

**Loop transversal codes over finite rings**

by

Ruben Hranti Aydinyan

A dissertation submitted to the graduate faculty  
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Mathematics

Program of Study Committee:  
Jonathan D. H. Smith, Major Professor  
Clifford Bergman  
Wolfgang Kliemann  
Ling Long  
Glenn Luecke

Iowa State University

Ames, Iowa

2005

Graduate College  
Iowa State University

This is to certify that the doctoral dissertation of  
Ruben Hranti Aydinyan  
has met the dissertation requirements of Iowa State University

---

Committee Member

---

Committee Member

---

Committee Member

---

Committee Member

---

Major Professor

---

For the Major Program

## TABLE OF CONTENTS

<b>INTRODUCTION</b> . . . . .	1
<b>CHAPTER I. ERROR-CORRECTING CODES</b> . . . . .	3
1.1 Basic concepts in coding theory . . . . .	3
1.2 Linear codes . . . . .	7
1.3 Polynomial rings and cyclic codes . . . . .	10
1.4 Nonlinear codes . . . . .	14
1.5 Lexicodes . . . . .	15
<b>CHAPTER II. QUATERNARY CODES</b> . . . . .	16
2.1 Hensel's lemma and the Hensel lift . . . . .	16
2.2 Cyclic codes over $\mathbb{Z}_4$ . . . . .	21
2.3 The Gray map and images of $\mathbb{Z}_4$ -codes . . . . .	23
<b>CHAPTER III. LOOP TRANSVERSAL CODES</b> . . . . .	26
3.1 General construction . . . . .	26
3.2 Greedy approach . . . . .	28
<b>CHAPTER IV. NEW RESULTS ON LOOP TRANSVERSAL CODES</b> . . . . .	31
4.1 Generator matrix . . . . .	31
4.2 Error detection in loop transversal codes . . . . .	34
4.3 Minimum syndrome weight and $m$ -independence . . . . .	35
4.4 Nonlinear images of LT codes . . . . .	38
<b>BIBLIOGRAPHY</b> . . . . .	40
<b>APPENDIX A. TABLES FOR THE DIMENSIONS OF GREEDY LT CODES</b>	43

**APPENDIX B. FORTRAN SOURCE CODE FOR THE GREEDY LOOP  
TRANSVERSAL ALGORITHM . . . . . 48**

## INTRODUCTION

This paper discusses a class of codes known as loop transversal or LT codes introduced in [16]. These are linear codes constructed with attention to a linear function known as the syndrome. In particular, if this function is constructed by a greedy algorithm [11], then the code constructed is called a *greedy loop transversal* code. Greedy LT codes are similar to the lexicodes constructed in [6] and [7], or more generally to the greedy codes constructed in [2], in the sense that they search greedily through sets in an effort to construct optimal codes. In contrast with lexicodes LT codes are always linear, whereas the lexicodes are linear only in the binary case. A comprehensive study of the binary greedy LT codes has appeared in the papers [11, 12, 13]. In this paper a generalization of greedy LT codes to the case of arbitrary finite rings is presented, with especial attention devoted to the ring of integers modulo 4.

In the first chapter the basic concepts and definitions of coding theory are given. The Hensel lift, a useful tool in the construction of nonlinear binary codes by taking quaternary cyclic codes and projecting onto  $\mathbb{Z}_2$  using the Gray map, is discussed in Chapter II. Relationships between polynomial rings and cyclic codes are given in Section 1.3 for the binary channel, and continued in Sections 2.1 and 2.2 for the quaternary case. In Chapter III the general construction of LT codes proposed in [16] and the greedy construction given in [11, 12] are presented. Chapter IV summarizes major results obtained through this research. We first present an algorithm to compute a generator matrix of an LT code in Section 4.1, then discuss a way to extend the LT codes having odd minimum distance in Section 4.2. Section 4.3 discusses some algebraic tools used to speed up the syndrome computations, and Section 4.4 presents the results related to the quaternary LT codes and their binary images under the Gray map defined in Section 2.3. The tables for the dimensions of LT codes for  $q = 3, 4, 5,$  and  $7$  are given in Appendix A.

Comparisons with the best known linear codes (if available) for  $q = 3, 5,$  and  $7$  are given in the parenthesis next to each dimension in the tables. Appendix B concludes the paper with the FORTRAN source code implemented to calculate the data in the tables for  $q = 3, 5,$  and  $7$  in Appendix A. The fortran source code implemented to produce the results related to the quaternary LT codes was too long to be included in this paper.

## CHAPTER I. ERROR-CORRECTING CODES

### 1.1 Basic concepts in coding theory

Error-correcting codes have been extensively studied throughout the last five decades. The subject of reliable communication is in the core of this research. During transmission or storage, information is subject to corruption due to interference or the time factor, and means of recovery of the original information are being devised. The idea behind this is to add some extra information bits to the original information so that it will be possible to recover the information sent (or stored) after transmission (elapse of time). A typical simplified diagram of this process is given in Figure 1.

In an ideal system the word coming out of the channel should match the word that entered the channel. In a practical system there are occasional errors, and it is the purpose of codes to detect and possibly correct such errors. Much of coding theory has been based on the assumption that each symbol is affected by the noise independently of others, so that the probability of a given error pattern depends only on the number of errors. For example codes have been designed to correct any pattern of  $t$  or less errors in a block of length  $n$ . While error

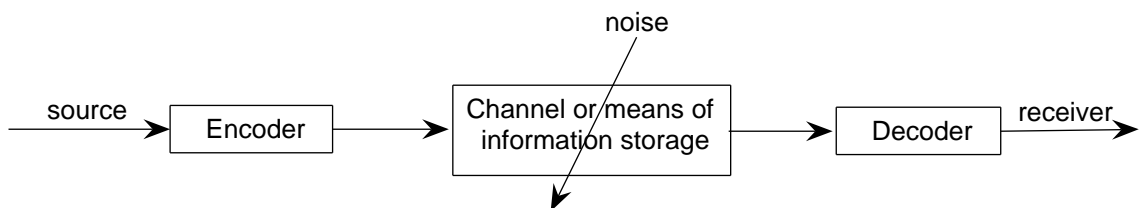


Figure 1 A typical scheme for a digital network

correction is essential for certain communication schemes (e.g. space communication), error detection and retransmission may be more effective in various practical systems.

A continuous sequence of information digits are entered into the encoder. At its output another sequence is produced with additional digits, which is fed to the modulator of the channel. On the other end the decoder accepts a sequence of channel symbols and translates it into a shorter sequence of information digits. The code that is being employed determines the rules of operation for the encoder and decoder.

There exist two fundamentally different types of codes. The *block codes* are designed so that the encoder breaks the continuous sequence of information digits into  $k$ -symbol blocks. Then each block is being processed independently. An  $n$ -tuple is associated with each block, where  $n > k$ . The resulting  $n$ -tuple, called a *codeword* is transmitted, possibly corrupted, and decoded independently of other codewords.

The *Hamming distance* between two words of the same length is defined to be the number of positions in which the two words differ. Thus if a single error occurred then the Hamming distance between the transmitted and received words is one.

At the receiver a decision based on the received  $n$ -tuple is made. The decoder decides which codeword has been transmitted. Most of the decoding schemes are based on the maximum likelihood decoding, i.e. the closest codeword to the received word is assumed to be the codeword transmitted. Of course, this does not exclude a possibility of a wrong decision, but with “good” codes the probability of a right decision is much larger than the probability of erroneous decoding.

One of the simplest decoding schemes is table look-up decoding. A table can be made so that the codewords make up the first row of the table. If a codeword is received, it is reasonable to assume that the received codeword coincides with the transmitted codeword. Then one may specify the decision of the receiver for each of the possible received words by listing under each codeword the words that would be decoded into it.

**Example.** Suppose we want to transmit one of the four messages  $a = 00000$ ,  $b = 00111$ ,  $c = 11100$ , and  $d = 11011$ , using a binary block code of length five. Table 1.1 specifies

Codewords	<u>00000</u>	<u>00111</u>	<u>11100</u>	<u>11011</u>
	00001	00110	11101	11010
Other	00010	00101	11110	11001
words	00100	00011	11000	11111
	01000	01111	10100	10011
	10000	10111	01100	01011
Words with errors that can be	01001	01110	01101	10001
detected but not corrected	01010	10110	10101	10010

Table 1.1 Decoding table for a binary code with  $n = 5$  and  $k = 2$ .

the decisions of the receiver. This code can correct any single error. Note that there are words that cannot be corrected as they are equidistant from at least two codewords. For example if the word 10110 is received then it is not possible to determine if the transmitted codeword was 00111 or 11100. In such cases the decoder may announce an error detection but will not make any further effort to correct it.

The other type of codes, called *tree codes*, operate by breaking the continuous information sequence into rather dependent small tuples. Based on the current tuple and the preceding information symbols the encoder emits a longer tuple which merges into the continuous code sequence. Convolutional codes are example of tree codes.

Both types of codes share similar capabilities and limitations. In particular, Shannon's fundamental theorem stating that using suitable codes it is possible to transmit information at any rate less than the channel capacity holds for both types.

For a finite number  $q > 1$ , a  $q$ -ary *alphabet*  $Q$  is a set of  $q$  distinct symbols. For the sake of simplicity we use  $Q = \{0, 1, \dots, q - 1\}$ . A subset  $C \subset Q^n$  is called a  $q$ -ary *block code of length  $n$* , or just a  $q$ -ary *code of length  $n$* . The elements of  $Q^n$  are called *words* of length  $n$ , and the elements of  $C$  are called *codewords*. The *size*  $M$  of the code  $C$  is the cardinality of  $C$ , i.e.  $M = |C|$ . If  $q = p^k$ , where  $p$  is prime, then  $Q$  may be viewed as the Galois field  $GF(p^k)$ , and  $Q^n$  will be a vector space. If  $Q$  is a ring, then  $Q^n$  is a module.

The *Lee weight* of a word  $v = (a_{n-1}, \dots, a_1, a_0)$ , for  $a_i \in Q$ , is defined to be

$$wt_L(v) = \sum_{i=0}^{n-1} |a_i|$$

$$\text{where } |a_i| = \begin{cases} a_i & 0 \leq a_i \leq \frac{q}{2} \\ q - a_i & \frac{q}{2} < a_i \leq q - 1. \end{cases}$$

The *Lee distance* between two words is defined to be the Lee weight of their difference. For  $q = 2$  and  $3$  the Hamming and Lee distances coincide. For  $q > 3$ , the Lee distance is greater than or equal to the Hamming distance between two words.

A *metric* is defined as a real-valued function satisfying the following three properties:

- i.  $d(x, y) = 0$  iff  $x = y$  (reflexive)
- ii.  $d(x, y) = d(y, x)$  (symmetric)
- iii.  $d(x, y) \leq d(y, z) + d(x, z)$  (triangle inequality)

It is not difficult to verify that both Hamming and Lee distances are metrics.

It is possible to detect all patterns of  $t$  errors if and only if the minimum distance between codewords is  $2t$ . For if the minimum distance is  $2t$  then no pattern of  $t$  errors can change one codeword into another, while if the minimum distance is less than  $2t$  there is a pattern of fewer than  $t$  errors that will carry one codeword into another. It is possible to correct all patterns of  $t$  or fewer errors if and only if the minimum distance between codewords is at least  $2t + 1$ . For any received word with  $t' \leq t$  errors is at a distance  $t'$  from the transmitted codeword, and the distance between the received word and any other codeword is at least  $2t + 1 - t' > t'$ . Conversely, if the minimum distance is less than  $2t + 1$ , then occurrence of  $t$  errors results in a received word at least as close to an incorrect codeword as to the transmitted codeword.

A similar argument shows that to correct  $t$  errors and detect  $d \geq t$  errors it is necessary and sufficient to have a minimum distance  $t + d + 1$ .

The *rate* of a code of length  $n$  over  $Q$  is the logarithm to base  $q$  of the number of the codewords  $M$  divided by  $n$ .



or equivalently

$$G' = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

is a generator matrix for this code. Both  $G$  and  $G'$  generate the same code. The matrix  $G$  is obtained by the Reed-Muller construction, whereas the matrix  $G'$  is obtained by the greedy loop transversal algorithm which will be discussed in chapter III. The code  $C$  generated by each of these matrices has length 16 (the number of columns), dimension 5 (the number of rows), and minimum distance  $8 = 3 + 4 + 1$ . It can correct three and detect four random errors. Its size is  $M = 2^5 = 32$  (number of codewords), and its rate is  $\frac{\log_2 32}{16} = \frac{5}{16}$ .

More generally, a Reed-Muller code of order  $r$ , denoted by  $RM(r, m)$ , is a linear code with parameters

$$\begin{aligned} n &= 2^m, \quad m > 0, \\ k &= 1 + \binom{m}{1} + \cdots + \binom{m}{r} \\ d &= 2^{m-r}, \end{aligned}$$

where  $r$  is an integer in the range  $0 \leq r \leq m$ . The code in this example is  $RM(1, 4)$ .

An alternative description of a linear code is by specifying a generator matrix for its null space. Again, if  $C$  is a subspace of dimension  $k$  then its null space  $C^\perp$  has dimension  $n - k$ . Thus a matrix  $H$  of rank  $n - k$  with basis vectors of  $C^\perp$  as its rows will completely describe the code. For a vector  $v$  is a codeword if and only if it is orthogonal to every row of  $H$ , i.e.  $vH^T = 0$ . The matrix  $H$  is called a *parity-check matrix* of  $C$ . As  $C^\perp$  is also a subalgebra of  $Q^n$ , it is a linear code as well. The codes  $C$  and  $C^\perp$  are called *dual codes*.

**Example.** Consider the binary  $[7, 4, 3]$  Hamming code. Its parity-check matrix  $H$  has all nonzero binary representations of integers  $1, 2, \dots, n = 7 = 2^3 - 1$  as its columns.

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

The vector  $v = 1110000$  is a codeword since  $vH^T = 0$ . The generator matrix for this code  $C$  is

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The code generated by  $H$  is the dual  $C^\perp$  of  $C$ .

In general, for a binary Hamming code that has a dimension  $k = n - m$ , the parity-check matrix  $H$  has  $m$  rows and  $n = 2^m - 1$  columns, where the columns are all possible nonzero  $m$ -tuples. Since all columns of  $H$  are different then no two of them would add up to 0. This implies<sup>1</sup> that the null space of  $H$  has a minimum weight 3 and is capable of correcting all single errors.

If a code vector  $v$  is transmitted and a single error occurs, the received vector is  $u = v + e_i$ , where  $e_i$  is a vector that has a 1 in the  $i$ -th<sup>2</sup> position,  $i = 1, 2, \dots, n$ , and 0's everywhere else. Then

$$uH^T = (v + e_i)H^T = vH^T + e_iH^T = e_iH^T$$

since  $v$  is a codeword and it must be in the null space of  $H$ . But  $e_iH^T$  is just the  $i$ -th column of  $H$ . Thus if the product  $uH^T$  is the zero vector then no error occurred during the transmission, otherwise it must be one of the columns of  $H$  and hence there was an error in the corresponding position of  $u$  that may be easily corrected. The vector  $uH^T$  is called the *syndrome* of  $u$ .

A binary code with even minimum distance can be formed by adding a *parity-check symbol* to all the codewords of a code with odd minimum distance, thus making the code length one

<sup>1</sup>This implication is not trivial and will become clear in chapter IV

<sup>2</sup>This is the position where the error has occurred

more. The parity-check symbol or the *overall parity-check symbol*, say  $c_n$ , for a codeword  $c = (c_0, c_1, \dots, c_{n-1})$  is defined to be  $q - [wt(c) \pmod{q}]$ , where  $q$  is the size of the alphabet and  $wt(c)$  is either the Hamming or Lee weight of  $c$ . In fact, adding a parity-check symbol to any binary code will always add a 0 to all codewords of even weight and 1 to all codewords of odd weight. Therefore, if the minimum weight in a binary linear code is odd, the overall parity-check increases the minimum weight by 1. The codes having an overall parity-check symbol added are called *extended codes*. Thus the extended  $[8, 4, 4]$  Hamming code may be obtained from the  $[7, 4, 3]$  Hamming code by adding a parity-check symbol. For example, if we decide the last symbol to be the parity-check symbol, then the generating matrix  $G$  from the previous example will become

$$G' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

### 1.3 Polynomial rings and cyclic codes

There is a special class of linear codes, called *cyclic*, defined by the condition of invariance under cyclic shifts. We will say that the vector  $(c_{n-2}, c_{n-3}, \dots, c_0, c_{n-1})$  is a *cyclic shift* of the vector  $(c_{n-1}, c_{n-2}, \dots, c_0)$ . Then a code (or a subspace) is *cyclic* if for every codeword all of its cyclic shifts are also codewords.

Let  $\mathbb{F}$  be a field. Then the ring of polynomials  $\mathbb{F}[x]$  is a Euclidean domain<sup>3</sup>, and hence for each pair of polynomials  $a(x), b(x) \in \mathbb{F}[x]$  such that  $b(x) \neq 0$  there exists a unique pair  $q(x)$  and  $r(x)$  in  $\mathbb{F}[x]$  such that

$$a(x) = q(x)b(x) + r(x), \quad \text{where } r(x) = 0 \text{ or } \text{degree } r(x) < \text{degree } b(x).$$

---

<sup>3</sup>The proof of this fact may be found in any abstract algebra book, for example in [9]

Now, if  $p(x)$  is a polynomial of degree  $n \geq 1$ , let  $(p(x))$  be the principal ideal generated by  $p(x)$ . Then the quotient ring, the ring of residue classes modulo  $p(x)$ , is the ring of all polynomials of degree less than  $n$ . In fact, the quotient ring is isomorphic to the  $n$  dimensional vector space  $\mathbb{F}^n$  via the bijection

$$\begin{aligned} \mathbb{F}/(p(x)) &\rightarrow \mathbb{F}^n; \\ f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_0 &\mapsto (a_{n-1}, a_{n-2}, \dots, a_0). \end{aligned} \quad (1.1)$$

Hence, the equivalence classes of polynomials in  $\mathbb{F}/(p(x))$  can be identified with their corresponding vectors in  $\mathbb{F}^n$ . These descriptions will be used interchangeably in the rest of this thesis.

Thus for each polynomial  $g(x) \in \mathbb{F}[x]$  there exists a unique residue class representative  $\bar{g}(x) \in \mathbb{F}/(p(x))$  such that

$$g(x) = q(x)p(x) + \bar{g}(x), \text{ with degree } \bar{g}(x) < n.$$

If  $p(x) = x^n - 1$ , then  $x^n - 1 = 0$ , i.e.  $x^n = 1$  in  $\mathbb{F}/(x^n - 1)$ . Then multiplication by  $x$  is equivalent to a cyclic shift, for

$$\begin{aligned} xf(x) = a_{n-1}x^n + a_{n-2}x^{n-1} + \dots + a_0x &= a_{n-2}x^{n-1} + \dots + a_0x + a_{n-1} \\ (a_{n-1}, a_{n-2}, \dots, a_0) &\mapsto (a_{n-2}, \dots, a_0, a_{n-1}). \end{aligned} \quad (1.2)$$

An immediate consequence of (1.2) is the next proposition.

**Proposition 1.1.** *Every ideal in  $\mathbb{F}[x]/(x^n - 1)$  is isomorphic to a cyclic code via the bijection given in (1.1).*

□

The next several propositions state some well known facts and some properties of binary codes and will be stated here without proof.

**Proposition 1.2.** *Every ideal  $I$  of  $\mathbb{F}_2[x]/(x^n - 1)$  is principal. Moreover,  $I$  is generated by the polynomial  $g(x) \in I$  of least degree, and if  $g(x) \neq 0$ , then  $g(x)$  divides  $x^n - 1$  in  $\mathbb{F}_2[x]$ .*

□

The polynomial  $g(x)$  in Proposition 1.2 is called the *generator polynomial* of  $I$ . Since  $I$  is isomorphic to some cyclic code  $C$ , from now on a cyclic code will be identified with its corresponding ideal, and the generating polynomial of this ideal will be called the *generating polynomial of the code*. Since  $g(x)$  divides  $x^n - 1$ , the polynomial  $h(x) = \frac{x^n - 1}{g(x)}$  is also a divisor of  $x^n - 1$ . Thus it seems reasonable to seek a relationship between the codes generated by  $g(x)$  and  $h(x)$ . Let  $k$  be the degree of  $g(x)$ . Then  $h(x)$  has a degree  $n - k$ . Define the *reciprocal polynomial* to  $h(x)$  as

$$\tilde{h}(x) = x^{n-k} h\left(\frac{1}{x}\right).$$

Note that if  $h(x) = h_{n-k}x^{n-k} + \dots + h_1x + h_0$ , then

$$\tilde{h}(x) = x^{n-k} h\left(\frac{1}{x}\right) = x^{n-k} \left( h_{n-k} \frac{1}{x^{n-k}} + \dots + h_1 \frac{1}{x} + h_0 \right) = h_{n-k} + \dots + h_1 x^{n-k-1} + h_0 x^{n-k}$$

has the same coefficients with the reversed order. The following proposition establishes the relationship between the codes generated by  $\tilde{h}(x)$  and  $g(x)$ .

**Proposition 1.3.** *If  $g(x)$  is the generating polynomial of a cyclic code  $C$ , then  $\tilde{h}(x)$  is the generating polynomial of  $C^\perp$ , the dual code of  $C$ . Moreover,  $C^\perp$  is also cyclic.*

□

**Proposition 1.4.** *Let  $g(x)$  be a nontrivial divisor of  $x^n - 1$ . Then  $(g(x))$  is a prime ideal of  $\mathbb{F}_2/(x^n - 1)$  if and only if  $g(x)$  is an irreducible factor of  $x^n - 1$  in  $\mathbb{F}_2[x]$ . Moreover, every prime ideal of  $\mathbb{F}_2[x]/(x^n - 1)$  is maximal.*

□

Thus every irreducible factor of  $x^n - 1$  will generate a maximal ideal in  $\mathbb{F}_2[x]/(x^n - 1)$ . More generally, if

$$p(x) = g_1(x)^{k_1} g_2(x)^{k_2} \dots g_s(x)^{k_s}$$

is the factorization of  $p(x)$  into irreducibles in  $\mathbb{F}[x]$ , where the  $g_i(x)$  are distinct, then

$$\mathbb{F}/(p(x)) \cong \mathbb{F}[x]/(g_1(x)^{k_1}) \times \mathbb{F}[x]/(g_2(x)^{k_2}) \times \dots \times \mathbb{F}[x]/(g_s(x)^{k_s})$$

This is a consequence of the Chinese Remainder Theorem, since the ideals  $(g_i(x)^{k_i})$  and  $(g_j(x)^{k_j})$  are comaximal<sup>4</sup> if  $g_i(x)$  and  $g_j(x)$  are distinct.

**Example.** Consider the factorization of  $x^7 - 1$  in  $GF(2)$ .

$$x^7 - 1 = (x - 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

$GF(2)/(x^7 - 1)$  is a 7 dimensional vector space, isomorphic to the product  $GF(2)/(x - 1) \times GF(2)/(x^3 + x + 1) \times GF(2)/(x^3 + x^2 + 1)$  of one 1 and two 3 dimensional vector spaces.

The polynomial  $g(x) = x^3 + x + 1$  generates a cyclic  $[7,4,3]$  code. The polynomial  $h(x) = (x - 1)(x^3 + x^2 + 1) = x^4 + x^2 + x + 1$ , and hence  $\tilde{h}(x) = x^4 h(\frac{1}{x}) = x^4(\frac{1}{x^4} + \frac{1}{x^2} + \frac{1}{x} + 1) = x^4 + x^3 + x^2 + 1$ . Then  $\tilde{h}(x)$  generates the dual code  $C^\perp$ .

The elements

$$\begin{aligned} \{g(x)\} &= (0001011) \\ \{xg(x)\} &= (0010110) \\ \{x^2g(x)\} &= (0101100) \\ \{x^3g(x)\} &= (1011000) \end{aligned}$$

can be taken as basis vectors. Therefore, the matrix

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

can be taken as the generator matrix. Similarly, the elements

$$\begin{aligned} \{\tilde{h}(x)\} &= (0011101) \\ \{x\tilde{h}(x)\} &= (0111010) \\ \{x^2\tilde{h}(x)\} &= (1110100) \end{aligned}$$

---

<sup>4</sup>The term *comaximal* is used here without a prior definition. It appeared more convenient to define it in section 2.1 (see Definition 2.2).

can be taken as basis for the null space, and the matrix

$$H = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

can be taken as the generating matrix of the dual code  $C^\perp$  or the parity-check matrix for  $C$ . Notice, that  $GH^T = 0$ .

Two codes are called (*permutationally*) *equivalent* if the generator (equivalently parity-check) matrix for one code may be obtained by permuting the columns of the generator (parity-check) matrix of the other. The code  $C$  in the previous example is equivalent to the Hamming  $[7, 4, 3]$  code since the matrix  $H$  has all possible nonzero binary combinations of length 3 as its columns.

## 1.4 Nonlinear codes

An important aspect of nonlinear codes is that for almost every code length and fixed minimum distance there exists a nonlinear code which has more codewords than any linear code with the same parameters. Non-binary lexicode and codes derived from Hadamard matrices are some examples of nonlinear codes. Although nonlinear codes have more codewords, their implementation is usually very complex, and in the absence of a linear structure it is not clear if they will have a practical significance. Many of the nonlinear codes do not have much of a mathematical structure, which makes the process of encoding and decoding very hard. A few of them were recently discovered to have a nice linear structure over the ring of integers modulo 4. The nonlinear codes found by Nordstrom-Robinson, Kerdock, Preparata, Goethals, and Delsarte-Goethals were shown to be the binary images of linear codes over  $\mathbb{Z}_4$  under the Gray map [3], [4], [10]. The Gray map will be defined in chapter II where the nonlinear images of quaternary cyclic codes will be considered.

## 1.5 Lexicodes

Lexicographic codes, or *lexicodes* for short, are greedily constructed codes which were introduced by Conway and Sloane in [6], [7]. Lexicodes appear to have good coding parameters. In many cases these codes are optimal. They represent codes that are linear in the binary channel and not necessarily linear otherwise. In the binary case lexicodes coincide with the loop transversal codes [11]. A generalization of lexicodes, called *greedy codes*, were constructed by Brualdi and Pless in [2].

In the construction of lexicodes the minimum distance is predetermined, say  $d$ . The elements of the channel  $Q^n$  are arranged in a lexicographic order, starting with  $\underbrace{00\dots 0}_n$ , then  $\underbrace{00\dots 1}_n$ , etc. The all-zero vector is the first one admitted into the code. Then a new word is allowed if it has a Hamming distance  $d$  from all the previously admitted codewords.

**Example.** The following are the 12 codewords of the quaternary lexicode (over  $\mathbb{Z}_4$ ) with code length  $n = 4$  and minimum distance  $d = 3$ :

0000	0333	1201	2130
0111	1012	1320	2210
0222	1103	2021	2302

This code is not linear, because for example  $1012 \oplus_4 1103 = 2111$  is not a codeword although both 1012 and 1103 are.

## CHAPTER II. QUATERNARY CODES

Quaternary codes have become of major interest with the discovery that several nonlinear codes, such as those found by Nordstrom-Robinson, Kerdock, Preparata, Goethals, and Delsarte-Goethals, are the binary images of linear codes over  $\mathbb{Z}_4$  under the Gray map [3], [4], [10]. The necessary background related to quaternary codes is given in this chapter. Note that  $\mathbb{Z}_4$  is not an integral domain. Thus special care is needed for working in this ring. In particular, we do not have the unique factorization property, and hence some new tools need to be devised to work with polynomials and the ideals generated by these polynomials over this ring. Our main result related to the quaternary codes will be presented in chapter IV. A comprehensive study of quaternary codes may be found in [20], and a nice review in [14]. In this thesis only the bare minimum will be presented so that the reader can understand the nature of these codes.

### 2.1 Hensel's lemma and the Hensel lift

In this section Hensel's lemma, an important tool for studying quaternary codes, is introduced. Although the Hensel's lemma generalizes to Galois rings, the focus in this thesis will be restricted to the special case, the polynomials over  $\mathbb{Z}_4$ . The Hensel lift, a tool to obtain basic irreducible polynomials over  $\mathbb{Z}_4$  from irreducible polynomials over  $\mathbb{Z}_2$ , will be used in the construction of cyclic codes over  $\mathbb{Z}_4$ .

Let  $\mathbb{Z}_4[x]$  be the polynomial ring over  $\mathbb{Z}_4$ . Define the map

$$\phi : \mathbb{Z}_4 \rightarrow \mathbb{Z}_2; 0, 2 \mapsto 0; 1, 3 \mapsto 1.$$

It is not hard to verify that  $\phi$  is a ring homomorphism. Rather than writing  $\phi(x)$  every time we will use bar to denote the image of an element of  $\mathbb{Z}_4$  under  $\phi$ , i.e. for  $x \in \mathbb{Z}_4$ ,  $\bar{x} = \phi(x)$ . In fact, applying  $\phi$  is equivalent to reduction modulo 2.

Thus  $\bar{0} = \bar{2} = 0$ , and  $\bar{1} = \bar{3} = 1$ . The map  $\phi$  extends naturally to a map

$$\phi : \mathbb{Z}_4[x] \rightarrow \mathbb{Z}_2[x]; \tag{2.1}$$

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_0 \mapsto \bar{f}(x) = \bar{a}_{n-1}x^{n-1} + \bar{a}_{n-2}x^{n-2} + \dots + \bar{a}_0.$$

This extended map is also a ring homomorphism with kernel

$$(2) = 2\mathbb{Z}_4[x] = \{2f(x) \mid f(x) \in \mathbb{Z}_4[x]\}.$$

Let  $R$  be a commutative ring with identity.

**Definition 2.1.** *Two polynomials  $f_1(x)$  and  $f_2(x)$  in  $R[x]$  are said to be coprime if they have no common divisor of degree  $\geq 1$ , or equivalently if there exist  $\lambda_1(x), \lambda_2(x) \in R[x]$  such that*

$$\lambda_1(x)f_1(x) + \lambda_2(x)f_2(x) = 1.$$

**Definition 2.2.** *Two ideals  $I$  and  $J$  of  $R$  are said to be comaximal if  $I + J = R$ . Where  $I + J = \{a + b \mid a \in I, b \in J\}$ . More generally, the ideals  $I_1, I_2, \dots, I_s$  of  $R$  are said to be comaximal if  $\sum_{i=1}^s I_i = R$ .*

Thus in the view of the previous definitions, coprime polynomials generate comaximal ideals.

**Proposition 2.1.** *Two ideals  $(f(x))$  and  $(g(x))$  of  $R[x]$  are comaximal if and only if the polynomials  $f(x)$  and  $g(x)$  are coprime.*

*Proof.* If the ideals  $(f(x))$  and  $(g(x))$  are comaximal, then for every polynomial  $r(x) \in R[x]$  there is a pair of polynomials  $a(x), b(x) \in R[x]$  such that

$$r(x) = a(x)f(x) + b(x)g(x). \tag{2.2}$$

In particular, (2.2) is true for  $r(x) = 1$ . Conversely, if  $f(x)$  and  $g(x)$  are coprime then there exist  $\lambda_1(x), \lambda_2(x) \in R[x]$  such that

$$\lambda_1(x)f(x) + \lambda_2(x)g(x) = 1. \quad (2.3)$$

Then for each  $r(x) \in R[x]$  we can multiply both sides of (2.3) by  $r(x)$  then substitute  $a(x) = \lambda_1 r(x)$  and  $b(x) = \lambda_2 r(x)$  to get  $r(x) = a(x)f(x) + b(x)g(x) \in (f(x)) + (g(x))$ .

□

We next derive some properties for polynomials over  $\mathbb{Z}_4$  in relation with polynomials over  $\mathbb{Z}_2$  using the ring homomorphism  $\phi : \mathbb{Z}_4[x] \rightarrow \mathbb{Z}_2[x]$  defined in (2.1).

**Lemma 2.2.** *Two polynomials  $f_1(x)$  and  $f_2(x)$  are coprime in  $\mathbb{Z}_4[x]$  if and only if their images  $\overline{f_1}(x)$  and  $\overline{f_2}(x)$  are coprime in  $\mathbb{Z}_2[x]$ .*

*Proof.* One direction is simple. If  $f_1(x), f_2(x) \in \mathbb{Z}_4[x]$  are coprime, then there are  $\lambda_1(x), \lambda_2(x) \in \mathbb{Z}_4[x]$  such that

$$\lambda_1(x)f_1(x) + \lambda_2(x)f_2(x) = 1. \quad (2.4)$$

Then application of  $\phi$  to (2.4) will yield a valid equation in  $\mathbb{Z}_2$

$$\overline{\lambda_1}(x)\overline{f_1}(x) + \overline{\lambda_2}(x)\overline{f_2}(x) = \overline{1} = 1.$$

Now, assume  $\overline{f_1}(x), \overline{f_2}(x) \in \mathbb{Z}_2[x]$  are coprime. Then there exist  $\lambda_1(x), \lambda_2(x) \in \mathbb{Z}_4[x]$  such that

$$\overline{\lambda_1}(x)\overline{f_1}(x) + \overline{\lambda_2}(x)\overline{f_2}(x) = 1.$$

Thus

$$\lambda_1(x)f_1(x) + \lambda_2(x)f_2(x) = 1 + 2k(x), \quad (2.5)$$

for some  $k(x) \in \mathbb{Z}_4[x]$ . Multiplication of both sides of the equation (2.5) by  $2k(x)$  yields

$$2k(x)\lambda_1(x)f_1(x) + 2k(x)\lambda_2(x)f_2(x) = 2k(x). \quad (2.6)$$

Substitute (2.6) into (2.5) to get

$$[1 - 2k(x)]\lambda_1(x)f_1(x) + [1 - 2k(x)]\lambda_2(x)f_2(x) = 1.$$

Hence  $f_1(x)$  and  $f_2(x)$  are coprime in  $\mathbb{Z}_4[x]$ .

□

Next we state the Hensel's lemma which proves the existence of a polynomial factorization into pairwise coprime factors over  $\mathbb{Z}_4$  if there is one over  $\mathbb{Z}_2$ . We will omit the proof of this lemma here as it may be found in Section 5.1 in [20].

**Lemma 2.3 (Hensel).** *Let  $f(x)$  be a monic polynomial in  $\mathbb{Z}_2[x]$  with  $f(x) = f_1(x)f_2(x)\dots f_r(x)$ , where  $f_1(x), f_2(x), \dots, f_r(x)$  are pairwise coprime in  $\mathbb{Z}_2[x]$ . Then there exist pairwise coprime monic polynomials  $g_1(x), g_2(x), \dots, g_r(x) \in \mathbb{Z}_4[x]$  such that  $\bar{g}_i(x) = f_i(x)$ ,  $\deg g_i(x) = \deg f_i(x)$ ,  $i = 1, 2, \dots, r$ , and if  $g(x) = g_1(x)g_2(x)\dots g_r(x)$  then  $\bar{g}(x) = f(x)$ .*

□

We now use Hensel's lemma to show that if  $n$  is odd then for any divisor of  $x^n - 1$  in  $\mathbb{Z}_2[x]$  there exists a uniquely determined monic divisor of  $x^n - 1$  in  $\mathbb{Z}_4[x]$ .

**Proposition 2.4.** *Let  $n$  be an odd integer, and let  $f(x)$  be a polynomial in  $\mathbb{Z}_2[x]$  dividing  $x^n - 1$ . Then there exists a unique monic polynomial  $g(x) \in \mathbb{Z}_4[x]$  dividing  $x^n - 1$  with  $\bar{g}(x) = f(x)$ .*

*Proof.* Let  $p(x) = x^n - 1$ . The derivative

$$\frac{d}{dx}p(x) = nx^{n-1} \neq 0 \quad (2.7)$$

in any extension field of  $\mathbb{Z}_2$  since  $n$  is odd and hence  $2 \nmid n$ . If  $\alpha$  is a root of  $p(x)$  of multiplicity  $k > 1$ , then

$$\begin{aligned} p(x) &= (x - \alpha)^k q(x), \text{ with } q(\alpha) \neq 0 \\ \frac{d}{dx}p(x) &= k(x - \alpha)^{k-1}q(x) + (x - \alpha)^k \frac{d}{dx}q(x) \end{aligned}$$

Since  $k > 1$ , then  $\frac{d}{dx}p(\alpha) = 0$  which contradicts (2.7). Thus the roots of  $x^n - 1$  are all distinct, and hence since  $\mathbb{Z}_2[x]$  is a unique factorization domain, then  $x^n - 1$  may be written as a product of irreducible polynomials

$$x^n - 1 = f_1(x)f_2(x)\dots f_r(x). \quad (2.8)$$

Each factor of  $x^n - 1$  must have distinct roots as well. In particular, the polynomials  $f_1(x), f_2(x), \dots, f_r(x)$  of (2.8) cannot have a common factor, and hence they are pairwise coprime. By Hensel's Lemma, there exist monic, pairwise coprime polynomials  $g_1(x), g_2(x), \dots, g_r(x)$  in  $\mathbb{Z}_4[x]$  such that  $x^n - 1 = g_1(x)g_2(x) \dots g_r(x)$ , and  $\bar{g}_i(x) = f_i(x)$ ,  $i = 1, \dots, r$ . Since  $f(x)$  divides  $x^n - 1$ , then  $f(x) = f_1(x)f_2(x) \dots f_s(x)$  for some  $1 \leq s \leq r$ . Taking  $g(x) = g_1(x)g_2(x) \dots g_s(x)$  we will have that  $g(x)$  divides  $x^n - 1$  and  $\bar{g}(x) = f(x)$ .

The uniqueness part is elaborate and will be omitted.

□

The unique monic polynomial  $g(x) \in \mathbb{Z}_4[x]$  of Proposition 2.4 is called *the Hensel lift* of the polynomial  $f(x) \in \mathbb{Z}_2[x]$ . So far we have shown the existence of the Hensel lift of a polynomial over  $\mathbb{Z}_2$ . We conclude this section by showing how to compute the Hensel lift using Graeffe's method for finding a polynomial whose roots are the squares of the roots of  $f(x) \in \mathbb{Z}_2[x]$ .

**Proposition 2.5.** *Let  $f(x) \in \mathbb{Z}_2[x]$  be not divisible by  $x$  and have no multiple roots. Write  $f(x) = e(x) - d(x)$ , where  $e(x)$  contains only even power terms and  $d(x)$  contains only odd power terms. Let  $g(x^2) = \pm [e^2(x) - d^2(x)]$ , where we take  $+$  if  $\deg e(x) > \deg d(x)$  and  $-$  otherwise. Then  $g(x)$  is the Hensel lift of  $f(x)$ .*

*Proof.* Clearly, by the choice of  $\pm$  sign the polynomial  $g(x^2)$  is forced to be monic, and that is all that the choice of  $\pm$  affects. Thus, we may assume that  $\deg e(x) > \deg d(x)$ , and the proof of the other case is similar. Note that  $e(-x) = e(x)$  since  $e(x)$  contains only even power terms, and  $d(-x) = -d(x)$  since  $d(x)$  contains only odd power terms. Therefore,  $g(x^2) = [e(x) - d(x)][e(x) + d(x)] = f(x)[e(-x) - d(-x)] = f(x)f(-x)$ , computed in  $\mathbb{Z}_4[x]$ .

Now, from the theory of fields, there exists an odd positive integer  $n$  such that  $f(x)|x^n - 1$  over  $\mathbb{Z}_2$ . Then in  $\mathbb{Z}_4[x]$  we can write

$$x^n - 1 = a(x)f(x) + 2b(x),$$

for some  $a(x), b(x) \in \mathbb{Z}_4[x]$ . Then

$$(-x)^n - 1 = a(-x)f(-x) + 2b(-x),$$

and since  $n$  is odd

$$\begin{aligned}
(x^2)^n - 1 &= (x^n - 1)(x^n + 1) = -(x^n - 1)((-x)^n - 1) \\
&= -[a(x)f(x) + 2b(x)][a(-x)f(-x) + 2b(-x)] \\
&= -a(x)a(-x)f(x)f(-x) + 2[b(x)a(-x)f(-x) + b(-x)a(x)f(x)] \\
&= -g(x^2)a(x)a(-x) + 2[b(x)a(-x)f(-x) + b(-x)a(x)f(x)].
\end{aligned}$$

Writing  $f(x) = e(x) - d(x)$ ,  $a(x) = e_a(x) - d_a(x)$ , and  $b(x) = e_b(x) - d_b(x)$ , where  $e(x)$ ,  $e_a(x)$ ,  $e_b(x)$  consist of only even power terms and  $d(x)$ ,  $d_a(x)$ ,  $d_b(x)$  only of odd power terms, it is easy to verify that

$$2[b(x)a(-x)f(-x) + b(-x)a(x)f(x)] = 0.$$

Thus  $g(x^2)|(x^2)^n - 1$  in  $\mathbb{Z}_4[x]$ . Hence  $g(x)|x^n - 1$  in  $\mathbb{Z}_4[x]$ . Therefore,  $g(x)$  is the Hensel lift of  $f(x)$ .

□

**Example.** The polynomial  $f(x) = x^3 + x^2 + 1$  is a monic irreducible polynomial and a divisor of  $x^7 - 1$  over  $\mathbb{Z}_2$ . Write  $f(x) = e(x) - d(x)$ , where  $e(x) = x^2 + 1$  and  $d(x) = -x^3$ . Since  $\deg d(x) > \deg e(x)$ , then  $g(x^2) = -[e^2(x) - d^2(x)] = -[(x^2 + 1)^2 - x^6] = x^6 - x^4 - 2x^2 - 1$ . Then  $g(x) = x^3 - x^2 - 2x - 1$  is the Hensel lift of  $f(x) = x^3 + x^2 + 1$ , and  $g(x)$  is a divisor of  $x^7 - 1$  in  $\mathbb{Z}_4[x]$ .

## 2.2 Cyclic codes over $\mathbb{Z}_4$

A *quaternary* code is a code over the alphabet  $Q = \mathbb{Z}_4$ , the integers modulo 4.

The quaternary cyclic codes will be called  $\mathbb{Z}_4$ -*cyclic*. As in the binary case, we have a bijection

$$\begin{aligned}
\mathbb{Z}_4/(x^n - 1) &\rightarrow \mathbb{Z}_4^n; \\
f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_0 &\mapsto (a_{n-1}, a_{n-2}, \dots, a_0),
\end{aligned} \tag{2.9}$$



$$H = \begin{pmatrix} h_0 & h_1 & \dots & h_{n-k} & & & \\ & h_0 & h_1 & \dots & h_{n-k} & & \\ & & \ddots & \ddots & & \ddots & \\ & & & h_0 & h_1 & \dots & h_{n-k} \end{pmatrix}.$$

### 2.3 The Gray map and images of $\mathbb{Z}_4$ -codes

In the previous two sections the main discussion was about the construction of the quaternary codes. To be able to analyze the capabilities and limitations of these codes one needs a metric on  $\mathbb{Z}_4$ . It turns out that the Lee metric defined towards the end of Section 1.1 yields more interesting results than the Hamming metric.

To get a binary code from a  $\mathbb{Z}_4$  code, one uses the Gray map, defined by

$$\Phi : \mathbb{Z}_4 \rightarrow \mathbb{Z}_2^2; \tag{2.10}$$

$$0 \mapsto 00, 1 \mapsto 01, 2 \mapsto 11, 3 \mapsto 10.$$

It extends naturally to a map  $\Phi : \mathbb{Z}_4^n \rightarrow \mathbb{Z}_2^{2n}$ . It turns out that this map is an isometry between  $\mathbb{Z}_4^n$  and  $\mathbb{Z}_2^{2n}$  if we consider  $\mathbb{Z}_4^n$  with the Lee metric and  $\mathbb{Z}_2^{2n}$  with the Hamming metric.

**Theorem 2.7.** *The Gray map  $\Phi : (\mathbb{Z}_4^n, \text{Lee metric}) \rightarrow (\mathbb{Z}_2^{2n}, \text{Hamming metric})$  is an isometry.*

*Proof.* It is easy to verify that for each  $x \in \mathbb{Z}_4$

$$wt_L(x) = wt_H(\Phi(x)),$$

where  $wt_L$  and  $wt_H$  are the respective Lee and Hamming weights defined in Section 1.1. Thus if  $v = (v_0, v_1, \dots, v_{n-1}) \in \mathbb{Z}_4^n$ , then

$$wt_L(v) = \sum_{i=0}^{n-1} wt_L(v_i) = \sum_{i=0}^{n-1} wt_H(\Phi(v_i)) = wt_H(\Phi(v)).$$

□

We now discuss the construction of some well known nonlinear binary codes. Start with a polynomial  $f(x) = x^3 + x + 1$ . This polynomial is irreducible over  $\mathbb{Z}_2$ , and it is a divisor of  $x^7 - 1$ . Hensel lift  $f(x)$  to the polynomial  $g(x) = x^3 + 2x^2 + x - 1$  which is also a divisor of  $x^7 - 1$  in  $\mathbb{Z}_4[x]$ , and  $\bar{g}(x) = f(x)$ , where  $\bar{g}(x)$  is defined as in Section 2.1. Then  $g(x)$  generates a cyclic code of length 7 over  $\mathbb{Z}_4$ . Add a parity-check symbol, and we get an equivalent code to the self-dual ‘‘octacode’’, of length 8 over  $\mathbb{Z}_4$ . The octacode is a code with 256 codewords and minimal Lee distance 6, i.e. it is a  $[8, 4, 6]_4$  code. This code is obtained from the construction of the Leech lattice [8]. Its generator matrix which is also its parity-check matrix is

$$G = \begin{pmatrix} 1 & 2 & 1 & 3 & 0 & 0 & 0 & 1 \\ 0 & 1 & 2 & 1 & 3 & 0 & 0 & 1 \\ 0 & 0 & 1 & 2 & 1 & 3 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 & 1 & 3 & 1 \end{pmatrix}$$

It has been shown in [10] that the Nordstrom-Robinson code is the binary image of the octacode under the Gray map. The Nordstrom-Robinson code is a nonlinear code of length 16, with 256 codewords and minimum distance 6.

Analogous descriptions of Preparata and Kerdock codes were given in [4]. If we replace the polynomial  $f(x) = x^3 + x + 1$  by a generator polynomial of a Hamming code of length  $2^m - 1$ , where  $m$  is odd, and repeat the process, we get the Preparata codes. Using the duals of the codes we get the Kerdock codes.

Let  $f(x)$  be an irreducible polynomial of degree  $m$  in  $\mathbb{Z}_2[x]$ , and let  $g(x)$  be its Hensel lift in  $\mathbb{Z}_4[x]$ . Let  $\tilde{h}(x)$  be the reciprocal polynomial to  $h(x) = \frac{x^n - 1}{(x - 1)g(x)}$ .

**Theorem 2.8** ([4]). (a) *The cyclic code with generator polynomial  $\tilde{h}(x)$ , extended by the addition of an overall parity-check symbol, is mapped by the Gray map  $\Phi$  onto the Kerdock code  $K(2m)$ .*

(b) *The cyclic code with generator polynomial  $g(x)$ , extended by the addition of an overall parity-check symbol, is mapped by the Gray map  $\Phi$  onto the ‘Preparata’ code  $P'(2m)$ .*

□

The quotes around ‘Preparata’ in the theorem are due to the fact that although this code has the same parameters and the same weight distribution as the Preparata code, it is not equivalent to it. It is compared to the case of Hamming codes, where several different descriptions exist but one of them turns out to be cyclic.

## CHAPTER III. LOOP TRANSVERSAL CODES

Loop transversal (LT) codes were introduced by J.D.H. Smith in [16]. The purpose of that paper was to initiate a new approach to linear block codes. Rather than first constructing a code and then analyzing to see which errors can be corrected, here one first specifies the errors to be corrected and then applies some algorithm to construct a code capable of correcting these errors. Thus one has freedom in deciding which errors to be corrected. This is especially beneficial in the case of burst error correction as described in [13] and [5]. As with the lexicodes, we may use a greedy algorithm to construct the LT codes. In contrast, nonbinary lexicodes were shown to be nonlinear. The loop transversal codes are always linear. The binary images of LT codes over  $\mathbb{Z}_4$  also produce some interesting results. In particular, it will be shown in chapter IV that an  $[8, 4, 6]_4$  greedy LT code is equivalent to the octacode, hence the Nordstrom-Robinson code is the binary image of the  $[8, 4, 6]_4$  LT code under the Gray map.

### 3.1 General construction

Given a set  $E \subset V$  of errors, where  $(V, +, 0)$  is a (not necessarily abelian) group, the *channel*, a linear map  $\varepsilon : V \rightarrow V$  is defined such that  $\varepsilon|_E$  is injective. The map  $\varepsilon$  is called the *syndrome* function. The kernel  $C$  of  $\varepsilon$  is realized as the code correcting  $E$ . Then  $V$  may be represented as the disjoint union of cosets of  $C$ ,

$$V = \bigcup_{e \in E} (C + e)$$

where  $E$  is the set of coset representatives, called a *transversal* to  $C$ . Thus each element  $v \in V$  can be expressed uniquely as

$$v = \delta(v) + \varepsilon(v) \quad (3.1)$$

where  $\delta(v) \in C$  and  $\varepsilon(v) \in E$ . (In the coding context, a received word  $v$  has been exposed to error  $\varepsilon(v)$ , and hence has to be decoded as  $\delta(v)$ ). A binary operation  $*$  is defined on  $E$  by

$$t_1 * t_2 = \varepsilon(t_1 + t_2) \quad (3.2)$$

For any  $t_1, t_2 \in E$ , the equation  $x * t_1 = t_2$  has a unique solution  $x$  [15]. If the equation  $t_1 * y = t_2$  also has a unique solution, then  $E$  is said to be a *loop transversal*. The algebra  $(E, *, \varepsilon(0))$  is a loop [16]. In traditional coding theory the channel  $V$  is an abelian group, and thus each transversal is a loop transversal, and the loop  $(E, *, \varepsilon(0))$  is an abelian group. For  $t_i$  in  $E$  the product  $\prod_{i=1}^r t_i$  is defined inductively by

$$\prod_{i=1}^0 t_i = \varepsilon(0)$$

and

$$\prod_{i=1}^r t_i = \left[ \prod_{i=1}^{r-1} t_i \right] * t_r.$$

Now, if  $V$  is a finite dimensional vector space over a field  $\mathbb{F}$ , define  $\lambda \times t = \varepsilon(\lambda t)$  for  $\lambda$  in  $\mathbb{F}$  and  $t$  in  $E$ . Then the algebra  $(E, *, \mathbb{F})$  becomes a vector space over  $\mathbb{F}$ . Induction on  $r$  extends (3.2) to

$$\varepsilon\left(\sum_{i=1}^r \lambda_i t_i\right) = \prod_{i=1}^r (\lambda_i \times t_i) \quad (3.3)$$

for  $t_i$  in  $E$ . It is reasonable to require  $E$  to contain a basis  $\{e_1, \dots, e_n\}$  for  $V = \mathbb{F}^n$ , where  $e_i$  has 1 in the  $i$ -th position<sup>1</sup> and 0's everywhere else (the set of single errors). Then knowledge of a portion of the vector space  $(E, *, \mathbb{F})$  is sufficient to determine the corresponding portion

---

<sup>1</sup>The  $i$ -th position is counted from the right, i.e.  $e_1 = 0 \dots 001$ ,  $e_2 = 0 \dots 010$ ,  $e_3 = 0 \dots 100$ , etc.

of the code  $C$ . Indeed, for  $k \leq n$ ,

$$\begin{aligned} C &= \{\delta(v) | v \in V\} = \{v - \varepsilon(v) | v \in V\} \\ &= \left\{ \sum_{i=1}^k \lambda_i e_i - \varepsilon \left( \sum_{i=1}^k \lambda_i e_i \right) \middle| \lambda_i \in \mathbb{F} \right\} \\ &= \left\{ \sum_{i=1}^k \lambda_i e_i - \prod_{i=1}^k (\lambda_i \times e_i) \middle| \lambda_i \in \mathbb{F} \right\}, \end{aligned}$$

the so-called *Principle of Local Duality* [16].

Decoding of these codes is described as a simple table look-up, motivated by the fact that a table look-up is much cheaper nowadays than many of the algebraic techniques used for decoding. A more detailed study on loop transversal code construction may be found in [16].

### 3.2 Greedy approach

In Section 3.1 a general loop transversal code construction was described. In this section a more specific algorithm will be given. It is the description of the syndrome function or parity map  $\varepsilon$  as given in [16] that defines the code. The way we choose our syndrome map also plays a crucial role in the performance of these codes. In his Ph.D. dissertation [11], Frank Hummer proposed a greedy approach. It was shown that greedy LT codes coincide with the lexicodes in a binary channel. The greedy LT construction also produces the perfect Hamming codes and Golay codes (both binary and ternary). In the next chapter, it will be shown that the quaternary greedy LT codes with Lee metric include the octacode whose image under the Gray map is the nonlinear Nordstrom-Robinson code.

We next give the basic outlines of the greedy algorithm following [11]. The syndrome function  $\varepsilon$  is constructed so that it is linear and  $\varepsilon|_E$  is injective, i.e. no two errors that we are willing to correct should get the same syndrome value assigned. First the algorithm sorts the set  $E$  of errors into a lexicographic order [12]. Unless otherwise specified, for the

remainder of this thesis the order will be assumed to be lexicographic. Then for each error  $u \in E$  a syndrome value  $\varepsilon(u)$  is assigned, so that  $\varepsilon(u) \neq \varepsilon(v)$  for any  $v < u$  (thus preserving injectivity). To preserve the linearity, for  $u_i$  in  $E$  and  $\alpha_i$  in  $\mathbb{F}_q$ , if  $u_t = \sum_{i=1}^{t-1} \alpha_i u_i$  then  $\varepsilon(u_t)$  is assigned to  $\sum_{i=1}^{t-1} \alpha_i \varepsilon(u_i)$ . Thus for each error  $e \in E$  the syndrome value

$$\varepsilon(e) = \begin{cases} 0 & \text{if } e = 0 \\ \min\{v \in V \mid v \neq \varepsilon(e'), e' < e_{i+1}\} & \text{if } e = e_i \\ \sum_{i=1}^t \alpha_i \varepsilon(e_i) & \text{if } e = \sum_{i=1}^t \alpha_i e_i \end{cases} \quad (3.4)$$

is assigned successively, where the  $e_i$  ( $i = 1, \dots, n$ ) are the standard basis vectors with 1 at the  $i$ -th position and 0 everywhere else. In the first line the initial value of the syndrome is assigned, the second line is the greedy choice, and the last line guarantees the linearity condition. Then the kernel of  $\varepsilon$  defined in (3.4) is a linear loop transversal code correcting  $E$ . If  $E$  is the set of all vectors  $e \in \mathbb{F}^n$  of weight up to  $t$ , then the LT code produced will have a minimum distance  $d = 2t + 1$ .

Of course, there can be many ways to choose the syndrome values that satisfy the above mentioned conditions. The name greedy in this algorithm refers to the fact that for each standard basis vector  $e_i$  the very first vector in the lexicographic order of  $\mathbb{F}_q^n$  that satisfies these requirements is chosen. Thus greedily choosing the smallest possible vectors as syndrome values one minimizes the dimension of the syndrome space hence maximizing the dimension of the code.

**Example.** The following is an example of a binary loop transversal code of length 6 correcting the set  $E$  of burst errors of length at most 2. The assigned syndrome value for each

error is given in the next column .

<b>E</b>	<b>Syndrome</b>	<b>E</b>	<b>Syndrome</b>
000000	0000	001000	1000
000001	0001	001100	1100
000010	0010	010000	0101
000011	0011	011000	1101
000100	0100	100000	1010
000110	0110	110000	1111

The syndrome space of this code  $C$  has dimension 4. Thus the dimension of  $C$  is  $6 - 4 = 2$ . The four codewords of  $C$  are 000000, 010101, 101010, and 111111. These codewords may be obtained by using the principle of local duality. For example,

$$(010000 \oplus_2 001000) \ominus_2 (010000 * 001000) = 011000 \ominus_2 001101 = 010101,$$

$$(100000 \oplus_2 010000) \ominus_2 (100000 * 010000) = 110000 \ominus_2 001111 = 111111.$$

The other codewords may be obtained similarly, but the fact that the code is linear makes it sufficient to find just two nonzero codewords, and take their linear combinations.

An interesting property of the LT codes is that if two error patterns  $E_1$  and  $E_2$  are nested, in the sense that  $E_1 \subset E_2$  and  $t_2 \not\prec t_1$  for all  $t_1 \in E_1$  and  $t_2 \in E_2 - E_1$ , then the corresponding codes  $C_1$  and  $C_2$  are also nested, i.e.  $C_1 \subset C_2$ . Thus if we take  $E_1$  to be the first ten vectors of  $E$  in the previous example, then the corresponding code will be the subspace of  $C$  generated by 010101. This is the notion of  $E$  being self-subordinate in [11].

## CHAPTER IV. NEW RESULTS ON LOOP TRANSVERSAL CODES

The implementation of the greedy loop transversal algorithm has produced many astonishing results in both the binary and nonbinary cases. A vast number of optimal and best known codes are produced using the algorithm. Along with these codes the extended Hamming codes, the binary Golay [24, 12, 8], Reed-Muller [16, 5, 8], the quadratic residue [18, 9, 6], and the ternary Golay [12, 6, 6] codes are obtained. Among the quaternary codes a code equivalent to the octacode described in Section 2.3 is obtained, and consequently the Nordstrom-Robinson code as its binary image under the Gray map. Record breaking codes over the 7-ary alphabet are also obtained.

### 4.1 Generator matrix

To give a complete description of the greedy LT codes an algorithm to compute a generator matrix is presented here. First, obtaining the syndrome values for the standard basis vectors

$$e_1 = 0 \dots 001, e_2 = 0 \dots 010, e_3 = 0 \dots 100, \dots, e_n = 1 \dots 000 \quad (4.1)$$

of  $V = Q^n$ , where  $Q$  is the alphabet, completely specifies the syndrome function. Thus we can narrow down our attention to these vectors and their images under  $\varepsilon : V \rightarrow V$ . Second, if the dimension of the syndrome space or the redundancy of the code is  $m \leq n$ , then the vectors  $e_1, \dots, e_m$  must appear as images  $\varepsilon(e_{i_1}), \dots, \varepsilon(e_{i_m})$  as given in the next proposition.

Let  $\varepsilon^{-1}(e_i)$  denote the vector  $e_j$  such that  $\varepsilon(e_j) = e_i$ .

**Proposition 4.1.** *Let  $e_1, \dots, e_n$  given by (4.1) be the standard basis vectors for  $V$ , and let  $m$  be the redundancy of a greedy LT code. Then for each  $e_i$ ,  $i = 1, \dots, m$  there exists an integer*

$1 \leq j \leq n$  such that  $e_i = \varepsilon(e_j)$ . Moreover, the map  $\varepsilon^{-1}$  defined on  $e_1, \dots, e_m$  is a strictly monotone increasing function.

*Proof.* By the greedy choice in the construction of  $\varepsilon$  given in (3.4) each  $e_i$ ,  $i = 1, \dots, m$ , must be considered as a choice before any other vector  $v > e_i$  in  $V$ . The vector  $e_i$  cannot be a linear combination of any set of vectors which are less than  $e_i$ , hence  $e_i$  must be assigned as a syndrome value before any other vector  $v > e_i$  can be assigned. Thus for some error vector  $t \in E$  we have  $e_i = \varepsilon(t)$ . Moreover, for all  $t' < t$  in  $E$  we have  $\varepsilon(t') < e_i$ . In particular, the vectors  $e_1, \dots, e_{i-1}$  must have been assigned as a syndrome value before  $e_i$  gets assigned. Therefore,  $\varepsilon^{-1}(e_{i-1}) < \varepsilon^{-1}(e_i)$ . By induction,

$$\varepsilon^{-1}(e_1) < \varepsilon^{-1}(e_2) < \dots < \varepsilon^{-1}(e_n)$$

which proves the second assertion.

Because we require  $e_j \in E$  for all  $j = 1, \dots, n$ , if  $t \neq e_j$  then  $e_{j-1} < t < e_j$  for some  $1 \leq j \leq n$ . Then  $t$  may be written as a linear combination

$$t = \alpha_1 e_1 + \dots + \alpha_{j-1} e_{j-1},$$

where  $\alpha_1, \dots, \alpha_{j-1}$  are elements of  $Q$ . By linearity of  $\varepsilon$

$$e_i = \varepsilon(t) = \alpha_1 \varepsilon(e_1) + \dots + \alpha_{j-1} \varepsilon(e_{j-1}),$$

which is a contradiction since  $\varepsilon(e_k) < e_i$  for all  $k = 1, \dots, j-1$ . Therefore,  $t = e_j$  for some  $1 \leq j \leq n$ . □

**Corollary 4.2.** *If  $n > m$  then there exist  $n - m$  vectors  $e_j$ ,  $m + 1 \leq j \leq n$ , such that  $\varepsilon(e_j) < e_{m+1}$  and  $\varepsilon(e_j) \neq e_i$  for any  $i = 1, \dots, m$ . Hence for all  $e_j$  such that  $\varepsilon(e_j)$  is not a unit vector we have  $e_j \geq e_{m+1}$ .*

*Proof.* The existence follows from the injectivity of  $\varepsilon$ . To prove that  $j \geq m + 1$ , suppose the contrary. From  $\varepsilon(e_j) < e_{m+1}$  it follows that  $\varepsilon(e_j)$  may be written as

$$\varepsilon(e_j) = \alpha_1 e_1 + \dots + \alpha_m e_m,$$

for  $\alpha_1, \dots, \alpha_m$  in  $Q$ . Then

$$e_j = \alpha_1 \varepsilon^{-1}(e_1) + \dots + \alpha_m \varepsilon^{-1}(e_m). \quad (4.2)$$

By Proposition 4.1,  $\varepsilon^{-1}(e_i) = e_{k_i}$  for all  $i = 1, \dots, m$ . Then (4.2) becomes

$$e_j = \alpha_1 e_{k_1} + \dots + \alpha_m e_{k_m}$$

which is possible only if  $e_j = e_{k_i} = \varepsilon^{-1}(e_i)$  for some  $i$ . Then  $\varepsilon(e_j) = e_i$  which contradicts the assumption. □

The next theorem sheds light on how to construct a generator matrix for an LT code.

Let  $m$  be the redundancy, and  $n > m$  be the length of an LT code. Let  $v_k = \varepsilon(e_{j_k})$ ,  $k = 1, \dots, n - m$ , such that  $v_k \neq e_i$  for any  $i = 1, \dots, m$ , guaranteed by Corollary 4.2. Then the  $v_k$  may be written as

$$v_k = \alpha_{k1} e_1 + \dots + \alpha_{km} e_m.$$

Define

$$u_k = e_{j_k} - \alpha_{k1} \varepsilon^{-1}(e_1) - \dots - \alpha_{km} \varepsilon^{-1}(e_m), \quad k = 1, \dots, n - m \quad (4.3)$$

**Theorem 4.3.** *The vectors  $u_1, u_2, \dots, u_{n-m}$  given by (4.3) are linearly independent.*

*Proof.* Suppose  $\beta_1 u_1 + \dots + \beta_{n-m} u_{n-m} = 0$ . Then

$$\begin{aligned} & \beta_1 [e_{j_1} - \alpha_{11} \varepsilon^{-1}(e_1) - \dots - \alpha_{1m} \varepsilon^{-1}(e_m)] \\ + & \beta_2 [e_{j_2} - \alpha_{21} \varepsilon^{-1}(e_1) - \dots - \alpha_{2m} \varepsilon^{-1}(e_m)] \\ & \vdots \\ + & \beta_{n-m} [e_{j_{n-m}} - \alpha_{n-m,1} \varepsilon^{-1}(e_1) - \dots - \alpha_{n-m,m} \varepsilon^{-1}(e_m)] \\ = & \sum_{k=1}^{n-m} \beta_k e_{j_k} - \left( \sum_{k=1}^{n-m} \beta_k \alpha_{k1} \right) \varepsilon^{-1}(e_1) - \dots - \left( \sum_{k=1}^{n-m} \beta_k \alpha_{k,m} \right) \varepsilon^{-1}(e_m) = 0 \end{aligned}$$

Because  $\varepsilon^{-1}(e_m) > \varepsilon^{-1}(e_i)$ ,  $i = 1, \dots, m - 1$ , either  $\beta_i = 0$  for all  $i = 1, \dots, n - m$  or there exists a  $1 \leq k \leq n - m$  such that  $\beta_k e_{j_k} = a_m \varepsilon^{-1}(e_m)$ , where  $a_m = \sum_{k=1}^{n-m} \beta_k \alpha_{k,m}$ . The latter would imply that  $v_k = \varepsilon(e_{j_k}) = e_m$ , which would contradict the hypothesis. □

**Corollary 4.4.** *The matrix*

$$G = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-m} \end{pmatrix}$$

*is a generator matrix for an LT code satisfying the hypothesis of Theorem 4.3.*

## 4.2 Error detection in loop transversal codes

Traditionally, in coding theory codes correcting and codes both correcting and detecting some errors are considered. In the case of white noise, the former are referred to as codes of odd minimum distance and the latter are referred to as codes of even minimum distance. For example, the perfect Hamming codes have a minimum distance three and they correct any single error, whereas the extended Hamming codes have a minimum distance four and they correct any single error and detect any double error (any vector of Hamming weight two). Loop transversal codes have been known as codes correcting the given set of errors. If we take  $E$ , the set of errors, to be the set of all vectors in  $V$  of weight less than  $t$ , then the LT code constructed by the greedy algorithm given in [12] will have a minimum distance  $d = 2t + 1$ , i.e. it will be a code correcting  $t$  random errors. In general, except for the binary case where one can add a parity-check, it is not clear if one can obtain a code of minimum distance  $d$  from a code of minimum distance  $(d - 1)$ . A natural question would be if we can find a way of extending the LT codes. It turns out that the answer is yes. When we construct an LT code correcting  $t$ -random errors, we start with explicitly stating the set  $E$  of errors we want our code  $C$  to correct. From the usual coding theory perspective this is equivalent to saying that the minimum distance of  $C$  is at least  $d = 2t + 1$ , an odd number. On the other hand, if we wish to detect the error  $v$ , a vector of weight  $t + 1$  we want  $v \neq c \oplus_q e$  for any codeword  $c$  and correctable error  $e$ . Thus we do not wish any difference  $v \ominus_q e$  of a detectable error and a correctable error to be a codeword. Furthermore, a linear code correcting  $t$  random errors

is guaranteed to have a minimal distance, and hence minimal weight  $d = 2t + 1$ . Therefore, no difference vector  $v \ominus_q e$  of weight  $wt(v \ominus_q e) < 2t + 1$  can be a codeword. So, if in the construction of the LT codes we avoid assigning a zero syndrome to any vector from the set  $DE = \{(v \ominus_q e) | v \in D, e \in E, wt(v - e) = 2t + 1\}$ , then the resulting code will detect  $t + 1$  errors and have a minimum distance  $2(t + 1)$ . This shows that adding an extra check to the greedy loop transversal algorithm given in [12] will enable one to construct LT codes with even minimal distances and error detection. Note that extending the LT codes this way is independent of whether we choose the Hamming metric or the Lee metric on  $V$ . The algorithm simplifies a great deal when we restrict our attention to the Hamming metric.

### 4.3 Minimum syndrome weight and $m$ -independence

In this section some algebraic tools to simplify the syndrome function computations in channel  $a$  with a Hamming metric are provided. Although these applications are based on simple observations, the theorems proved here help a great deal in making the computations feasible. The major results of these computations, in the form of tables for the dimensions of greedy LT codes, are presented in Appendix A. In particular, record-breaking codes of minimum distance six with parameters  $[32, 24, 6]$ ,  $[33, 25, 6]$ ,  $[34, 26, 6]$ ,  $[35, 27, 6]$ ,  $[36, 28, 6]$ ,  $[37, 29, 6]$ , and  $[38, 30, 6]$  over  $\mathbb{Z}_7$  are obtained.

The first observation is that the images of the standard basis vectors  $e_i$  ( $i = 1, \dots, n$ ) must either be standard basis vectors themselves, or they must have a Hamming weight greater than the minimum distance of the code minus one.

**Theorem 4.5.** *Let  $C$  be a greedy LT code of length  $n$ , dimension  $k$ , and minimum distance  $d$ , and let  $\{e_1, \dots, e_n\}$  be the standard basis of  $V$  given by (4.1). Then for  $j = 1, \dots, n$  either  $\varepsilon(e_j) = e_i$  for some  $i \leq j$ , or  $wt_H(\varepsilon(e_j)) \geq d - 1$ .*

*Proof.* The first part is a consequence of Proposition 4.1. So assume that  $\varepsilon(e_j) = v \neq e_i$ , for any  $1 \leq i \leq n - k$ . Assume also that  $wt_H(v) < d - 1$ . Then  $v$  can be written as

$$v = \varepsilon(e_j) = \alpha_1 e_{i_1} + \alpha_2 e_{i_2} + \dots + \alpha_{d-2} e_{i_{d-2}},$$

where  $\alpha_i \in V_q$ . Again by Proposition 4.1 each of the  $e_{i_t}$  ( $1 \leq t \leq d-2$ ) must be a syndrome vector for some  $e_j$ , i.e.

$$v = \varepsilon(e_j) = \alpha_1 \varepsilon(e_{j_1}) + \alpha_2 \varepsilon(e_{j_2}) + \dots + \alpha_{d-2} \varepsilon(e_{j_{d-2}}). \quad (4.4)$$

By linearity of  $\varepsilon$

$$\varepsilon(e_j) = \varepsilon(\alpha_1 e_{j_1} + \alpha_2 e_{j_2} + \dots + \alpha_{d-2} e_{j_{d-2}}).$$

Then

$$\varepsilon(e_j - \alpha_1 e_{j_1} - \alpha_2 e_{j_2} - \dots - \alpha_{d-2} e_{j_{d-2}}) = 0.$$

Hence  $c = e_j - \alpha_1 e_{j_1} - \alpha_2 e_{j_2} - \dots - \alpha_{d-2} e_{j_{d-2}}$  must be a codeword. Since each of the terms contributes only one to the Hamming weight of  $c$ , then  $c$  cannot have a Hamming weight greater than  $d-1$ . But this contradicts the assumption that  $C$  has a minimum distance  $d$ .

□

**Theorem 4.6.** *Let  $C$  be a linear code of length  $n$  and minimum distance  $d$ , and let  $\varepsilon$  be the syndrome function defining that code. Then every subset of  $d-1$  vectors from the set  $\{\varepsilon(e_1), \dots, \varepsilon(e_n)\}$  is linearly independent.*

*Proof.* Let  $\alpha_1 \varepsilon(e_{i_1}) + \alpha_2 \varepsilon(e_{i_2}) + \dots + \alpha_{d-1} \varepsilon(e_{i_{d-1}}) = 0$ . By linearity of  $\varepsilon$

$$\varepsilon(\alpha_1 e_{i_1} + \alpha_2 e_{i_2} + \dots + \alpha_{d-1} e_{i_{d-1}}) = 0.$$

Then the word

$$c = \alpha_1 e_{i_1} + \alpha_2 e_{i_2} + \dots + \alpha_{d-1} e_{i_{d-1}} \quad (4.5)$$

must be a codeword. Each term on the right hand side of (4.5) contributes at most one to the Hamming weight of  $c$ . By assumption,  $c$  cannot have a Hamming weight less than  $d$ . Therefore,  $\alpha_i = 0$  for all  $i = 1, \dots, d-1$ .

□

**Definition 4.1.** *A set  $S$  of vectors is called  $m$ -independent if each set of  $m$  vectors from  $S$  is linearly independent.*

Our next theorem states that any  $(d-1)$ -independent set determines a syndrome function of a linear code with minimum distance  $d$ .

**Theorem 4.7.** *A linear map  $\varepsilon : V \rightarrow V$  is a syndrome function defined by a linear  $[n, k, d]$  code if and only if the set  $S = \{\varepsilon(e_1), \varepsilon(e_2), \dots, \varepsilon(e_n)\}$  is  $(d-1)$ -independent.*

*Proof.* One direction follows from Theorem 4.6.

If  $S = \{v_1, \dots, v_n\}$  is a  $(d-1)$ -independent set, define a linear function  $\varepsilon : V \rightarrow V; e_i \mapsto v_i$ . Suppose  $\varepsilon(c_1) = \varepsilon(c_2)$  for some  $c_1 \neq c_2 \in V$ . By linearity of  $\varepsilon$  the vector  $c_1 - c_2$  is in the kernel of  $\varepsilon$ . Using the standard representation we can write:

$$c_1 = \alpha_1 e_1 + \dots + \alpha_n e_n ;$$

$$c_2 = \beta_1 e_1 + \dots + \beta_n e_n .$$

Then

$$0 = \varepsilon(c_1 - c_2) = (\alpha_1 - \beta_1)\varepsilon(e_1) + (\alpha_2 - \beta_2)\varepsilon(e_2) + \dots + (\alpha_n - \beta_n)\varepsilon(e_n)$$

and

$$0 = \gamma_1 v_1 + \gamma_2 v_2 + \dots + \gamma_n v_n , \tag{4.6}$$

where  $\gamma_i = \alpha_i - \beta_i$ . Since  $c_1 \neq c_2$ , not all of the  $\gamma_i$  are zero. The  $(d-1)$ -independence of  $S$  guarantees that there must be at least  $d$  nonzero summands on the right hand side of equation (4.6). Thus  $\alpha_i \neq \beta_i$  for at least  $d$  values of  $i \in \{1, 2, \dots, n\}$ , i.e.  $wt_H(c_1 - c_2) \geq d$ . Therefore the kernel  $C$  of  $\varepsilon$  is a linear code with minimum distance  $d$ .

□

Thus the knowledge of a  $(d-1)$ -independent set  $S$  completely determines a corresponding  $[n, k, d]$  linear code, where  $n = |S|$  and  $k = n - \dim(S)$ . Since LT codes are linear, and have the syndrome function linear as well, then as a corollary any  $[n, k, d]$  LT code may be specified by finding an appropriate  $(d-1)$ -independent set.

**Corollary 4.8.** *The  $(d-1)$ -independent set of vectors chosen greedily (with lexicographic order) defines the syndrome function for an LT code of minimum distance  $d$ .*

To conclude this section we discuss the efficiency of the LT codes. Certain criteria must be used to determine the efficiency of a particular code. For an  $[n, k, d]$  linear code it is a common practice to fix two of the parameters and estimate the bounds on the third parameter. For example, if  $n$  and  $k$  are fixed one tries to maximize the minimum distance  $d$ , and if  $n$  and  $d$  are fixed one tries to maximize the dimension  $k$  of the code. Many well known upper and lower bounds for  $d$  and  $k$  exist. Some of these bounds are tight for some code lengths where the others are loose and vice versa. The last theorem of this section states that the greedy loop transversal codes meet the well known Varshamov-Gilbert bound. This theorem is an immediate consequence of Corollary 4.8 and R. R. Varshamov's bounding argument in [19].

**Theorem 4.9.** *The greedy loop transversal codes meet the Varshamov-Gilbert bound, i.e. if  $[n, k, d]_q$  are the parameters of a given LT code, then*

$$\sum_{i=0}^{d-2} \binom{n}{i} (q-1)^i \geq q^{n-k}.$$

□

#### 4.4 Nonlinear images of LT codes

As discussed in Section 4.2, one may obtain an LT code of even minimum Lee distance. Of particular interest are the LT codes over  $\mathbb{Z}_4$  of minimum Lee distance  $d$ . An implementation of the greedy loop transversal algorithm over  $\mathbb{Z}_4$  for the error pattern having every word of Lee weight up to 2, with an additional check for error detection, produced the  $[7, 3, 6]_4$  and  $[8, 4, 6]_4$  LT codes. The binary image of the former under the Gray map is a nonlinear code of length 14 having  $2^6 = 64$  codewords and minimum distance 6. This code has the parameters of the best known nonlinear binary code of length 14. The binary image of the  $[8, 4, 6]_4$  LT code is the famous Nordstrom-Robinson code of length 16 having 256 codewords and minimum distance 6.

**Theorem 4.10.** *The Nordstrom-Robinson code is the binary image of the  $[8, 4, 6]_4$  LT code under the Gray map.*

*Proof.* This follows from the uniqueness of the Nordstrom-Robinson code proved by S. L. Snover in [17].

□

In conclusion, it is worth mentioning that the greedy algorithm does not always produce an optimal code. As can be seen in the tables of Appendix A, many of the greedy LT codes are far from being optimal. For example, the ternary LT code of length 72 and minimum distance 5 has dimension 61, whereas the best known linear code of length 72 and minimum distance 5 has dimension 63. There might be a better way of constructing the LT codes, e.g. using dynamic programming, that might result in an optimal construction. More research can also be done on applications of LT codes correcting error patterns specific for those applications.

## BIBLIOGRAPHY

- [1] A. E. Brouwer, *Linear code bounds* (on-line server), Eindhoven University of Technology, The Netherlands, <http://www.win.tue.nl/aeb/voorlincod.html> (Retrieved: 19 December 2004).
- [2] R. A. Brualdi and V. S. Pless, *Greedy codes*, *J. Combin. Theory*, **A-64** (1993), pp. 10–30.
- [3] A. R. Calderbank, A. R. Hammons, Jr., P. V. Kumar, N. J. A. Sloane, and P. Solé, *A linear construction for certain Kerdock and Preparata codes*, *Bull. Amer. Math. Soc.*, **29** (1993), pp. 218-222.
- [4] A. R. Calderbank, A. R. Hammons, Jr., P. V. Kumar, N. J. A. Sloane, and P. Solé, *The  $\mathbb{Z}_4$ -linearity of Kerdock, Preparata, Goethals and related codes*, *IEEE Trans. Inform. Theory*, **40** (1994), pp. 301-319.
- [5] D. -H. Choi and J. D. H. Smith, *Greedy loop transversal codes for correcting error bursts*, *Discrete Math.* **264** (2003), pp. 37-43.
- [6] J. H. Conway, *Integral lexicographic codes*, *Discrete Math*, **83** (1990), pp. 219 - 235.
- [7] J. H. Conway and N. J. A. Sloane, *Lexicographic codes: error-correcting codes from game theory*, *IEEE Trans. Information Theory*, **IT-32** (1986), pp. 337-348.
- [8] J. H. Conway and N. J. A. Sloane, *Sphere-packings, lattices and groups*, 2nd ed., Springer-Verlag, NY 1992, ISBN 0-38-798585-9.
- [9] D. S. Dummit and R. M. Foote, *Abstract algebra*, Prentice-Hall, 1991, ISBN 0-13-004771-6.

- [10] G. D. Forney Jr., N. J. A. Sloane, and M. D. Trott, *The Nordstrom-Robinson code is the binary image of the octacode*, Coding and Quantization: DIMACS/IEEE Workshop, October 19-21, 1992, R. Calderbank, G.D. Forney Jr., and N. Moayeri, Eds., Amer. Math. Soc., 1993, pp. 19-26.
- [11] F. A. Hummer, *Loop transversal codes*, Ph.D. Dissertation, Math. Dept, Iowa State University, 1992.
- [12] F. A. Hummer and J. D. H. Smith, *Greedy loop transversal codes, metrics, and lexicodes*, J. Comb. Math. Comb. Comp. **22** (1996), pp. 143–155.
- [13] F. -L. Hsu, F. A. Hummer, and J. D. H. Smith *Logarithms, syndrome functions, and the information rates of greedy loop transversal codes*, J. Comb. Math. Comb. Comp. **22** (1996), pp. 33-49.
- [14] N. J. A. Sloane, *Algebraic coding theory: recent developments related to  $\mathbb{Z}_4$* , Study of Algebraic Combinatorics (Proceedings Conference on Algebraic Combinatorics, Kyoto 1993), Research Institute for Mathematical Sciences, Kyoto, 1995, pp. 38-52.
- [15] J. D. H. Smith, *Representations of infinite groups and finite quasigroups*, Les Presses de l'Université de Montréal, Montreal, 1986.
- [16] J. D. H. Smith, *Loop transversals to linear codes*. Journal of Combinatorics, Information & System Sciences, **17** (1992), Nos. 1–2, 1–8.
- [17] S. L. Snover, *The uniqueness of the Nordstrom-Robinson and the Golay binary codes*, Ph.D. Dissertation, Math. Dept., Michigan State Univ., 1973.
- [18] W. W. Peterson, E. J. Weldon, Jr., *Error-correcting codes*, 2nd Ed., MIT Press, 1972, ISBN 0-26-216039-0.
- [19] R. R. Varshamov, *Estimate of the number of signals in error correcting codes*, Dokl. Acad. Nauk SSSR, **117** (1957), pp. 739 – 741.

- [20] Zhe-Xian Wan, *Quaternary codes*, World Scientific Publishing Co. Inc., 1997, ISBN 9-81-023274-8.

## APPENDIX A. TABLES FOR THE DIMENSIONS OF GREEDY LT CODES

The following are tables for the dimensions of LT codes for  $q = 3, 4, 5,$  and  $7$ . The column  $\mathbf{n}$  is the length,  $\mathbf{d}$  is the minimum distance. For  $q = 3, 5,$  and  $7$  the numbers in the parentheses next to dimensions of LT codes are the dimensions of best known linear codes with the same parameters taken from A. E. Brouwer's online catalog [1].

Table A.1 Dimensions of ternary greedy LT codes

<b>n</b>	<b>d=5</b>	<b>d=6</b>	<b>n</b>	<b>d=5</b>	<b>d=6</b>	<b>n</b>	<b>d=5</b>	<b>d=6</b>	<b>n</b>	<b>d=5</b>
<b>7</b>	2(2)	1(1)	<b>30</b>	21(22)	20(20)	<b>53</b>	43(44)	41(42)	<b>76</b>	65(67)
<b>8</b>	3(3)	2(2)	<b>31</b>	22(23)	21(21)	<b>54</b>	44(45)	42(43)	<b>77</b>	66(68)
<b>9</b>	4(4)	3(3)	<b>32</b>	23(24)	22(22)	<b>55</b>	45(46)	43(44)	<b>78</b>	67(69)
<b>10</b>	5(5)	4(4)	<b>33</b>	24(25)	23(23)	<b>56</b>	46(47)	44(45)	<b>79</b>	68(70)
<b>11</b>	6(6)	5(5)	<b>34</b>	25(26)	24(24)	<b>57</b>	47(48)	45(46)	<b>80</b>	69(71)
<b>12</b>	6(6)	6(6)	<b>35</b>	26(27)	24(25)	<b>58</b>	48(49)	46(47)	<b>81</b>	70(72)
<b>13</b>	6(7)	6(6)	<b>36</b>	27(28)	25(26)	<b>59</b>	49(50)	47(48)	<b>82</b>	71(73)
<b>14</b>	7(8)	6(7)	<b>37</b>	28(29)	26(27)	<b>60</b>	50(51)	48(49)	<b>83</b>	72(74)
<b>15</b>	8(8)	7(7)	<b>38</b>	29(30)	27(28)	<b>61</b>	51(52)	49(50)	<b>84</b>	73(75)
<b>16</b>	8(9)	8(8)	<b>39</b>	30(31)	28(28)	<b>62</b>	52(53)	50(51)	<b>85</b>	74(76)
<b>17</b>	9(10)	8(9)	<b>40</b>	31(32)	29(29)	<b>63</b>	53(54)		<b>86</b>	75(77)
<b>18</b>	10(11)	9(10)	<b>41</b>	32(33)	30(30)	<b>64</b>	54(55)		<b>87</b>	76(77)
<b>19</b>	11(12)	10(11)	<b>42</b>	33(33)	31(31)	<b>65</b>	55(56)		<b>88</b>	77(78)
<b>20</b>	12(13)	11(12)	<b>43</b>	34(34)	32(32)	<b>66</b>	56(57)		<b>89</b>	78(79)
<b>21</b>	13(14)	12(13)	<b>44</b>	35(35)	33(33)	<b>67</b>	57(58)		<b>90</b>	79(80)
<b>22</b>	14(15)	13(14)	<b>45</b>	36(36)	34(34)	<b>68</b>	58(59)		<b>91</b>	80(81)
<b>23</b>	15(16)	14(15)	<b>46</b>	37(37)	34(35)	<b>69</b>	59(60)		<b>92</b>	81(82)
<b>24</b>	16(17)	15(16)	<b>47</b>	37(38)	35(36)	<b>70</b>	60(61)		<b>93</b>	82(83)
<b>25</b>	16(18)	16(17)	<b>48</b>	38(39)	36(37)	<b>71</b>	61(62)		<b>94</b>	83(84)
<b>26</b>	17(19)	16(18)	<b>49</b>	39(40)	37(38)	<b>72</b>	61(63)		<b>95</b>	
<b>27</b>	18(20)	17(19)	<b>50</b>	40(41)	38(39)	<b>73</b>	62(64)		<b>96</b>	
<b>28</b>	19(20)	18(20)	<b>51</b>	41(42)	39(40)	<b>74</b>	63(65)		<b>97</b>	
<b>29</b>	20(21)	19(20)	<b>52</b>	42(43)	40(41)	<b>75</b>	64(66)		<b>98</b>	

Table A.2 Dimensions of quaternary greedy LT codes of Lee distance  $d$ 

<b>n</b>	<b>d=5</b>	<b>d=6</b>	<b>n</b>	<b>d=5</b>	<b>n</b>	<b>d=5</b>
<b>6</b>	2	2	<b>28</b>	21	<b>50</b>	42
<b>7</b>	3	3	<b>29</b>	22	<b>51</b>	43
<b>8</b>	4	4	<b>30</b>	23	<b>52</b>	44
<b>9</b>	4	4	<b>31</b>	24	<b>53</b>	45
<b>10</b>	5	4	<b>32</b>	25	<b>54</b>	46
<b>11</b>	6	5	<b>33</b>	26	<b>55</b>	47
<b>12</b>	7	6	<b>34</b>	27	<b>56</b>	48
<b>13</b>	8	7	<b>35</b>	28	<b>57</b>	49
<b>14</b>	9	8	<b>36</b>	29	<b>58</b>	50
<b>15</b>	9	8	<b>37</b>	30	<b>59</b>	51
<b>16</b>	10	9	<b>38</b>	31	<b>60</b>	52
<b>17</b>	11		<b>39</b>	32	<b>61</b>	53
<b>18</b>	12		<b>40</b>	33	<b>62</b>	54
<b>19</b>	13		<b>41</b>	34	<b>63</b>	55
<b>20</b>	14		<b>42</b>	35	<b>64</b>	56
<b>21</b>	15		<b>43</b>	36	<b>65</b>	57
<b>22</b>	16		<b>44</b>	37	<b>66</b>	58
<b>23</b>	17		<b>45</b>	38	<b>67</b>	59
<b>24</b>	18		<b>46</b>	38	<b>68</b>	60
<b>25</b>	19		<b>47</b>	39	<b>69</b>	61
<b>26</b>	19		<b>48</b>	40	<b>70</b>	62
<b>27</b>	20		<b>49</b>	41	<b>71</b>	63

Table A.3 Dimensions of 5-ary greedy LT codes

<b>n</b>	<b>d=5</b>	<b>d=6</b>	<b>d=7</b>	<b>d=8</b>	<b>n</b>	<b>d=5</b>	<b>d=6</b>	<b>n</b>	<b>d=5</b>	<b>n</b>	<b>d=5</b>
<b>7</b>	2(2)	1(1)	1(1)	-	<b>39</b>	32(32)	30(30)	<b>71</b>	63(63)	<b>103</b>	94(95)
<b>8</b>	3(3)	2(2)	1(1)	1(1)	<b>40</b>	33(33)	31(31)	<b>72</b>	64(64)	<b>104</b>	95(96)
<b>9</b>	4(4)	3(3)	2(2)	1(1)	<b>41</b>	33(34)	32(32)	<b>73</b>	65(65)	<b>105</b>	96(97)
<b>10</b>	5(5)	4(4)	3(3)	2(2)	<b>42</b>	34(35)	32(33)	<b>74</b>	65(66)	<b>106</b>	97(98)
<b>11</b>	6(6)	5(5)	4(4)	3(3)	<b>43</b>	35(36)	33(34)	<b>75</b>	66(67)	<b>107</b>	98(99)
<b>12</b>	6(7)	6(6)	4(4)	4(4)	<b>44</b>	36(37)	34(35)	<b>76</b>	67(68)	<b>108</b>	99(100)
<b>13</b>	7(7)	6(6)	5(5)	4(4)	<b>45</b>	37(37)	35(35)	<b>77</b>	68(69)	<b>109</b>	100(101)
<b>14</b>	8(8)	7(7)	6(6)	5(5)	<b>46</b>	38(38)	36(36)	<b>78</b>	69(70)	<b>110</b>	101(102)
<b>15</b>	9(9)	8(8)	7(7)	6(6)	<b>47</b>	39(39)	37(37)	<b>79</b>	70(71)	<b>111</b>	102(103)
<b>16</b>	10(10)	9(9)	7(8)	6(6)	<b>48</b>	40(40)	38(38)	<b>80</b>	71(72)	<b>112</b>	103(104)
<b>17</b>	11(11)	10(10)	8(9)	7(7)	<b>49</b>	41(41)	39(39)	<b>81</b>	72(73)	<b>113</b>	104(105)
<b>18</b>	12(12)	10(11)	9(9)	8(8)	<b>50</b>	42(42)	40(40)	<b>82</b>	73(74)	<b>114</b>	105(106)
<b>19</b>	13(13)	11(12)	10(10)	9(9)	<b>51</b>	43(43)	41(41)	<b>83</b>	74(75)	<b>115</b>	106(107)
<b>20</b>	14(14)	12(13)	11(11)	10(10)	<b>52</b>	44(44)	42(42)	<b>84</b>	75(76)	<b>116</b>	107(108)
<b>21</b>	15(15)	13(14)	12(12)	10(11)	<b>53</b>	45(45)	43(43)	<b>85</b>	76(77)	<b>117</b>	108(109)
<b>22</b>	16(16)	14(15)	13(13)	11(12)	<b>54</b>	46(46)	44(44)	<b>86</b>	77(78)	<b>118</b>	109(110)
<b>23</b>	16(17)	15(16)	13(14)		<b>55</b>	47(47)	45(45)	<b>87</b>	78(79)	<b>119</b>	110(111)
<b>24</b>	17(18)	16(17)	14(15)		<b>56</b>	48(48)	46(46)	<b>88</b>	79(80)	<b>120</b>	111(112)
<b>25</b>	18(19)	17(18)	15(16)		<b>57</b>	49(49)	47(47)	<b>89</b>	80(81)	<b>121</b>	112(113)
<b>26</b>	19(20)	18(19)	16(17)		<b>58</b>	50(50)	48(48)	<b>90</b>	81(82)	<b>122</b>	113(114)
<b>27</b>	20(21)	19(20)	17(18)		<b>59</b>	51(51)	49(49)	<b>91</b>	82(83)	<b>123</b>	114(115)
<b>28</b>	21(22)	19(21)			<b>60</b>	52(52)	50(50)	<b>92</b>	83(84)	<b>124</b>	115(116)
<b>29</b>	22(23)	20(21)			<b>61</b>	53(53)	51(51)	<b>93</b>	84(85)	<b>125</b>	116(116)
<b>30</b>	23(24)	21(22)			<b>62</b>	54(54)	52(52)	<b>94</b>	85(86)	<b>126</b>	117(117)
<b>31</b>	24(24)	22(23)			<b>63</b>	55(55)	53(53)	<b>95</b>	86(87)	<b>127</b>	118(118)
<b>32</b>	25(25)	23(24)			<b>64</b>	56(56)		<b>96</b>	87(88)	<b>128</b>	119(119)
<b>33</b>	26(26)	24(25)			<b>65</b>	57(57)		<b>97</b>	88(89)	<b>129</b>	120(120)
<b>34</b>	27(27)	25(25)			<b>66</b>	58(58)		<b>98</b>	89(90)	<b>130</b>	121(121)
<b>35</b>	28(28)	26(26)			<b>67</b>	59(59)		<b>99</b>	90(91)	<b>131</b>	122(122)
<b>36</b>	29(29)	27(27)			<b>68</b>	60(60)		<b>100</b>	91(92)	<b>132</b>	123(123)
<b>37</b>	30(30)	28(28)			<b>69</b>	61(61)		<b>101</b>	92(93)		
<b>38</b>	31(31)	29(29)			<b>70</b>	62(62)		<b>102</b>	93(94)		

Table A.4 Dimensions of 7-ary greedy LT codes

<b>n</b>	<b>d=5</b>	<b>d=6</b>	<b>n</b>	<b>d=5</b>	<b>d=6</b>	<b>n</b>	<b>d=5</b>	<b>d=6</b>	<b>n</b>	<b>d=5</b>
<b>6</b>	2(2)	1(1)	<b>31</b>	25(25)	23(23)	<b>56</b>	49	47	<b>81</b>	73
<b>7</b>	3(3)	2(2)	<b>32</b>	25(26)	<b>24(23)</b>	<b>57</b>	50	48	<b>82</b>	74
<b>8</b>	4(4)	3(3)	<b>33</b>	26(27)	<b>25(24)</b>	<b>58</b>	51	49	<b>83</b>	75
<b>9</b>	4(4)	3(3)	<b>34</b>	27(28)	<b>26(25)</b>	<b>59</b>	52	50	<b>84</b>	76
<b>10</b>	5(5)	4(4)	<b>35</b>	28(29)	<b>27(26)</b>	<b>60</b>	53		<b>85</b>	77
<b>11</b>	6(6)	5(5)	<b>36</b>	29(30)	<b>28(27)</b>	<b>61</b>	54		<b>86</b>	78
<b>12</b>	7(7)	6(6)	<b>37</b>	30(30)	<b>29(28)</b>	<b>62</b>	55		<b>87</b>	79
<b>13</b>	8(8)	7(7)	<b>38</b>	31(31)	<b>30(29)</b>	<b>63</b>	56		<b>88</b>	80
<b>14</b>	9(9)	8(8)	<b>39</b>	32(32)	30(30)	<b>64</b>	57		<b>89</b>	81
<b>15</b>	10(10)	8(9)	<b>40</b>	33(33)	31(31)	<b>65</b>	58		<b>90</b>	82
<b>16</b>	11(11)	9(9)	<b>41</b>	34(34)	32(32)	<b>66</b>	58		<b>91</b>	83
<b>17</b>	11(12)	10(10)	<b>42</b>	35(35)	33(33)	<b>67</b>	59		<b>92</b>	84
<b>18</b>	12(13)	11(11)	<b>43</b>	36(36)	34(34)	<b>68</b>	60		<b>93</b>	85
<b>19</b>	13(13)	12(12)	<b>44</b>	37(37)	35(35)	<b>69</b>	61		<b>94</b>	86
<b>20</b>	14(14)	13(13)	<b>45</b>	38(38)	36(36)	<b>70</b>	62		<b>95</b>	87
<b>21</b>	15(15)	14(14)	<b>46</b>	39(39)	37(37)	<b>71</b>	63		<b>96</b>	88
<b>22</b>	16(16)	15(15)	<b>47</b>	40(40)	38(38)	<b>72</b>	64		<b>97</b>	89
<b>23</b>	17(17)	15(16)	<b>48</b>	41(41)	39(39)	<b>73</b>	65		<b>98</b>	90
<b>24</b>	18(18)	16(17)	<b>49</b>	42(42)	40(40)	<b>74</b>	66		<b>99</b>	91
<b>25</b>	19(19)	17(18)	<b>50</b>	43(43)	41(41)	<b>75</b>	67		<b>100</b>	92
<b>26</b>	20(20)	18(19)	<b>51</b>	44	42	<b>76</b>	68		<b>101</b>	93
<b>27</b>	21(21)	19(20)	<b>52</b>	45	43	<b>77</b>	69		<b>102</b>	94
<b>28</b>	22(22)	20(21)	<b>53</b>	46	44	<b>78</b>	70		<b>103</b>	95
<b>29</b>	23(23)	21(22)	<b>54</b>	47	45	<b>79</b>	71			
<b>30</b>	24(24)	22(22)	<b>55</b>	48	46	<b>80</b>	72			

**APPENDIX B. FORTRAN SOURCE CODE FOR THE GREEDY LOOP  
TRANSVERSAL ALGORITHM**

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The following program computes the syndrome function for Loop  !
! Transversal codes by finding a (d-1)-independent set.          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```

program lt
implicit none
integer i,j,k,l,m,rank,count,answer,wt
integer t0,t1,t2,time,time_rate,time_max,stdout
integer n_old,q_old,d_old,picked
integer,parameter::q=7,n=16,d=6
integer, dimension (1:1150,1:n)::syn
integer, dimension (1:n,1:d-1):: S
integer, dimension (1:n)::y
integer, dimension(1:d-2)::config
intrinsic mod
call system_clock (t0,time_rate,time_max)
do i=1,d-1
  do j=1,n
    if (j==i) then
```

```

        syn(i,n-j+1)=1
    else
        syn(i,n-j+1)=0
    endif
enddo
enddo
syn(d,1:n-d+2)=0
syn(d,n-d+2:n)=1
count=d
y=syn(d,1:n)
picked=0

! This block must be removed if running the first time
open(unit=3,file='restore_Z7_d6',status='unknown')
read(3,'(1x,1i2,1x,1i3,1x,1i2,1x,1i4)') n_old,q_old,d_old,count
read(3,'(25i3)') config(1:d-2)
read(3,10) y(1:n)
do i=1,count
    read(3,10) syn(i,1:n)
enddo
picked=1
if ((n.ne.n_old).or.(q.ne.q_old).or.(d.ne.d_old)) then
    print*, 'Error: while reading from file'
endif
close (3) ! End of block

do while (y(1).lt.q)
    if (picked==1) then

```

```

    picked=0
    goto 100
else
    do i=1,d-2
        config(i)=i
    enddo
endif
answer=1
do while (answer==1.and.y(1).lt.q)
    call next (n,q,d, y,wt)
    if (wt==1.and.y(1).lt.q) then
        syn(count+1,1:n)=y
        count=count+1
        call next (n,q,d, y,wt)
    endif
    call sieve (n,q,d,y,count,syn, answer)
enddo
100  S(1:n,1)=y
do while (config(1).lt.count-d+4)
    call system_clock(t2,time_rate,time_max) ! checkpoint the state of the process
    if (t2.lt.t0) then
        t2=t2+time_max
    endif
    time=(t2-t0)/time_rate
    if (time.gt.172000) then
        open(unit=2,file='restore_Z7_d6',status='unknown')
        write(2,'(1x,1i2,1x,1i3,1x,1i2,1x,1i4)') n,q,d,count
        write(2,'(25i3)') config(1:d-2)
    endif
enddo

```

```

write(2,10) y(1:n)

do i=1,count
    write(2,10) syn(i,1:n)
enddo

close (2)

stop 0

endif ! end of checkpoint

do i=1,d-2

    S(1:n,i+1)=syn(config(i),1:n)

enddo

call row_reduce(n,q,d,S,n,d-1, rank)

if (rank.ge.d-1) then

    config(d-2)=config(d-2)+1

    i=d-2

    if (config(i).gt.count+i-d+2) then

        do while ((config(i)==count+i-d+3).and.(i.gt.1))

            config(i-1)=config(i-1)+1

            i=i-1

        enddo

        do k=i+1,d-2

            config(k)=config(k-1)+1

        enddo

    endif

else

    exit

endif

enddo

if (config(1)==count-d+4) then

```

```
syn(count+1,1:n)=y
count=count+1
open (unit=1,file='current_Z7_d6',status='unknown')
10 format(1x,75I1)
write(1,*) 'count=',count
do i=1,count
    write(1,10) syn(i,1:n)
enddo
close(unit=1)
endif
enddo
open (unit=1,file='Z7_d6',status='unknown')
write(1,*) 'count=',count
do i=1,count
    write(1,10) syn(i,1:n)
enddo
end
```

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The following subroutine will compute lexicographically the next !
! vector to v of weight d-1                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine next (m,q,d, v,weight)
integer m,q,i,d,weight
integer, dimension(1:m)::v
do while(v(1).lt.q)
  call increase(m,q, v)
  call weigh_H(m,q,v, weight)
  if (weight.ge.d-1) then
    exit
  else if (weight==1) then
    i=1
    do while (v(i)==0)
      i=i+1
    enddo
    if (v(i)==1) then
      exit
    endif
  endif
enddo
end

```

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The following subroutine will seive out the vectors v such that !
! av+bs has a Hamming weight less than d-2, where s=syn(b_i), and !
! a,b in F_q !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine sieve (n,q,d,y,count,syndroms, ans)
integer n,q,d,nonzero,i,ans,count,wt,j,k,dif
integer, dimension(1:n)::y,s
integer, dimension(1:1150,1:n)::syndroms
external weigh_H
do i=1,count
  ans=0
  call weigh_H (n,q,syndroms(i,1:n),wt)
  if (wt.gt.1) then
    do j=1,q-1
      dif=0
      do k=1,n
        s(k)=j*syndroms(i,k)+y(k)
        if (s(k).ne.0) then
          dif=dif+1
        endif
      enddo
      if (dif.lt.d-2) then
        ans=1
        exit
      endif
    enddo
  endif
enddo
endif

```

```
if (ans==1) then
    exit
endif
enddo
end
```

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The following subroutine will compute a row-reduced echelon form !
! of matrix A and return the column rank of that matrix in r      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine row_reduce(n,q,d,A,m,k, r)
integer m,k,q,d,i,j,md,md_prev,g_c_d,r,i1,i2
integer, dimension (1:m,1:k+1)::B
integer, dimension (1:n,1:d-1)::A
integer, dimension (0:q-1)::inv,div
integer, dimension (1:k)::v,prod
external multiply, gcd
! finding the inverses and annihilators for the elements of Z_q and
! storing them in arrays "inv" and "div"
inv(0:q-1)=q+1
div(0:q-1)=0      !initialize
do i=1,q-1
  do j=1,q-1
    ! finding the inverse of element i in Z_q (if exists)
    md=1
    do l=1,j ! computing i^j(mod q) memorizing i^{j-1} in variable md_prev
      md_prev=md
      md=mod(md*i,q)
    enddo
    if (md==1) then
      inv(i)=md_prev
      exit
    endif
  enddo
enddo

```

```

! finding the annihilator of i (if exists)
call gcd(i,q, g_c_d)
if (g_c_d.ne.1) then
    div(i)=q/g_c_d
endif
enddo
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
do i=1,m
    B(i,1:k)=A(i,1:k)
enddo
! The matrices A and B have m rows and k columns
! For each column pivot the entries and compute the rank of the resulting matrix
do j=1,k
! For each row either multiply it by pivot's inverse or by its annihilator
do i=j,m
    if (inv(B(i,j)).ne.q+1) then
        call multiply (k,q,inv(B(i,j)),B(i,1:k), B(i,1:k))
    else if (B(i,j).ne.0) then
        call multiply (k,q,div(B(i,j)),B(i,1:k), B(i,1:k))
    endif
enddo
if (B(j,j).ne.1) then
! Find a row with 1 in that position (if any) and exchange it with the original
do i=j+1,m
    if (B(i,j)==1) then
        v=B(j,1:k)
        B(j,1:k)=B(i,1:k)
        B(i,1:k)=v
    endif
enddo
endif

```

```

        exit
    endif
enddo
endif
! get 0's under the diagonal entries
if (B(j,j)==1) then
    do i=j+1,m
        if (B(i,j)==1) then
            call subtract (k,q,B(i,1:k),B(j,1:k), B(i,1:k))
        endif
    enddo
    ! compute the rank by putting 1 as the (k+1)-th entry of the row if the row is
    ! different from zero
    r=0
    do i1=1,m
        B(i1,k+1)=0
        do i2=1,k
            if (B(i1,i2).ne.0) then
                B(i1,k+1)=1
                exit
            endif
        enddo
        r=r+B(i1,k+1)
    enddo
    ! If rank is less than d-1 then columns of A are not independent, so exit
    if (r.lt.d-1) then
        exit
    endif
endif

```

```
endif  
enddo  
end
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The following subroutine computes the greatest common divisor of !
! a and b !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine gcd (a,b, g)
integer a,b,g,c,r,a1,b1
intrinsic mod

a1=a
b1=b

if (a1.gt.b1) then
    c=a1
    a1=b1
    b1=c
endif

! now we have b>a so we use Euclid's algorithm

r=1

do while (r.ne.0)
    r=mod(b1,a1)
    b1=a1
    a1=r
enddo

g=b1

end
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   The following subroutine multiplies vector by scalar a in Z_q  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine multiply (k,q,a,vector, product)
integer a,q,k,l
integer, dimension(1:k)::vector,product
do i=1,k
  product(i)=vector(i)*a
  do while (product(i).ge.q)
    product(i)=product(i)-q
  enddo
enddo
enddo
end
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The following subroutine computes the difference x-y for vectors !
! x,y over Z_q !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine subtract (n,q,x,y, difference)
integer n,q,i,a
integer, dimension(1:n)::x,y,difference
do i=1,n
  a=x(i)-y(i)
  if (a.lt.0) then
    difference(i)=q+a
  else
    difference(i)=a
  endif
enddo
end
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The following subroutine will increase the given vector          !
! lexicografically by one (over Z_q)                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine increase (m,q,x)
integer m,q,j
integer, dimension(1:m)::x
x(m)=x(m)+1
j=m
do while(x(j)==q.and.j.gt.1)
  x(j)=0
  x(j-1)=x(j-1)+1
  j=j-1
enddo
end
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The following subroutine computes the Hamming weight of a given !
! vector !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine weigh_H (m,q,vector,weight)
integer m,weight,index,q
integer, dimension(1:m)::vector
weight=0
do index=1,m
  if(vector(index).ne.0) then
    weight=weight+1
  endif
enddo
end
```