4	1.1E5	2.8E7
2-byte erase.	SEC	*	*	*	*	*	*	*
in 14 v.c.	DEC	1.3E4	2.8E4	7.4E4	2.7E5	2.3E6	1.9E7	1.7E10
Expected number of bit errors in the data blocks		2.02	1.61	1.21	0.81	0.40	0.20	0.02

- Notes: 1) A Bundle contains eight data blocks and except for the case of 'No code' above there is a ninth data block consisting of check bytes. Thus the bundle length is $h=9$.
- 2) The abbreviation v.c. above stands for vertical code word.
- 3) The entries marked * have not been calculated at time of printing.

length. The expected number of bundles and of information bits before an uncorrectable bundle for various choices of bit error rate and bundle lengths are presented in Tables 1.2 and 1.3.

1.3 Options and Recommendations

As far as the error correction scheme is concerned there are two sets of options which have to be considered - the encoding options and the decoding options. The encoding options are the various possible ways in which the transmitting agency can encode the data for error protection. The decoding options are the range of procedures which teletext decoder manufacturers can build into their receivers to make use of the redundancy for error correction. The performance of the system depends on the encoding scheme, the decoding procedure and the patterns of the errors in the channel (e.g. independent, short dense bursts, etc.).

The encoding options all use the Hamming code on the prefix byte. There is then a choice of data block code and a choice of bundle code format. The data block code might be simply the byte parity checks. It might use one byte of redundancy to obtain a single bit error correcting code or a two-byte code to obtain double bit error correction. We recommend Product code for the one-byte code and code C for the two byte code. X

TABLE 1.2 Expected number of information bits until
incorrect decoding

BIT ERROR RATE:	.002	.001	.0008	.0006	.0004

BUNDLE LENGTH					
5	1.66E6	2.21E7	4.66E7	1.17E8	4.26E8
6	1.57E6	2.19E7	4.70E7	1.19E8	4.33E8
7	1.50E6	2.16E7	4.68E7	1.21E8	4.38E8
8	1.43E6	2.11E7	4.65E7	1.21E8	4.40E8
9	1.38E6	2.07E7	4.58E7	1.20E8	4.41E8
10	1.34E6	2.02E7	4.52E7	1.20E8	4.41E8
11	1.31E6	1.97E7	4.44E7	1.18E8	4.40E8
12	1.30E6	1.92E7	4.37E7	1.17E8	4.39E8
13	1.29E6	1.87E7	4.29E7	1.16E8	4.38E8
14	1.29E6	1.82E7	4.21E7	1.15E8	4.36E8
15	1.29E6	1.78E7	4.13E7	1.14E8	4.34E8

TABLE 1.3 Expected number of 504 byte pages until an
incorrect decoding

BIT ERROR RATE:	.002	.001	.0008	.0006	.0004

BUNDLE LENGTH					
5	4.13E2	5.48E3	1.15E4	2.92E4	1.05E5
6	3.19E2	5.44E3	1.16E4	2.95E4	1.07E5
7	3.72E2	5.37E3	1.16E4	3.02E4	1.08E5
8	3.56E2	5.25E3	1.15E4	3.00E4	1.09E5
9	3.43E2	5.13E3	1.13E4	2.98E4	1.09E5
10	3.33E2	5.01E3	1.12E4	2.97E4	1.09E5
11	3.27E2	4.88E3	1.10E4	2.94E4	1.09E5
12	3.22E2	4.76E3	1.08E4	2.91E4	1.09E5
13	3.20E2	4.64E3	1.06E4	2.89E4	1.08E5
14	3.20E2	4.53E3	1.04E4	2.85E4	1.08E5
15	3.21E2	4.42E3	1.02E4	2.83E4	1.07E5

TABLE 1.4 Expected number of bundles of length h until
incorrect decoding

BIT ERROR RATE:	.002	.001	.0008	.0006	.0004

BUNDLE LENGTH					
5	2.01E3	2.67E4	5.63E4	1.42E5	5.14E5
6	1.53E3	2.13E4	4.56E4	1.15E5	4.21E5
7	1.21E3	1.75E4	3.80E4	9.88E5	3.55E5
8	1.00E3	1.47E4	3.24E4	8.45E5	3.07E5
9	8.46E2	1.26E4	2.80E4	7.35E5	2.69E5
10	7.32E2	1.09E4	2.46E4	6.53E5	2.40E5
11	6.46E2	9.66E3	2.17E4	5.81E5	2.16E5
12	5.80E2	8.57E3	1.95E4	5.24E5	1.96E5
13	5.28E2	7.66E3	1.75E4	4.77E5	1.79E5
14	4.88E2	6.90E3	1.59E4	4.35E5	1.65E5
15	4.54E2	6.26E3	1.45E4	4.01E4	1.52E5

We then come to the bundle coding options. These are somewhat more involved. We may decide on not using bundle coding and that is the first option. We have described earlier in this Section an option using 14 vertical codewords and one data block of check bytes to form a bundle code of length h data blocks where h can range from 2 to 63. This second C bundle code is susceptible to a variety of decoding options as we have already described. It is clarified in Chapter 2 that by using only 13 instead of 14 vertical codewords this system can be made completely compatible with the two byte data block code. We have shown that, if the Full Decoding method is used, then this sort of encoding option - code C on the data blocks and code C in the vertical codewords - will give a great improvement in the expected interval between incorrect decodings when the errors are independent. We have also indicated that the bundle length is not especially important in this context but that long bursts may cause trouble particularly if the burst runs from one data packet into the next from the same page.

What could a decoder designed to deal with Product code do with this bundle code? It can still perform single error corrections on the data blocks since code C extends Product code. But now, although it can decode the vertical codewords as if they were from Product code, it can't perform the erasure corrections of two bytes on which

the power of the bundle code depends. One option is to allow a bundle code which has 38 vertical codewords, each from Product code. This bundle code will perform as the SEC-data block / Erasure - only decoding option of Table 1.1 (or actually a bit better). It can correct one erased data block and uses one data block of checks. We have not analyzed the performance in great detail.

We can infact put all of this together into a new bundle encoding option. The bundle in this new case will use one or two data blocks of check bytes and always has $28-s$ vertical codewords where s is the number of check bytes in the data block code. The vertical codewords cover only the data bytes and the check blocks are encoded with the data block code. The vertical code is just whichever code is used on the data blocks be it Product or C . This would allow either a Product decoder or a C -code decoder to decode the bundle. The Product decoder acting on the vertical codewords could replace an erased block and perhaps correct a few random bit errors not corrected by the horizontal codewords. The real benefit comes when code C is used on the 26 columns of bytes and two check bytes are added, one in each of the two check blocks. This would allow one error correction or two erasure corrections (of bytes) in each column. This permits two missing data packets to be replaced. More importantly we have noted

at the end of Chapter 3 that a burst of errors which runs over from the end of one data block into the prefix of the next data packet cannot be corrected by the bundle arrangement studied throughout the rest of this report. Such a burst would however, be correctable by this new bundle scheme.

The price which we are (apparently) paying for this added power is increased redundancy which some may find a hard pill to swallow. But we have already shown that the bundle length does not affect the expected number of information bits between incorrect decodings very significantly when the errors are independent. So if we just double the bundle length and use two check blocks the information rate is the same but a potentially dangerous class of burst error patterns is now correctable.

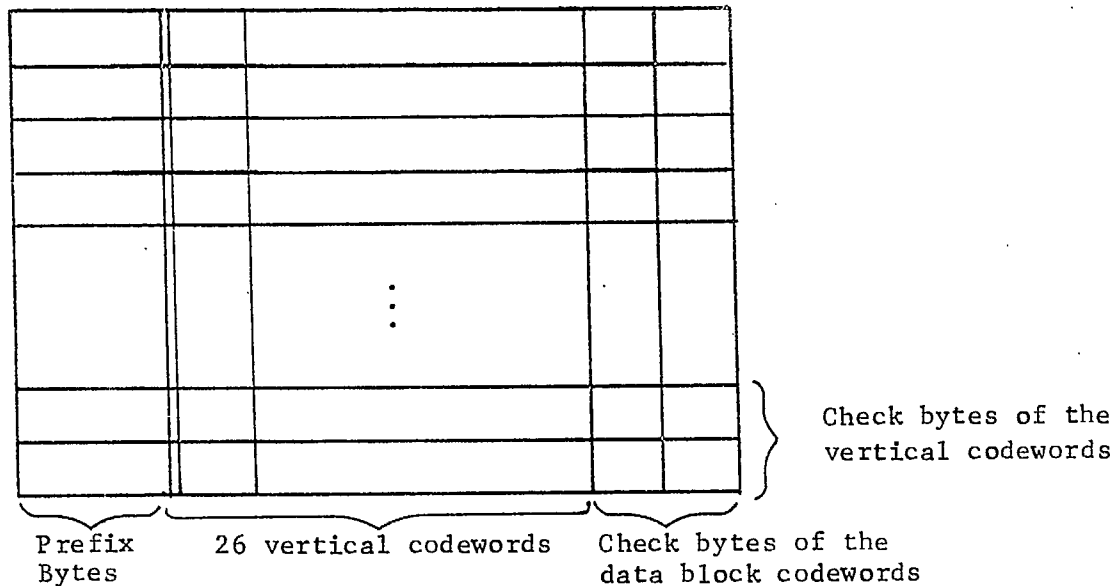


Figure 1.2 A New Bundle Scheme

This covers the encoding options both the ones analyzed to date and the new bundle coding scheme which is as yet unanalyzed. Several decoding options have been mentioned earlier and they are quite numerous. For example code C can be decoded either bitwise or byte-wise (which is quicker). We have discussed in Section 1.3 several decoding options for the bundle code and showed that "Full Decoding" is the most effective of these when errors are independent. Actually after the bulk of the current study was complete it occurred to us that even more performance can be squeezed out of the bundle code. We could count the decoding failures in the horizontal codewords. If there is one decoding failure and $(h-1)$ correct decodings then we replace the one data block with the decoding failure by erasure decoding. We have not analyzed this decoding strategy but note in passing that most of the error patterns declared "uncorrectable" in Chapter 3 would actually be corrected by this procedure.

The problem of estimating how well a given encoding scheme and matched decoding procedure will work requires for its solution a description of the patterns of errors which occur in the channel. We have assumed in most of this report that the errors are independent but we do not have any reason to suppose that this captures the essence of most or even many broadcast teletext channels. Until field data is available there is not much we can do about this problem.

CHAPTER 2

THE CODES

2.1 The Hamming and Product Codes

The five prefix bytes of each data packet are encoded using a Hamming [8,4]-code. The Hamming codes are a very large general class of codes. If we allow shortening of the codes then every single error correcting code is a (shortened) Hamming code. (In this context we note that the Carleton code which was described in [7] is an example of a Hamming code.) There is only one Hamming code of length 8 although we have to be specific about the order of the digits. The code consists of all of the weight 4 bytes, the weight 8 byte and zero. Each codeword contains four information bits and four check bits. Thus it is a rate $\frac{1}{2}$ code. This code corrects all single bit errors and detects all double and all odd weight errors. It falsely corrects all error patterns of 4, 6 or 8 errors.

The particular version of the Hamming [8,4]-code used in BS-14 [1] is an "odd parity" implementation. If the byte is $(b_8, b_7, \dots, b_2, b_1)$ then b_8, b_6, b_4, b_2 are information bits and the odd bits are the checks. The checks are defined by,

$$b_7 = b_8 + b_6 + b_4$$

$$b_5 = b_6 + b_4 + b_2$$

$$b_3 = b_4 + b_2 + b_8$$

$$b_1 = b_2 + b_8 + b_6$$

The notation b_2 denotes the complement; $\bar{0} = 1$ and $\bar{1} = 0$. These complements are used to give the code bytes an odd overall parity, see [1], Appendix B.

The data block consists of odd parity bytes. The first error correction option is to specify (via the Packet Structure Byte) that a single byte of redundant bits is in use. There have been a number of suggestions for the particular one-byte data block code to choose. The first (and enduring) code to be proposed was the Product code [2]. This is a simple code with many good features which has often been discussed under various names (row-column code, two-way parity check etc.). The code is defined by the condition that if all 28 bytes of the data block are added mod 2 (exclusive-or) then the result is (1, 1, 1, 1, 1, 1, 1, 1). To encode, the first 27 bytes are (exclusive-or) ed together to obtain a byte U. Then the 28th byte is \bar{U} , the complement of U with 0 replaced by 1 and vice versa. Since there are 8 columns and

an odd number of ones in each column there is an even number of ones in the whole codeword. The first 27 (odd parity) bytes contain an odd number of ones, so the last byte has odd parity automatically.

There have been a number of suggestions of codes to replace this Product code as the one-byte data block code [3], [4], [6], [7]. However, we have assumed throughout the current work that the one byte code is the Product code. In looking at two byte codes we have always assumed that one of the check bytes used is the Product code check. The second byte supplements this byte in an advantageous way. The best way to add this second byte discovered so far is described below in Section 2.2. It is a Reed-Solomon code we call simply code C. Other less fruitful possibilities are discussed in Chapter 6.

2.2 The Code C

The code C which is proposed for error protections of each line (data block) of 28 bytes is a Reed-Solomon code using the last 2 bytes of the line as check symbols. This code can also be used to form an interleaved code to protect bundles of lines. The advantages of the code C include the following:

- it extends the product code
- it extends the Carleton code
- it corrects all double bit errors
- it corrects any error patterns confined to one byte
- it has a straightforward algebraic definition.

The code C is a Reed-Solomon code with symbols taken from the field F_{128} consisting of 128 elements. We take a primitive element β in F_{128} so that the powers $\beta^0, \beta^1, \beta^2, \dots, \beta^{126}$ are the 127 non-zero elements of F_{128} and $\beta^{127} = 1$. Each codeword of C is regarded as a polynomial $C(X)$ in a single variable with coefficients in F . In order to be a codeword, the polynomial must satisfy two conditions

- (i) degree $C(X) < n$ (where n is fixed)
- (ii) $C(\beta^0) = C(\beta^1) = 0$.

For a data block, n will be 28, and for interleaved codes on bundles of data blocks, when 14 codes are interleaved to protect a bundle of h lines, then n will be $2h$. In particular n is the length of the code (in symbols) and can be at most 127.

The bytes in the data block must now be related to the symbols from the field F . In order to do this we let α be a primitive

element of F and take α to be a root of the polynomial $X^7 + X^3 + 1$, so that

$$\alpha^7 + \alpha^3 + 1 = 0.$$

We further require that no polynomial of lower degree, with coefficients in $F_2 = \{0,1\}$ has α as a root. Then the first 7 powers of α , namely $1, \alpha, \alpha^2, \dots, \alpha^6$ are linearly independent over F_2 .

Each field element γ of F can be uniquely represented by a 7-tuple of bits by $\gamma \rightarrow (b_0, b_1, \dots, b_6)$ when $\gamma = b_0 + b_1\alpha + \dots + b_6\alpha^6$. For example $\alpha^8 \in F$ and, since $\alpha^7 = 1 + \alpha^3$, we get $\alpha^8 = \alpha(1 + \alpha^3) = \alpha + \alpha^4$ so $\alpha^8 \rightarrow (0, 1, 0, 0, 1, 0, 0)$. To correspond to bytes we need 8-tuples of bits and we observe that every field element of F has exactly two representations in terms of the first 8 powers of α where we can change from one representation to the other by adding $0 = 1 + \alpha^3 + \alpha^7$. Thus

$$\alpha^8 = \alpha + \alpha^4 = 1 + \alpha + \alpha^3 + \alpha^4 + \alpha^7.$$

Since adding $1 + \alpha^3 + \alpha^7$ is the same as performing an "exclusive-or" operation with the byte $(1, 0, 0, 1, 0, 0, 0, 1) \rightarrow 1 + \alpha^3 + \alpha^7$ one representation has odd parity, the other has even parity.

By fixing the parity (to be odd say) we have exactly one byte for each field element.

2.3 Encoding

Each of the bytes B_0, B_1, \dots, B_{n-3} in a data-block represent field elements and we require B_{n-2} and B_{n-1} such that

$$C(1) = B_0 + B_1 + \dots + B_{n-1} = 0$$

$$C(\beta) = B_0 + B_1\beta + \dots + B_{n-1}\beta^{n-1} = 0$$

Since we are working in a field of characteristic 2, $C(1)$ is simply the exclusive-or of the corresponding bytes. Thus a codeword of C is also a codeword of the product code (if odd parity is desired for transmission, then B_{n-1} must be replaced by \bar{B}_{n-1} .)

If we now choose $\beta = \alpha^8$ and write B_0, B_1, \dots, B_{n-1} as a bit string $b_0 b_1 \dots b_{8n-1}$, then $B_0 = b_0 + \dots + b_1 \alpha^1$, $B_1 \beta = b_8 \alpha^8 + \dots + b_{15} \alpha^{15}, \dots$ and

$$C(\beta) = \sum_{i=0}^{8n-1} b_i \alpha^i = 0$$

since $B_0 + B_1 \beta + \dots = (b_0 + b_1 \alpha + \dots + b_7 \alpha^7) + (b_8 + b_9 \alpha + \dots + b_{15} \alpha^7) \alpha^8 + \dots$

This means that every codeword of C is also a codeword of the Carleton code as defined in [Fig. 1.1].

To determine B_{n-2} and B_{n-1} we set S and T to be zero bytes and successively replace S by $S + B_i$ and T by $T \beta^{-1} + B_i$ for $i = 0, 1, 2, \dots, n-3$. Then the end values are

$$S = \sum_{i=0}^{n-3} B_i, \quad T = \frac{1}{\beta^{n-3}} \sum_{i=0}^{n-3} B_i \beta^i$$

We require

$$\left\{ \begin{array}{l} S + B_{n-2} + B_{n-1} = 0 \\ T\beta^{n-3} + B_{n-2}\beta^{n-2} + B_{n-1}\beta^{n-1} = 0 \end{array} \right. \text{ i.e. } \left\{ \begin{array}{l} B_{n-2} + B_{n-1} = S \\ B_{n-2} + B_{n-1}\beta = T\beta^{-1} \end{array} \right.$$

The solution of these two equations gives us

$$B_{n-2} = (S\beta + T\beta^{-1})(1+\beta)^{-1}$$

$$B_{n-1} = (S + T\beta^{-1})(1+\beta)^{-1} .$$

We note that, in terms of α , $\beta^{-1} = \alpha + \alpha^5 + \alpha^6$, $(1+\beta)^{-1} = \alpha^6$.

2.4 Decoding Code C (in theory)

The condition that a codeword of C must satisfy the relations $C(\beta) = C(1) = 0$ allows the code to correct errors. Suppose a codeword $C(X)$ is sent and a single symbol (byte) is received in error. Thus for some e_j the message $(B_0, B_1, \dots, B_{n-1})$ was sent and $(B_0, \dots, B_j + e_j, \dots, B_{n-1})$ arrived. Writing this as polynomials $R(X) = C(X) + E(X)$ is received where $E(X) = e_j X^j$. Now we evaluate $R(X)$ at $\beta^0 = 1$ and β^1 noting that $C(\beta^0) = C(\beta^1) = 0$. We obtain,

$$\begin{aligned} R(\beta^0) &= C(\beta^0) + E(\beta^0) \\ &= E(\beta^0) \\ &= e_j \beta^{0 \cdot j} = e_j \end{aligned}$$

which is the error value, and

$$\begin{aligned} R(\beta) &= C(\beta) + E(\beta) \\ &= E(\beta) \\ &= e_j \beta^j . \end{aligned}$$

Thus we write $R(\beta)/R(1)$ as a power β^j of β to obtain j , the index of the faulty byte. Then $R(1)$ is the quantity to be subtracted to correct the error. This proves that C is single symbol correcting.

Now suppose that the error polynomial has the form $E(X) = e_j X^j + e_k X^k$ so that errors occurred in the j -th and k -th symbols. Suppose that the integers j and k are known. This is the erasure situation.

Again we evaluate at $\beta^0 = 1$ and $\beta^1 = \beta$. Then

$$\begin{aligned} e_j + e_k &= R(1) \\ e_j \beta^j + e_k \beta^k &= R(\beta) \end{aligned}$$

are known as are the coefficients β^j and β^k . Thus we can solve for e_j and e_k (since $j \neq k$) and correct the errors. Therefore, two symbols whose locations are known can be corrected.

2.5 Bundle Codes

In order to protect bundles of data blocks we use an additional block for error control. A bundle consists of h data blocks and

The discussion above uses 13 vertical codewords whereas the description in Chapter 1 and the calculations of Chapter 3 assume 14 vertical codewords. The difference is in the matter of whether the check block is encoded with a data block code or not. It is best in fact to use the 13 vertical codewords since there is no advantage in correcting errors in the check bytes of the data blocks. Using 14 vertical codewords gives slightly worse performance and so our performance estimates are, if anything, conservative.

We emphasize that in the arrangement we have specified the code used for the vertical and horizontal codewords is the same (though of different lengths). Thus only one decoder is necessary and using the bundle code involves adding a portion of overhead to the decoding program but not a new error correction procedure. Also we have described in Chapter 1 other arrangements for bundle coding which are based on single byte codes and double check blocks.

2.6 Encoding and Decoding the Bundle Code

We have discussed above the theoretical decoding procedures that demonstrate the capabilities of the code C . Since this is a Reed-Solomon code it is susceptible to any of the decoding algorithms which have been proposed for this class of codes. It is, however,

a particularly simple example of such a code and it is quite feasible to implement the procedures indicated in Sections 2.3 and 2.4. In this regard the paper [14] may be of interest.

We have implemented a decoder for code C which decodes a single codeword in less than a millisecond using a Motorola 6809 microprocessor running at a clock speed of 1.29 Mhz.

If code C is also used for the vertical code in the bundle then each of these codewords would also be decoded in less than a millisecond. A bundle decoder would have a code C decoder as a subroutine. It would have to decide for each bundle whether it was going to simply decode the vertical codewords or attempt to replace a missing data block. In the latter case the index of the missing data block would have to be passed to the code C decoder along with an indication that these are the relevant bytes to correct.

CHAPTER 3

PERFORMANCE OF THE BUNDLE CODE SYSTEM

3.1 System Performance with Independent Errors

This chapter is devoted to the calculation of the expected performance of the complete Bundle Code system. The performance is measured by calculating the expected number of bundles processed before a bundle is not correctly decoded (either through a decoding failure or decoding error). In all sections except the last we assume that the errors are independent events while in the final section we look at errors clustered in bursts.

The performance of a particular code depends very much on the decoding strategy used. The Bundle Code is actually a complex combination of many codewords from several codes. This means that a number of decoding strategies are possible. We will concentrate on one particular decoding procedure throughout most of this chapter; we will call it Full Decoding.

The Full Decoding procedure decodes each prefix and then each data block as it arrives. Then the vertical codewords are decoded. If two or more prefixes have uncorrectable (but detected) errors then we declare a decoding failure. If exactly one prefix has a decoding

failure, then when the vertical codewords are decoded, this decoding is an erasure decoding which attempts to replace the bytes in the data block of the defective prefix. If all prefixes are correctable then the vertical codewords are decoded bitwise: either one-byte-error or two-byte-erasure whichever is indicated by the byte-parity failures.

The Full Decoding procedure correctly decodes a bundle if either the prefixes are all correctable and the horizontal and vertical codes can correct the errors in the data blocks or there is exactly one of the h prefixes with an uncorrectable error and all the errors in the other $(h-1)$ data blocks are corrected by the horizontal code (thereby leaving the vertical codewords free to replace the missing data block). We therefore have four probabilities to calculate:

P_{PCD} = probability that a single prefix is correctly decoded,

P_{HVCD} = probability that the Hor. & Vert. codewords are correctly decoded given that all prefixes are correctly decoded,

P_{HCD} = probability that a single horizontal codeword is correctly decoded

P_{CD} = probability that a bundle of h data packets is correctly decoded.

Then for a bundle of h data packets (i.e. a bundle of length h) we easily have a formula;

$$P_{CD} = (P_{PCD})^h (P_{HVCD}) + h(1-P_{PCD})(P_{PCD})^{h-1} (P_{HCD})^{h-1}.$$

The first term comes from the case in which all prefixes are correctly decoded and the second from that in which exactly one of the prefixes fails. We will show how to calculate P_{HVCD} in section 3.3 and the next section treats the other probabilities.

3.2 The Probability of Correct Decoding for a t -Error Correcting Code

If we have independent errors which occur with probability p and use a length n code which corrects t bit errors then there is a simple formula for the probability of a correct decoding. For i less than or equal to t there are $\binom{n}{i}$ patterns of i errors and each of them occurs with probability $p^i q^{n-i}$ where $q = 1-p$ is the probability of a correct bit. Then the probability of a correct decoding is the probability that one of these patterns occurs which is

$$\sum_{i=0}^t \binom{n}{i} p^i q^{n-i}.$$

In the particular case of an $[8,4]$ -Hamming code this is

$$q^8 + 8pq^7$$

since the code is single error correcting. For a double error correcting data block code this is

$$q^{224} + 224pq^{223} + 24976p^2q^{222}$$

since $\binom{224}{2} = 24976$. (For a single error correcting data block code only the first two terms are used). We can then write

$$P_{PCD} = (q^8 + 8pq^7)^5$$

since a correct prefix has 5 correct $[8,4]$ -Hamming codewords and

$$P_{HCD} = q^{224} + 224pq^{223} + 24976p^2q^{222}.$$

We note in passing that if the code acted bitwise then q^8 is the probability that a byte is correctly received. Hence the probability that a length b bytes, single-byte error correcting code decodes correctly is

$$(q^8)^b + b(1-q^8)(q^8)^{b-1}.$$

If we let $n = 8b$ be the length in bits this is

$$q^n + \binom{n}{8}(1-q^8)q^{n-8}.$$

This is used in analysis of some of the decoding strategies.

3.3 The Vertical and Horizontal Codewords and Pernicious Error Patterns

The only probability left to calculate is the probability that a bundle of horizontal and vertical codewords decodes correctly; that is, P_{HVCD} . We continue to assume that errors are independent. In this

case we will calculate the probability P_{HVCD} by estimating $1 - P_{HVCD}$, the probability of an incorrect decoding.

Suppose t errors occur. This event arises with probability $p^t q^{n-t}$ where $q = 1 - p$ and $n = 224h$ is the number of bits in the (data blocks of the) bundle. We must then count the number N_t of patterns of t errors which are not correctable. The sum $\sum_t N_t p^t q^{n-t}$ then gives the probability of an incorrect decoding. The major problem of this section is the estimation of the numbers N_t for small t ($t < 11$). We find,

$$N_0 = N_1 = N_2 = 0,$$

$$N_3 = h \ 6272,$$

$$N_4 = \binom{h}{2} 2.8099E6 + h 1.3215E6,$$

$$N_5 = \binom{h}{3} 9.4411E8 + \binom{h}{2} 9.0536E8 + h 1.3737E8,$$

$$N_6 = \binom{h}{4} 2.8197E11 + \binom{h}{3} 4.0947E11 + \binom{h}{2} 3.8654E11 + h 9.5185E9,$$

$$N_7 \approx \binom{h}{5} 7.8953E13 + \binom{h}{4} 1.5373E14 + \binom{h}{3} 2.5081E14 + \binom{h}{2} 4.6303E13,$$

$$N_8 \approx \binom{h}{6} 2.1223E16 + \binom{h}{5} 5.1849E16 + \binom{h}{4} 1.1454E17 + \binom{h}{3} 5.3579E16,$$

$$N_9 \approx \binom{h}{7} 5.5462E18 + \binom{h}{6} 1.6303E19 + \binom{h}{5} 4.4148E19 + \binom{h}{4} 3.3981E19,$$

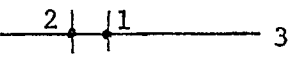
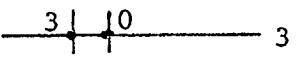
$$N_{10} \approx \binom{h}{8} 1.4198E21 + \binom{h}{7} 4.8788E21 + \binom{h}{6} 1.5377E22 + \binom{h}{5} 1.0755E22,$$

$$N_{11} \approx \binom{h}{9} 3.5780E23 + \binom{h}{8} 1.4073E24 + \binom{h}{7} 5.0049E24 + \binom{h}{6} 4.3280E24.$$

The t -errors are first attacked by the horizontal code. This code is decoded so as to correct all patterns of one or two errors in

any one data block. If more than two errors occur then they will survive to confront the vertical code. Now there is an important observation. As with any code there will be patterns of four or more errors which cause a decoding error; that is they may fool the horizontal decoder into making a false correction or ignoring them. Therefore the horizontal decoder may add errors to those already present. We will assume that this does not happen for the moment. We will take up this problem again once we have counted the most significant error patterns. We are considering the case of 14 vertical codewords each of which contains 2 bytes from each horizontal codeword.

Suppose that t errors occur. If $t = 0, 1$ or 2 then the horizontal codewords correct the errors and a correct decoding results. If $t = 3$ then the only uncorrectable pattern is to have all three errors in the same horizontal codeword and the same vertical codeword. To see this, simply observe that if the errors corrupt at least 2 horizontal codewords then the horizontal code corrects them since there are at most 2 errors in each. Similarly the three errors must be in one vertical codeword. Since the intersection of a vertical and horizontal codeword is two bytes, there are two possibilities for the errors, namely,

- 3(i) two in one byte and one in another 
- 3(ii) all three in one byte 

The diagrams contain one line for each horizontal codeword containing an error and two lines for each erroneous vertical codeword. An intersection represents a byte shared by a horizontal and vertical codeword and the number beside it indicates the number of errors in this byte (zeros are generally suppressed). The number at the end of each horizontal codeword is the number of errors in this codeword.

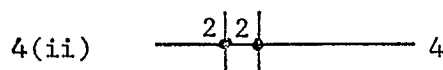
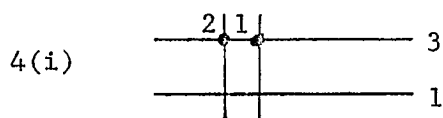
The type 3(ii) errors are correctable so only the type 3(i) errors contribute to N_3 , the number of uncorrectable triple error patterns. Let h be the number of horizontal codewords (thus h is the bundle length) and let $v = 14$ be the number of vertical codewords. It is convenient in the calculations that follow, to treat v as a parameter (with only value).

We now count the number of triple error patterns of type 3(i). There are h choices for the horizontal codeword and v choices for the vertical codeword. Once these choices are made there are 2 choices for the byte with one error, 8 bits in which that one error can occur and, independently, $28 = \binom{8}{2}$ ways for two errors to corrupt the other byte. We find then that $hv(8)(28)$ patterns of three errors result in either a decoding error or a decoding failure. Thus

$$N_3 = h(6272).$$

Now suppose $t = 4$. There are two possibilities - either the errors are all in one horizontal codeword or they are in two. If they

are in more than this, they will be corrected (since no codeword contains more than two errors). In addition, if the four errors are in two horizontal codewords then there must be three of them in one and one in the other. In fact, the single error will be corrected by the horizontal codeword leaving only three errors to confront the vertical codewords. This leaves us two types of patterns for four errors which are not correctable:



The type 4(i) error patterns are counted as follows. There are $\binom{h}{2}$ choices for the horizontal codewords, then 2 choices for the one to have a single error. Then there are v choices for the vertical codeword. As before, there are $(2)(28)(8)$ patterns for the three errors in the horizontal codeword with three errors and 224 places for the single error in the other horizontal codeword since it can be any of the 224 bits of this codeword.

The type 4(ii) error patterns can corrupt any of h horizontal codewords and any of v vertical codewords. Unless the four errors occur as 2 in each of the 2 intersection bytes the errors are correctable. There are $hv \binom{8}{2} \binom{8}{2}$ patterns of type 4(ii).

Therefore four errors produce a decoding error or failure in $\binom{h}{2}v(2)(2)(28)(8)(224)+h.v(28)(28)$ ways. Thus $N_4 = \binom{h}{2}(2809856) + h(10976)$.

As the number of errors increases this counting process becomes extremely complicated. The number of cases to be considered is large. We can simplify our calculations considerably by noting that many error patterns of t errors are actually a pattern of $(t-1)$ -errors, which is uncorrectable, plus an additional error in a new horizontal codeword. For example there are $\binom{h}{4}4v2\binom{8}{2}8(224)^3$ uncorrectable error patterns of 6 errors distributed across four horizontal codewords as 3-1-1-1. If 7 errors are distributed across five horizontal codewords as 3-1-1-1-1 then they can occur in $\binom{h}{5}5v2\binom{8}{2}8(224)^4$ ways. In other words, we multiply the first number by $\frac{5}{4}(224)$ to obtain the second. A similar method applies when $(t-2)$ -uncorrectable errors are augmented by a double error in a new horizontal codeword. For example, there are $h(1.3737E8)$ patterns of 5 errors in a single line which are uncorrectable. These can be increased to 7 uncorrectable errors distributed 5-2 in two horizontal codewords in $\binom{h}{2}2(1.3737E8)\binom{224}{2}$ ways.

In fact, these techniques leave us with only the error patterns with at least 3 errors in every horizontal data block. Denote by $a_1 - a_2 - \dots - a_m$ the distribution of $t = a_1 + a_2 + \dots + a_m$ errors into m horizontal codewords with a_i errors in the i^{th} codeword.

Then the cases we must calculate individually are 3, 4, 5, 3-3, 6, 4-3, 7, 4-4, 5-3, 8, We can in fact obtain the required precision by using only 3, 4, 5, 3-3, 6 and 4-3 (indeed 6 is not particularly significant). The first two cases have been considered and we deal with the other four below.

For five errors in a single horizontal codeword we count the number of uncorrectable patterns as follows.

$$\begin{array}{c} 4 | 1 \\ \hline \bullet \quad \bullet \end{array} \quad 5 \quad \text{h.v.} 2 \binom{8}{4} 8 = \text{h.v.}(1120)$$

$$\begin{array}{c} 3 | 2 \\ \hline \bullet \quad \bullet \end{array} \quad 5 \quad \text{h.v.} 2 \binom{8}{3} \binom{8}{2} = \text{h.v.}(3136)$$

$$\begin{array}{c} 2 | 2 + 1 \\ \hline \bullet \quad \bullet \end{array} \quad 5 \quad \text{h.v.} \binom{8}{2}^2 208 = \text{h.v.}(163072)$$

$$\begin{array}{c} 2 | 1 + 2 \\ \hline \bullet \quad \bullet \end{array} \quad 5 \quad \text{h.v.} 2 \binom{8}{2} 8 \binom{208}{2} = \text{h.v.}(9644544)$$

The total is $\text{h}(1.3737\text{E}8)$.

For six errors in a single horizontal codeword we obtain the following.

$$\begin{array}{c} 4 | 2 \\ \hline \bullet \quad \bullet \end{array} \quad 6 \quad \text{h.v.} 2 \binom{8}{4} \binom{8}{2} = \text{h.v.}(3920)$$

$$\begin{array}{c} 4 | 1 + 1 \\ \hline \bullet \quad \bullet \end{array} \quad 6 \quad \text{h.v.} 2 \binom{8}{4} 8 (208) = \text{h.v.}(232960)$$

$$\begin{array}{c} 3 | 2 + 1 \\ \hline \bullet \quad \bullet \end{array} \quad 6 \quad \text{h.v.} 2 \binom{8}{3} \binom{8}{2} (208) = \text{h.v.}(652288)$$

$$\begin{array}{c} 2 \quad | \quad 2 \quad + \quad 2 \\ \hline \end{array} \quad 6 \quad h.v \binom{8}{2}^2 \binom{208}{2} = h.v(16877952)$$

$$\begin{array}{c} 2 \quad | \quad 1 \quad + \quad 3 \\ \hline \end{array} \quad 6 \quad h.v. 2 \binom{8}{2} 8 \binom{208}{3} - h \binom{v}{2} (2 \binom{8}{2} 8)^2$$

$$= h.v(660432136) .$$

The first term in the last formula counts the ways of placing the three additional errors anywhere in the data block not in the 16 bits of the chosen vertical codeword. This counts the error patterns $\begin{array}{c} 2 \quad | \quad 1 \quad 2 \quad | \quad 1 \\ \hline \end{array}$ twice each. Hence the number of patterns of this form are subtracted.

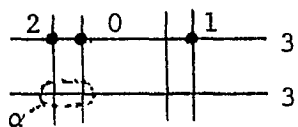
We come then to six errors distributed 3-3 in two horizontal codewords. The symbol α on a diagram indicates the intersection of some pair of horizontal and vertical codewords. First two horizontal codewords are picked. Consider the way the three errors in the first of these fall into vertical codewords. They can corrupt 3, 2 or 1 vertical codeword. For each possibility we count the ways in which the three errors in the second horizontal codeword can make the set of six errors uncorrectable.

$$\begin{array}{c} 1 \quad | \quad 1 \quad | \quad 1 \\ \hline \end{array} \quad 3 \quad \binom{h}{2} \binom{v}{3} (16)^3 \{ 3 \binom{16}{2} (208) + 3 \left(\binom{16}{3} - 2 \binom{8}{3} \right) \} = \binom{h}{2} (1.136457E11)$$

$$\begin{array}{c} 1 \quad | \quad 1 \quad | \quad 1 \\ \hline \end{array} \quad 3 \quad \binom{h}{2} \binom{v}{2} 2(8)^3 \{ 2 \binom{8}{2} 200 + 16 \binom{208}{2} + \binom{16}{2} 208 + \binom{16}{3} \}$$

of errors in $\alpha = \quad 0 \quad \quad 1 \quad \quad 2 \quad \quad 3$

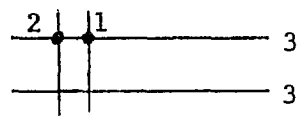
$$= \binom{h}{2} (7.103752E10)$$



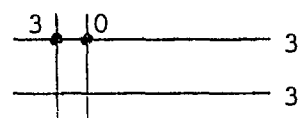
$$\binom{h}{2} \binom{v}{2} 2(2) \binom{8}{2} 16 \{ 12 \binom{8}{2} 200 + 16 \binom{208}{2} + \binom{16}{2} 208 + \binom{16}{3} \}$$

of errors in $\alpha =$ 0 1 2 3

$$= \binom{h}{2} (6.270053E10)$$



$$\binom{h}{2} v 2 \binom{8}{2} 8 \binom{224}{3} = \binom{h}{2} (1.159206E10)$$



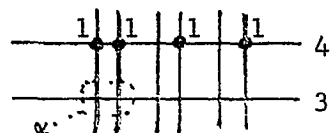
$$\binom{h}{2} v 2 \binom{8}{3} 2 \binom{8}{2} 216 = \binom{h}{2} (1.896653E7)$$

The total is $\binom{h}{2} (2.5899479E11)$.

We then proceed to the case of seven errors distributed as four in one horizontal codeword and three in another. The technique is the same.

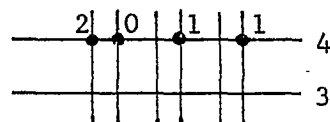


$$\binom{h}{2} 2 \binom{v}{4} (16)^4 \{ 4 \binom{16}{2} 208 + 4 \binom{16}{3} - 2 \binom{8}{3} \} = \binom{h}{2} (1.333443E11)$$

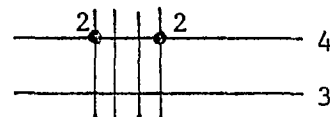


$$\binom{h}{2} 2 \binom{v}{3} 3 (8)^4 \{ 4 \binom{8}{2} 200 + 16 \binom{223}{2} \} = \binom{h}{2} (3.743295E12)$$

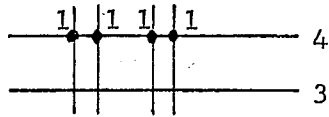
of errors in $\alpha =$ 0 ≥ 1



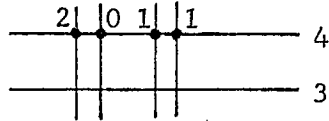
$$\binom{h}{2} 2 v 2 \binom{8}{2} \binom{208}{2} \{ 4 \binom{8}{2} 200 + 16 \binom{223}{2} \} = \binom{h}{2} (1.412509E13)$$



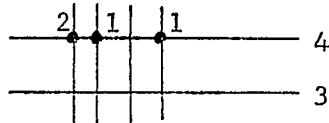
$$\binom{h}{2} 2 \binom{v}{2} (2 \binom{8}{2})^2 4 (8) \binom{223}{2} = \binom{h}{2} (4.520904E11)$$



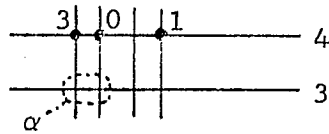
$$\binom{h}{2} 2 \binom{v}{2} (8)^4 4(8) \binom{223}{2} = \binom{h}{2} (5.904854E11)$$



$$\binom{h}{2} 2 \binom{v}{2} 2(2) \binom{8}{2} 8^2 (32) \binom{223}{2} = \binom{h}{2} (1.033349E12)$$



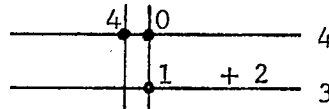
$$\binom{h}{2} 2 v 2 \binom{8}{2} 8 (208) \binom{224}{3} = \binom{h}{2} (4.822297E12)$$



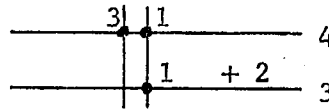
$$\binom{h}{2} 2 v 2 \binom{8}{3} 208 \{ 2 \binom{8}{2} 200 + 2 \binom{8}{2} + \binom{16}{2} (208) + [\binom{16}{3} - 2 \binom{8}{3} \}$$

of errors in $\alpha =$ 0 1 2 3

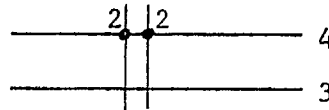
$$= \binom{h}{2} (2.391549E10)$$



$$\binom{h}{2} 2 v 2 \binom{8}{4} 16 \binom{223}{2} = \binom{h}{2} (1.552508E9)$$



$$\binom{h}{2} 2 v 2 \binom{8}{3} 8 (16) \binom{223}{2} = \binom{h}{2} (9.936052E9)$$



$$\binom{h}{2} 2 v \binom{8}{2}^2 \binom{224}{3} = \binom{h}{2} (4.057221E10)$$

The total is $\binom{h}{2} (3.817701514E13)$.

We have now counted the number of pernicious error patterns of the forms 3, 4, 5, 3-3, 6, 4-3 and indicated how to obtain the number of non-correctable error patterns obtained by adding single and double errors in new data blocks of the bundle. There is one

additional factor to be added, namely the decoding errors in the horizontal codewords.

The errors which the vertical codewords confront include both the ones left uncorrected by the horizontal code and those added as decoding errors by this code. In fact, the decoding errors cannot make an uncorrectable error correctable, unfortunately, however, the reverse can happen. It is possible that the horizontal code may corrupt a correctable pattern into an uncorrectable one. So far we have not counted these.

The simplest situation occurs when we have a codeword of weight 6 (the smallest non-zero weight) for the horizontal code of a special sort. Every weight 6 codeword of the horizontal code (code C) has two ones in each of three bytes and every pair of bytes has a one in exactly one common position. For example the bytes might be (0100100), (0100001) and (0000101). If two of these bytes are separated by 14 bytes they will both fall in the same vertical codeword and if the ones mark the location of six errors the total pattern is not correctable. Now if errors occur in exactly four of the positions indicated by the ones then the horizontal code will add the other two and leave six errors (a decoding error event). If the four errors occur in 2 of the positions in the same vertical codeword and then the other 2 (thus in 6 ways) the original error

error pattern was correctable but the added errors yield an uncorrectable error pattern. Let A_6^* be the number of codewords of weight 6 of the horizontal code with ones in two bytes of some vertical codeword (i.e. 14 bytes apart). Then there are $h6A_6^*$ patterns of four errors that are included in N_4 which we have not counted yet. We find that A_6^* is 1063. See Appendix A for the method of calculation.

A decoding error from five errors can't turn a correctable pattern into an uncorrectable pattern. This follows since taking one error away from a set of six in the positions of a codeword of weight 6 cannot give an uncorrectable pattern.

We have not proceeded any further with these calculations. The terms involving six or more errors in a single data block are not very significant in our calculations, and this added adjustment would not be noticed.

Using the counts for 3, 4, 3-3, 6 and 4-3, the method of extending by adding single and double errors and the extra patterns from decoding errors on quadruple errors we come finally to the Table 3.1 which records the number of error patterns of the various distributions with the terms $\binom{h}{m}$ suppressed. (Entries marked "*" have not been calculated. We stop at eleven errors.) An entry labelled $a_1 - a_2 - a_3 \dots - a_m$ will have a coefficient $\binom{h}{m}$.

3 errors		4 errors		5 errors		6 errors		7 errors	
3	6272	3-1	2.8099E6	3-1-1	9.4411E8	3-1-1-1	2.8197E11	3-1-1-1-1	7.8953E13
		4	1.3215E6	4-1	5.9206E8	4-1-1	1.9893E11	4-1-1-1	5.9413E13
				3-2	3.1330E8	3-2-1	2.1054E11	3-2-1-1	9.4321E13
				5	1.3737E8	5-1	6.1540E10	5-1-1	2.0677E13
						4-2	6.6013E10	4-2-1	4.4361E13
						3-3	2.5899E11	3-3-1	1.7404E14
						6	9.5185E9	3-2-2	1.1737E13
								6-1	1.2643E12
								5-2	6.8617E12
								4-3	3.8177E13
								7	*

Table 3.1 The Number of Error Patterns

Which are Uncorrectable Arranged

by Distribution in

Horizontal Codewords

Note: A distribution $a_1 - a_2 - \dots - a_m$ on m horizontal codewords requires a coefficient $\binom{h}{m}$.

The coefficients $\binom{h}{m}$ have been suppressed.

8 errors		9 errors		10 errors		11 errors	
3-1-1-1-1-1	2.1223E16	3-1-1-1-1-1-1	5.5462E18	3-1-1-1-1-1-1-1	1.4198E21	3-1-1-1-1-1-1-1-1	3.5780E23
4-1-1-1-1	1.6636E16	4-1-1-1-1-1	4.4717E18	4-1-1-1-1-1-1	1.1686E21	4-1-1-1-1-1-1-1	2.9916E23
3-2-1-1-1	3.5213E16	3 2-1-1-1	1.1831E19	3-2-1-1-1-1-1	3.7102E21	3-2-1-1-1-1-1-1	1.1081E24
5-1-1-1	6.1757E15	5-1-1-1-1	1.7292E18	5-1-1-1-1-1	4.6481E20	5-1-1-1-1-1-1	1.2147E23
4-2-1-1	1.9874E16	4-2-1-1-1	7.4195E18	4-2-1-1-1-1	2.4930E21	4-2-1-1-1-1-1	7.8179E23
3-3-1-1	7.7972E16	3-3-1-1-1	2.9110E19	3-3-1-1-1-1	9.7810E21	3-3-1-1-1-1-1	3.0673E24
3-2-2-1	1.0517E16	3-2-2-1-1	5.8894E18	3-2-2-1-1-1	2.6385E21	3-2-2-1-1-1-1	1.0343E24
6-1-1	1.4328E15	6-1-1-1	4.2793E17	6-1-1-1-1	1.1982E20	6-1-1-1-1-1	3.2208E22
5-2-1	4.6111E15	5-2-1-1	2.0658E18	5-2-1-1-1	7.7123E20	5-2-1-1-1-1	2.5913E23
4-3-1	2.5656E16	4-3-1-1	1.1493E19	4-3-1-1-1	4.2907E21	4-3-1-1-1-1	1.4417E24
4-2-2	2.4731E15	4-2-2-1	2.2159E18	4-2-2-1-1	1.2409E21	4-2-2-1-1-1	5.5592E23
3-3-2	1.9406E16	3-3-2-1	1.7388E19	3-3-2-1-1	3.8949E21	3-3-2-1-1-1	1.7449E24
7-1	*	3-2-2-2	3.9087E17	3-2-2-2-1	4.3777E20	3-2-2-2-1-1	2.9418E23
6-2	4.7547E14	6-2-1	3.1951E17	6-2-1-1	1.4314E20	6-2-1-1-1	5.3439E22
5-3	*	5-2-2	2.5707E17	5-2-2-1	2.3033E20	5-2-2-1-1	1.2899E23
4-4	*	4-3-2	2.8605E18	4-3-2-1	2.5630E21	4-3-2-1-1	1.4353E24
8	*	and others		4-2-2-2	8.2357E19	4-2-2-2-1	9.2240E22
				3-3-2-2	9.6937E20	3-3-2-2-1	1.0857E24
				6-2-2	3.1952E17	3-2-2-2-2	1.2203E22
				and others		6-2-2-1	2.8629E20
						5-2-2-2	8.5608E21
						and others	

Table 3.1 (continued)

In this Section we are trying to calculate the probability, P_{HVCD} , that the decoding of horizontal then vertical codewords results in a correct decoding. We have

$$1 - P_{\text{HVCD}} = q^n \sum_{t=0}^n N_t x^t$$

where $x = p/q$ and N_t is the number patterns of t errors which are not correctable. The parameter $n = 224h$ is the bundle length in bits. We have calculated or approximated N_t for $t \leq 11$.

The formulas were given at the beginning of this Section. Note

that the dependence on h is non-linear and that $\binom{h}{m} = 0$ if $h < m$.

We are now in a position to calculate P_{HVCD} . The question is now; how accurate is our calculation? The convergence of the series $\sum N_t x^t$ depends crucially on both the bundle length h and the bit error rate p (reflected in $x = p/(1-p)$). We look at the case $h = 10$ and $p = 10^{-3}$. We find

t	N_t	$N_t x^t$
3	6.272E4	6.6889E-6
4	1.3966E8	1.4909E-5
5	1.5541E11	1.6607E-5
6	1.2584E14	1.3461E-5
7	8.4460E16	9.0435E-6
8	4.9897E19	5.3480E-6
9	2.4603E22	2.6397E-6
10	1.0300E25	1.1061E-6
11	5.5407E28	5.9564E-7

The series converges slowly but the terms beyond $t = 7$ decrease by more than $\frac{1}{2}$. We are confident then that 2 significant digits in our results are correct. Note that the largest term is N_5 not N_3 . If h is too small then the largest terms are the ones we have ignored so these calculations do not yield anything of value. The code works better if h is small but the method of calculation falls apart.

The tremendous amount of work in calculating this one number P_{HVCD} has to be repeated for each new decoding scheme. In fact, this is one of the worst decoders to analyze so if you have mastered the calculation for Full Decoding then the other situations will not require any new ideas.

3.4 Putting the Pieces Together

We are after P_{CD} the probability that a bundle is correctly decoded when errors are independent at a bit error rate p . We have seen in Section 3.1 that

$$P_{CD} = (P_{PCD})^h (P_{HVCD}) + h(1 - P_{PCD})^{h-1} P_{PCD}^{h-1} P_{HCD}$$

We showed how to calculate P_{PCD} and P_{HCD} in Section 3.2 and how to obtain P_{HVCD} in Section 3.3. We can now calculate P_{CD} :

The only remaining problem is round off error - the demon that is always with us.

It is instructive to look at the relative sizes of the two terms in the formula for P_{CD} . Some sample values are given in Table 3.2. We see that the first term is by far the largest indicating that it is most common for all the prefixes to be correctly decoded. The bundle code works well because the second term covers most of the remaining cases.

Table 3.2 Some Sample Probabilities

h	=	10
p	=	.001
P_{PCD}	=	.999859
P_{HCD}	=	.998421
P_{HVCD}	=	.9999296
$P_{PCD}^h P_{HVCD}$	=	.998519
$h(1 - P_{PCD})P_{PCD}^{h-1} P_{HCD}^{h-1}$	=	.00138974

The Table 3.3 presents the expected number of bundles until a non-correct decoding for various values of p and h . This expected number is taken as $1 / (1 - P_{CD})$.

3.5 Bundle Length Effects

We can now calculate the expected number of bundles before an incorrect decoding at various bundle lengths h and bit error rates p . We postpone short bundles till later. We should really look at something else though. We are interested in communicating information. Of the 224 data block bits and 40 prefix bits $(7 \times 26) + (4 \times 5) = 202$ bits are information and 64 bits are checks. In the last data packet of a bundle only 20 bits (from the prefix) are information; the rest are checks. Thus of the $266h$ bits in the bundle only $20h + 182(h-1) = 202h - 182$ bits are information. So we can consider the expected number of information bits before an uncorrectable bundle arrives as a function of h for fixed values of bit error rate. We have shown some results in Table 3.4. ($BER = 10^{-3}$).

Table 3.4 Expected Number of Information Bits Before
an Incorrect Decoding

<u>Bundle Length</u>	<u>Number of Information Bits</u>
5	2.21E7
6	2.19E7
7	2.17E7
8	2.07E7
9	2.02E7
10	2.02E7
11	1.97E7
12	1.92E7
13	1.87E7
14	1.82E7
15	1.78E7

We observe that two countervailing forces nearly balance. Longer bundle codes are more susceptible to error but carry a relatively larger amount of information. Our calculations show that in fact the amount of information expected to arrive before the first incorrect decoding is roughly constant and decreases slightly as the bundles get longer. This trend, in the reverse direction, is not necessarily correct for shorter bundles. In fact, there is likely to be an optimal bundle length (of perhaps 5 data packets?). We emphasize that this bundle length would only be "optimal" for independent errors at a particular bit error rate. At other rates and with bursts of errors the situation will be difficult. Our conclusion should be that if we avoid short bundles (say $h < 5$) then we can be quite flexible about the bundle length. This is not to say that short bundles give bad performance. On the contrary, they give excellent performance at a depressed information rate. If the application dictates a bundle of length 2 or 3, then the code will work very well indeed.

3.6 Burst Errors

So far we have assumed that the errors are independent events. This is a convenient assumption since it characterizes the channel in terms of only one parameter, the bit error rate, and makes

calculations of performance parameters possible. We have no reason to suppose that the errors encountered in a teletext channel will be independent. We must sooner or later address the problem of burst errors.

To predict performance of a coding system in a burst error environment is a difficult or impossible task. We have to be quite specific about the sort of burstiness the code is confronting before we can conclude anything. In the absence of field data we are reduced to general, qualitative remarks.

We can describe the bursts which are correctable. The most important class consists of those bursts which are confined to a single data packet. These will be detected with high probability either since they render the prefix unrecognizable or because they are detected by the horizontal code. In either case erasure decoding of the vertical codewords will replace the whole data block of the corrupted packet and the burst is corrected. Even if the burst is undetected, if it is less than 106 bits ($106 = 2 + 8(14-1)$) it will be corrected by the vertical code by error correction. In each case all errors outside the burst must be removed by the horizontal codewords first. There is also the possibility that a collection of several short bursts will corrupt at most one byte from each vertical codeword and hence will be correctable.

The problem comes when the data packets are broadcast consecutively in the channel. In this mode a burst which starts in one data packet is likely to end in the next. The second one will probably be lost. Thus the vertical codewords have to replace the whole lost packet and can not be used to correct the beginning of the burst. So if the data packets for a single picture are transmitted consecutively they are quite susceptible to burst errors. We can not quantify this without quantifying the channel though.

We note that the problem is in the prefix. Since the prefix is used to select the relevant data blocks, it can not be held for off-line decoding very easily. Thus we are stuck with the Hamming code on these bytes. The best protection in the case of consecutive transmission would be to ignore the prefixes after the first. This requires that the decoder know that the data packets are consecutive and also the length of the page. On the other hand, it no longer matters if the bursts run from packet to packet.

CHAPTER 4

OTHER DECODING STRATEGIES

4.1 Introduction

Throughout Chapter 3 we have been considering the performance of a bundle code (14 vertical codewords) decoded by what we have labelled "Full Decoding". Recall that this means that the vertical codewords are either used to replace a missing data block (due to decoding failure in one prefix code) or they are decoded by correcting two bytes if there are exactly two bytes in the vertical codeword with a parity failure or by correcting one byte if there are zero or one parity failures. The Table 1.1 compares this decoding strategy with a number of others. This Chapter will outline the methods used to calculate the numbers in Table 1.1. We are assuming throughout that errors occur independently.

4.2 No Bundle Coding

If no bundle code is used then the only error correction is carried out by the prefix code and the data block code. Assuming independent errors with frequency p (= BER) we can easily write

down formulas for the probability of correct decoding for the prefix code (5 Hamming [8,4] bytes) and either Parity, Single Error Correction or Double Error Correction on the data block. We set $n = 224$ and $q = 1 - p$.

Figure 4.1 Probability of Correct Decoding

Hamming [8,4]	$q^8 + 8pq^7$
Prefix Code	$(q^8 + 8pq^7)^5$
Parity Data Block	q^n
Single E.C. Data Block	$q^n + npq^{n-1}$
Double E.C. Data Block	$q^n + npq^{n-1} + \binom{n}{2} p^2 q^{n-2}$

For comparison with bundle coding we can consider h such data packets transmitted together. Actually for Table 1.1 we considered $h-1$ such data packets as equivalent to a bundle of length h , i.e. we compared units with the same number of "information" blocks. Thus the formulas used for Table 1.1 are as follows in Figure 4.2.

Figure 4.2 Probability of Correcting Decoding - No Bundle Code

No Bundle/Parity	$[(q^8 + 8pq^7)^5 q]^{n, h-1}$
No Bundle/SEC	$[(q^8 + 8pq^7)^5 (q^n + npq^{n-1})]^{h-1}$
No Bundle{DEC	$[(q^8 + 8pq^7)^5 (q^n + npq^{n-1} + \binom{n}{2} p^2 q^{n-2})]^{h-1}$

To translate this into numbers of "bundles" expected before an incorrect decoding we use the estimate $1/(1-P_{CD})$ where P_{CD} is the probability of correct decoding for whatever case we are dealing with.

It is better to calculate the expected number of information bits before an incorrect decoding of a data packet. This removes the need to bunch up a "bundle equivalent" of unbundled data blocks for comparisons sake when that is not how they would be sent down the channel. There are 20 information bits in the prefix and $(28-s)7$ in the data block where $s = 0, 1$ or 2 for Parity, SEC or DEC respectively. So we use the formulas of Figure 4.2 with $h = 2$ (i.e. units of $h-1 = 1$ data packet) and calculate P_{CD} . Then we use as our estimate of the expected number of information bits until an incorrect decoding the formulas:

Parity	$216/(1 - P_{CD})$
SEC	$209/(1 - P_{CD})$
DEC	$202/(1 - P_{CD})$.

4.3 Erasure Correction of One Data Block

One of the weaker procedures for bundle decoding is to use the vertical codewords to replace the data block following a prefix with a detected but uncorrectable error. This is only possible if there

are no other failed prefixes and the data block codes clean out all the other errors from the data blocks. We have mentioned that if this is the type of decoding that is really wanted then the two-byte Reed-Solomon code is not required. It is enough to use a product type check. Of course this sort of decoding will not be as effective against bursts, but if there is a channel in which the error events are dominated by synchronization problems then this might be a sensible strategy. We note though that it is only faster to decode this way if a product-type check block is used. With the code C vertical codewords, there is only a trivial simplification over Full Decoding. Our analysis serves to show that the extra corrections performed in Full Decoding are important (in the case of independent errors).

Given a bundle length h the formulas in Figure 4.2 with the $(h-1)$'s replaced by h 's give the formulas for correct decoding without using the vertical codewords. The probability that a single prefix code fails is $h(1 - P_{PCD})P_{PCD}^{h-1}$ where $P_{PCD} = (q^8 + 8pq^7)^5$ is the probability that a prefix codeword is correctly decoded.

Figure 4.3 Formulas for Probability of Correct Decoding
- Erasure Only

$$\text{Erasure Correction of One Data Block} \left\{ \begin{array}{l} \text{Parity} - P_{\text{PCD}}^h P^h + h(1 - P_{\text{PCD}}) P_{\text{PCD}}^{h-1} P_0^{h-1} \\ \text{SEC} - P_{\text{PCD}}^h P_1^h + h(1 - P_{\text{PCD}}) P_{\text{PCD}}^{h-1} P_1^{h-1} \\ \text{DEC} - P_{\text{PCD}}^h P_2^h + h(1 - P_{\text{PCD}}) P_{\text{PCD}}^{h-1} P_2^{h-1} \end{array} \right.$$

where $P_{\text{PCD}} = (q^8 + 8pq^7)^5$ and P_s is the probability of correct decoding for a data block with s check bytes i.e. Parity for $s = 0$, SEC for $s = 1$ and DEC for $s = 2$. Figure 4.1 gives the formulas for P_s .

The expected number of bundles and of information bits is then calculated as in the case of Full Decoding.

4.4 Single-Byte-Correction of the Vertical Codewords

The vertical codewords of the bundle code could be decoded as single symbol (byte) correcting Reed-Solomon codes while ignoring their erasure correcting capacity and their ability to replace a missing data block if a single prefix is uncorrectable or is miss-synchronized. Once again this is essentially an academic exercise since there is only a trivial reduction in decoder complexity compared to the much more effective Full Decoding.

The calculation of the probability of correct decoding follows the same method as we used in Chapter 3 for Full Decoding. We have P_{PCD} the probability that one prefix is correctly decoded and P_{HV} the probability that the horizontal codes and vertical codes are correctly decoded. We have to calculate P_{HV} by estimating $1 - P_{HV}$ which once again is done by counting the uncorrectable error patterns. This time there are many more uncorrectable error patterns. We have decided to omit the details of the counting process since it is similar to Chapter 3 and the results are of relatively minor importance. (In fact we might not have even done it at all if we had thought of Full Decoding first!)

Figure 4.4 Formulas for the Probability of Correct Decoding -

Single-Byte Correction Only

$$\text{Parity} - P_{PCD}^h [B^{2h} + 2h(1-B)B^{2h-1}]^{14}$$

Single byte correction of each vertical code-word

$$\text{SEC} - P_{PCD}^h P_{HV,1}$$

$$\text{DEC} - P_{PCD}^h P_{HV,2}$$

where $B = q^8$ is the probability that one byte is correctly received and $P_{HV,1}$ and $P_{HV,2}$ are the probabilities that the Horizontal and Vertical codes are correcting decoded with respectively SEC and DEC on the data blocks.

4.5 Full Decoding - the other cases

We have analyzed Full Decoding at great length in its most natural context of double error correction on the data blocks. We might consider the situations of 0 and 1 byte codes on the data blocks as well. The analysis of Full Decoding following single error correction would be quite similar to the work of Chapter 3. but with many more uncorrectable patterns. We have not carried out the requisite counting arguments and hence the asterixes in Table 1.1.

If we were to use Full Decoding after only checking parities in the data blocks then we can write down a formula for the probability of correct decoding. The probability that a byte is received correctly is $B_C = q^8$ and the probability that the byte has a parity failure is $B_F = (8pq^8 + \binom{8}{3}p^3q^5 + \binom{8}{5}p^5q^3 + 8p^7q)$. Then the probability of correct decoding is

$$P_{PCD}^h [B_C^{2h} + 2h(1-B_C)B_C^{2h-1} + \binom{2h}{2}B_F^2B_C^{2h-2}]^{14} + h(1-P_{PCD})P_{PCD}^{h-1} q^{224(h-1)}.$$

CHAPTER 5

TWO-BYTE EXTENSIONS OF THE PRODUCT CODE

5.1 Extensions in General

We are interested in codes of length 28 bytes which have two check bytes one of which is the mod 2 sum of the other 27 bytes (the Product check) and in which all bytes have known parity (say even). What is left is to specify the seven remaining independent bits of the other check byte. The code C is an example of such a code. In fact C is a double bit-error correcting code. Are there any triple error correcting two-byte extensions of the Product code? The next theorem due to James Currie provides a negative answer.

Theorem 5.1. Let H be a two byte double error correcting code which extends the Product code and is 28 bytes long. Then H has at least one codeword of weight six and hence cannot correct all triple error patterns.

Proof. Suppose that four errors occur in a pattern we call P_{jk} ; one error in each of bit 1 of byte 1, bit k of byte 1, bit k of byte j and bit 8 of byte j . Thus we assume that $1 < j \leq 28$ and $1 < k < 8$. The error pattern P_{jk} produces the same Product

syndrome whatever values of j and k are used, namely, (10000001).

We now look at the syndromes produced by the 7 additional checks.

There are 162 patterns P_{jk} since $162 = 6 \times 27$. There are 128 possible 7 bit syndromes. Therefore two of the patterns P_{jk} have the

same syndrome. Then the sum of the two patterns must have zero

Product syndrome and zero syndrome for the additional checks. Thus H

has a codeword which is $P_{jk} + P_{j'k'}$. If $j = j'$ or $k = k'$ then

there are four bits which coincide and the codeword has weight four

contrary to H being double error correcting. Thus $j \neq j'$ and

$k \neq k'$ and the resulting codeword has weight six.

The proof actually shows that we can find a weight six codeword with ones in a predetermined byte (in our case byte 1) and in a predetermined column (in our case column 8) and not in a predetermined column (in our case column 1). Therefore the argument actually shows that there must be a number of weight six codewords. See Appendix B for more calculations.

5.2 Some Extension Ideas that Don't Work

The Product code is a simple code to define and decode. It would be nice if a good extension code could be found which defines its 7 new checks in a similar combinatorial (i.e. non-algebraic)

way. One idea would be to use 8 "diagonal checks". Thus having summed over the rows (bytes) and columns to obtain the Product checks we might now sum over the diagonals. Lets describe the codewords as $(b_{0,0}, b_{0,1}, \dots, b_{27,7})$ where b_{ij} is the j -th bit in the i -th byte and $0 \leq i \leq 27$, $0 \leq j \leq 7$. The diagonal checks are

$$\sum_{(i-j) \equiv m \pmod{8}} b_{ij} = 0 \quad \text{for } m = 0, 1, \dots, 7.$$

For example $b_{00} + b_{11} + \dots + b_{77} + b_{80} + b_{91} + \dots = 0$. This unfortunately does not work. If we set $b_{00} = b_{80} = b_{01} = b_{81} = 1$ we get a codeword of weight 4 so the resulting code is not double error correcting.

In a series of four confusing papers [9], [10], [11], [12] Demetre Voukalis explored a similar but much more powerful code. He uses many more checks and does not wrap the sum around. Thus he uses $\sum_{i-j=m} b_{ij} = 0$ for $m = -7, -6, \dots, 28$. His codes are therefore not useful to us.

The next idea to look at is the possibility of using some "mosaic" pattern other than diagonals to define the new checks. The point is that what we mean by a simple code is one in which the syndromes can be calculated by a few microprocessor operations on each message byte. The diagonal checks can be calculated by cycling the syndrome accumulate once before each exclusive-or. Unfortunately

there are not any double error correcting mosaic codes that we have been able to find. We tried a great variety of shifts etc. and there is always a weight four codeword.

We are interested in byte correcting codes for the bundle coding application. In fact, you can not correct single bytes with fewer than two bytes of checks. A length 8 burst correcting Fire code for example would use 24 check bits (and would actually correct more than single byte errors. Other codes have been described that correct one byte at a time. Kaneda and Fujiwara [13] have outlined a construction for a class of single-byte-error correcting double-byte-error detecting codes. Their impetus is high speed encoding and decoding for application in main memory systems. Their codes use more than two check bytes.

Appendix A. Special Weight 6 Codewords

If $\bar{c} = (c_0, c_1, \dots, c_{223})$ is a codeword of code C then we must have

$$\sum_{i=0}^{223} c_i \alpha^i = 0$$

$$\sum_{k=0}^{27} c_{j+8k} = 0 \quad j = 0, \dots, 7$$

$$\sum_{j=0}^7 c_{j+8k} = 0 \quad k = 0, \dots, 27 .$$

So a codeword of weight six has two ones in each of three bytes, say bytes b_1, b_2, b_3 and these ones are in bits i and j of b_1 and j and k of b_2 and i and k of b_3 for some $i, j, k \in \{0, 1, \dots, 7\}$. The particular codewords of weight 6 required in Chapter 3 were those with two of these bytes separated by 14 bytes so that these two bytes are in the same vertical codeword.

Let us suppose that b_1, b_2 are the bytes separated at a distance of 14 bytes. Then b_3 might precede, separate or follow this pair. In the second two cases we may suppose that b_1 is the first byte and b_2 the 14th byte. We then have to count the patterns found as many times as it is possible to slide the three bytes down the codeword. If b_3 is between b_1 and b_2 then

it is b_2 that will hit the last byte first. Thus every pattern of six ones making a codeword with ones in bytes b_1, b_2, b_3 with $b_1 = 0, b_2 = 11$ and $b_1 < b_3 < b_2$ gives rise to 14 codewords. If $b_1 = 0, b_2 = 14$ and $b_3 > b_2$ then we can produce only $28 - b_3$ codewords from this pattern.

Finally, we count the patterns with b_3 preceding b_1 and b_2 by assuming that $b_1 = 0, b_2 = 14$ and that b_3 is $-1, -2, \dots, -13$ (this makes sense in the computation as we will see below). Such a pattern will have to be shifted over by $|b_3|$ bytes to get all the bytes in the range 0 to 27. This puts the highest numbered byte at $b_2 - b_3$ and then there are $28 - (b_2 - b_3) = 14 + b_3$ codewords coming from the pattern in b_1, b_2, b_3 .

So we take choices for the bit positions as i, j, k where $0 \leq i < j \leq 7$ and $0 \leq k \leq 7$ with $k \neq i, j$. For each such choice of bit positions there are in fact two arrangements. Letting b be the number of the third byte we are looking then for solutions to

$$\alpha^i + \alpha^j + \alpha^{(14)(8)} (\alpha^i + \alpha^k) + \alpha^{8b} (\alpha^j + \alpha^k) = 0$$

and to

$$\alpha^i + \alpha^j + \alpha^{(14)(8)} (\alpha^j + \alpha^k) + \alpha^{8b} (\alpha^i + \alpha^k) = 0$$

where $0 \leq i, j, k \leq 7, i < j$ and $k \neq i, j$ and α is a primitive

element of the field with 128 elements. (Thus α can be taken as a root of $X^7 + X^3 + 1$ for example). Then we allow $-13 \leq b \leq 27$ with $b \neq 0, 14$ and count $14+b$ codewords if $b < 0$, or 14 codewords if $0 < b < 14$ or $28-b$ codewords if $b > 14$.

We find a total of 1063 of these special weight 6 codewords if α is a root of $X^7 + X^3 + 1$. (If we take α as a root of $X^7 + X^6 + X^5 + X^2 + 1$ there are 987. The difference does not influence the calculations.)

APPENDIX B

Extensions of the Product Code: Weight Six Codewords

We have already shown that any 28 byte code H which extends the Product code by adding an additional byte of parity checks must have weight six codewords. In this Appendix we will show that there must be a large number of weight 6 codewords. We assume that H is at least double error correcting and that all bytes have known parity. Thus there are no weight four codewords and only seven independent check bits are introduced by the second check byte. Thus any pattern of errors will produce one of 128 possible syndromes s_i on these seven bits.

Suppose P^{br} is a pattern of three ones (or errors if you prefer) with ones in bit j of byte b and in bits r and j of byte c for some j and c . We must have $b \neq c$ and $r \neq j$ so there are $(27)(7)$ patterns of the type P^{br} . If two patterns of the same type P^{br} have the same syndrome s_i then we add the patterns (i.e. we write all six ones but cancel out any overlaps) the result has zero syndrome and hence is a codeword.

	bit j	bit j'	bit a
byte b	1	*	
byte c	1		1
byte c'		*	*

1 = one from pattern P_1

* = one from pattern P_2 .

If the two patterns overlap then the sum has weight 4 contrary to our assumptions about H . Thus combining two patterns of type P^{br} with some syndrome gives a weight 6 codeword.

Conversely, consider a weight six codeword. It has ones in bytes b_1 , b_2 , and b_3 say and ones in bits i and j of byte b_1 and i and k of b_2 and j and k of b_3 :

	bit i	bit j	bit k
byte b_1	*	*	
byte b_2	*		*
byte b_3		*	*

Now we can obtain this codeword by combining two patterns of weight three errors with the same syndrome s_i and of the same pattern P^{br} . This pattern could be $P^{b,k}$ or P^{b_2j} or P^{b_3i} . Thus each weight six codeword is constructed exactly three times by this method.

Now let K_i^{br} be the number of patterns P^{br} with syndrome s_i . Then there are $(128)(27)(7) = 24192$ parameters K_i^{br} and their sum is the total number of patterns of three ones forming three vertices of a rectangle. We therefore have,

$$\sum_{b,r,i} K_i^b = 4 \binom{28}{2} \binom{8}{2} = 42336$$

$$\sum_{b,r,i} \binom{K_i^{br}}{2} = 3A_6$$

where A_6 is the number of codewords of H of weight 6. Hence

$$A_6 = \frac{1}{6} \left(\sum_{b,r,i} (K_i^{br})^2 - \sum_{b,r,i} K_i^{br} \right).$$

Now given that $\sum K_i^{br} = 42336$ we will minimize the sum of squares by taking $K_i^{br} = 42336 / 24192 = 1.75$. The K_i^{br} are integers though so the minimum comes from taking some to be ones and some to be two.

If there are u ones and v twos then $u + v = 24192$ and $u + 2v = 42336$.

Thus $v = 18144$. But $\binom{1}{2} = 0$ and $\binom{2}{2} = 1$ so we have

$$3A_6 = \sum \binom{K_i^{br}}{2} \geq 18144.$$

Therefore $A_6 \geq 6048$.

This proof is due to Brian Leroux. We note that code C has

$$A_6 = 8600.$$

REFERENCES

1. "Television Broadcast Videotex", Broadcast Specification No. 14, Issue 1, Provisional, Telecommunication Regulatory Service, Department of Communications, Canada, June 19, 1981.
2. P. Allard, V.K. Bhargava and G.E. Seguin, "Realization, Economic and Performance Analysis of Error-correcting Codes and ARQ Systems for Broadcast Telidon and other Videotex Transmission", Department of Communications, Ottawa, DSS Contract No. OSU80-00133, Final Report, June, 1981.
3. G.E. Seguin, P. Allard and V.K. Bhargava, "A Class of High Rate Codes for Byte-oriented Information Systems", I.E.E.E. Trans. Comm. COM-31 (1983).
4. M. Sablatash and J.R. Storey, "Determination of Through puts, Efficiencies and Optimal Block Lengths for an Error Correcting Scheme for the Canadian Broadcast Telidon System", Can. Elec. Eng. J. 5 (1979), 25-39.
5. B.C. Mortimer, "A Correction to a Recent Analysis of a Product Code", Can. Elec. Eng. J., 7 (1982) 40 .

6. B. Leroux, M. Moore, B.C. Mortimer, L. Oattes and T. Ritchford, "A Study of the Use of Error Correcting Codes in the Canadian Broadcast Telidon System", Department of Communications, Ottawa, Progress Report, DSS Contract No. OSU81-00095, August, 1981.
7. Brian Mortimer, "Error-correcting Codes for the Canadian Broadcast Telidon System", Department of Communications, Ottawa, Final Report, DSS Contract No. OSU81-00095, February, 1982.
8. B.C. Mortimer and M.J. Moore, "Two Byte Data Block and Bundle Codes for the Broadcast Telidon System", Department of Communications, Ottawa, Progress Report, DSS Contract No. OSU82-00165, November, 1982.
9. D.C. Voukalis, "Generalization of the Column and Diagonal Matrix Burst Correcting Codes", Int. J. Electronics 47 (1979), 193-200.
10. D.C. Voukalis, "A New Concatenated Matrix Code with Cluster, Burst and Random Error Correcting Abilities", Int. J. Electronics 49 (1980), 345-357.
11. D.C. Voukalis, "Error Statistics and Protocol Construction Using General Column and Diagonal Matrix Burst Correcting Codes", Int. J. Electronics 48 (1980), 337-340.
12. D.C. Voukalis, "A Comparison Between Burst Error Correcting Codes", Int. J. Electronics 49 (1980), 319-325.

13. S. Kaneda and E. Fujiwara, "Single Byte Error Correcting - Double Byte Error Detecting Codes for Memory Systems", IEEE Int. Sym. Fault Tolerant Computing, Kyoto, 1980, 41-46.
14. E.R. Berlekamp, "Bit-Serial Reed-Solomon Encoders", IEEE Trans. Info. IT-28 (1982), 869-874.