# Generalized Hamming weights

Beugels, R.G.M.

*Award date:*
2006

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

# Generalized Hamming weights

by
R.G.M. Beugels

Supervisors:
Prof. dr. ir. Henk van Tilborg
dr. ir. Henk Hollmann

*Eindhoven, August 2006*

# Contents

# Chapter 1

# Introduction

One of the major problems in coding theory is finding the minimum distance of an error-correcting code, since this number determines its error-correcting capabilities. This is known to be a hard problem for arbitrary linear codes [5]. Of course there have been attempts to give lower bounds on the minimum distance. One of the methods, developed by van Lint and Wilson [22], is called *shifting* and is suited for cyclic codes. Upon closer examination of the shifting technique it turns out that, with some minor modifications, it can also be applied to all linear codes. The initial goal of this project was to further investigate this generalization. However, in the early stages we stumbled upon generalized Hamming weights of certain BCH codes. Since we discovered a new result in this area, we decided to continue the project in this new direction.

In Chapter 2 we mention the basic assumptions we use throughout the document. In the remainder of the chapter we list definitions and theorems related to generalized Hamming weights, trace functions and solving quadratic equations in a finite field of characteristic 2.

Although we did not spend that much time on shifting, we did make a bit of progress. The most important results on this matter are collected in Chapter 3.

The main subject, generalized Hamming weights, is treated in Chapter 4. In the discussion we do not have a specific code in mind; the results hold for linear codes in general. The chapter starts with an overview of the most important known results. After that we give some theorems we proved ourselves, most of which turned out to be already proved by others. We conclude the chapter by mentioning some applications of generalized Hamming weights.

We will focus on generalized Hamming weights of primitive narrow-sense 2-error-correcting codes in Chapter 5. There we give a somewhat more convenient proof of already known generalized Hamming weights and add a new result.

In the last chapter we discuss an algorithm to compute generalized Hamming weights for arbitrary linear codes.

# Chapter 2

# Definitions

## 2.1 Introduction

In this chapter we give the definitions of frequently used concepts and mention several theorems. Most of it is probably already known to a reader familiar with the subject. The next section is a must to read, since it is about the assumptions we make out of convenience.

## 2.2 Some basic assumptions

The complement of a set $X$ is most commonly denoted by $X^c$. However, since we will frequently talk about codewords $c$ and codes $C$, it is wise to use another notation to avoid confusion. We picked $\overline{X}$ to refer to the complement of a set $X$.

Whenever we talk about a code $C$, we tacitly assume that $C$ is a binary linear code.

By BCH(2,m) we denote the primitive narrow-sense 2-error correcting BCH code of length $2^m-1$. The extended BCH(2,m) code is referred to as EBCH(2,m).

## 2.3 Weights of codewords and codes

In this section we give all definitions needed to understand the main topic of this report: generalized Hamming weights. We also state the (generalized) Griesmer bound.

**Definition 2.1.** *The support $\chi(c)$ of a vector $c = (c_1, \ldots, c_n) \in \mathbb{F}_2^n$ is the set of nonzero positions of $c$,*

$$\chi(c) = \{i \mid c_i \neq 0\}. \tag{2.1}$$

**Definition 2.2.** *The support of a space $V \subseteq \mathbb{F}_2^n$ is simply defined as the union of the support of its elements:*

$$\chi(V) = \bigcup_{v \in V} \chi(v). \tag{2.2}$$

If $V$ is a linear subspace we also have the following alternative interpretation: the support is the set of indices where at least one of the basis vectors has a nonzero entry.

We will frequently talk about the size of the support of a vector or space, which we shorten to "weight" out of convenience.

**Definition 2.3.** *For any vector $c \in \mathbb{F}_2^n$ the weight $w(c)$ is defined as*

$$w(c) = |\chi(c)|. \tag{2.3}$$

*Similarly, we define for any (sub)code $C \subset \mathbb{F}_2^n$ its weight $w(C)$ as*

$$w(C) = |\chi(C)|. \tag{2.4}$$

With these definitions we introduce the concept of generalized Hamming weights.

**Definition 2.4.** *The $r$-th generalized Hamming weight $d_r(C)$ of a linear code $C$is defined as*

$$d_r(C) = \min\{w(D) \mid D \subseteq C, \dim(D) = r\}. \tag{2.5}$$

The minimum distance of a code corresponds with $d_1(C)$. The set of generalized Hamming weights make up the weight hierarchy of a code.

**Definition 2.5.** *The weight hierarchy of a $k$-dimensional code $C$ is defined as the set*

$$\{d_r(C) \mid 1 \leq r \leq k\}. \tag{2.6}$$

The Griesmer bound gives a lower bound on the length of a linear code and thus also on generalized Hamming weights.

**Theorem 2.6** (Griesmer bound, [14])**.** *Let $C$ be a binary $[n, k, d]$ code, then*

$$n \geq g(k, d) := \sum_{i=0}^{k-1} \left\lceil \frac{d}{2^i} \right\rceil. \tag{2.7}$$

Shortly after the notion of generalized Hamming weights was introduced, Helleseth, Kløve and Ytrehus [15] published a generalization of the Griesmer bound. With this result we can use any $d_r(C)$, not just $d_1(C)$, to lower bound the weight of higher dimensional subcodes.

**Theorem 2.7** ([15], theorem 5)**.** *Let $C$ be a binary $[n,k,d]$ code and $1 \leq r \leq k$, then*

$$n \geq d_r + \sum_{i=1}^{k-r} \left\lceil \frac{d_r}{2^i(2^r-1)} \right\rceil . \tag{2.8}$$

Finally, we will need a theorem by Dodunekov and Manev on the weight of codewords spanning the whole code.

**Theorem 2.8** ([8])**.** *If $C$ is an $[n,k,d]$ code of length $n = g(k,d)+t$, then $C$ is spanned by codewords of weight at most $d+t$.*

## 2.4 Description of some codes

### 2.4.1 Cyclic codes

In this paragraph we give a short overview of cyclic codes. It is certainly not intended as a complete introduction to the theory of such codes. For that we refer to [29] or any other book on coding heory.

We call a code $C$ of length $n$ *cyclic* if for every codeword $c = (c_0, c_1, \ldots, c_{n-1})$ its cyclic shift $(c_{n-1}, c_0, c_1, \ldots, c_{n-2})$ is also a codeword. The codeword $c$ can also be considered as a polynomial $c(X) = c_0 + c_1X + \ldots + c_{n-1}X^{n-1}$ in the quotient ring of polynomials $\mathbb{F}_2[X]/(X^n-1)$; the cyclic shift then corresponds with multiplying by $X$. This property makes the code an ideal in the polynomial ring. It is generated by a *generator polynomial* $g(X)$, that is, every codeword $c(X)$ can be written as $c(X) = g(X)a(X)$ for some polynomial $a(X)$. Since the all zero vector is always a codeword, we have $X^n - 1 = g(X)h(X)$. The polynomial $h(X)$ is called the *parity check polynomial* of $C$. The dual of $C$ is cyclic as well and its generator polynomial is given by $X^n h(1/X)$. If $n$ is odd, which we will assume from now on, there is an extension field $\mathbb{F}_{2^m}$, for some $m$, of $\mathbb{F}_2$ containing a primitive $n$-th root of unity $\alpha$. The polynomial $X^n - 1$ factorizes completely in linear factors in $\mathbb{F}_{2^m}$. As a consequence, all zeros of $g(X)$ are powers of $\alpha$. The *defining set* $I \subseteq \{0, 1, \ldots, n-1\}$ of a cyclic code is the set of powers of $\alpha$ that are a zero of the generator polynomial. If the defining set $I$ contains $d_{BCH} - 1$ consecutive integers (modulo $n$), then $C$ is called a BCH code of designed distance $d_{BCH}$. If these consecutive integers happen to be $\{1, \ldots, d_{BCH} - 1\}$, we call the code a *narrow-sense* BCH code. If $n = 2^m - 1$ for some $m$, the code is called a *primitive* BCH code. We will devote an entire chapter to primitive narrow-sense BCH codes of designed distance 5 (or equivalently: 2-error correcting). Out of convenience we will refer to such codes of length $2^m - 1$ as BCH(2,m). The extended BCH(2,m) is denoted by EBCH(2,m).

### 2.4.2 Reed-Muller codes

Codewords of a Reed-Muller code can be described by binary polynomials $f : \mathbb{F}_2^m \to \mathbb{F}_2$ in which each variable $X_i$ occurs at most to the power one. Let $(i)_2$

denote the binary representation of $i$. If $0 \leq i < 2^m$, we can view $(i)_2$ as a vector in $\mathbb{F}_2^m$.

**Definition 2.9.** *The r-th Reed-Muller code $\mathcal{RM}(r,m)$ of length $n = 2^m$ is defined by*

$$\mathcal{RM}(r,m) = \{(f((0)_2), f((1)_2), \ldots, f((n-1)_2)) \mid \deg(f) \leq r\}. \tag{2.9}$$

The degree of a codeword is simply the degree of the corresponding polynomial. The minimum distance of $\mathcal{RM}(r,m)$ is equal to $2^{m-r}$ and its dual is $\mathcal{RM}(m-r-1,m)$.

## 2.5 The trace mapping

There is a more general definition of the trace mapping, but we only need the so called absolute trace. It is defined as follows.

**Definition 2.10.** *For $x \in \mathbb{F}_{2^m}$ the trace of $x$ is defined by*

$$Tr(x) = x + x^2 + x^{2^2} + \ldots + x^{2^{m-1}}. \tag{2.10}$$

The trace is a linear mapping from $\mathbb{F}_{2^m}$ to $\mathbb{F}_2$ because $\mathrm{Tr}(x)^2 = \mathrm{Tr}(x)$ and $\mathrm{Tr}(x+y) = \mathrm{Tr}(x) + \mathrm{Tr}(y)$. It plays a major role in solving quadratic equations as we will show. But before we are able to do so we have to introduce some more notations and theorems.

The elements of $\mathbb{F}_{2^m}$ can be partitioned into two sets $\mathrm{Tr}_0$ and $\mathrm{Tr}_1$ according to their trace.

**Definition 2.11.**

$$Tr_0 = \{x \in \mathbb{F}_{2^m} \mid Tr(x) = 0\} \tag{2.11}$$

*and*

$$Tr_1 = \{x \in \mathbb{F}_{2^m} \mid Tr(x) = 1\}. \tag{2.12}$$

A quadratic equation is an equation of the form

$$ax^2 + bx + c = 0 \tag{2.13}$$

where $a \in \mathbb{F}_{2^m}^*$ and $b, c \in \mathbb{F}_{2^m}$. Our goal is to find all solutions in $\mathbb{F}_{2^m}$ of this equation. In the next lemma we will look at quadratic equations of the form $x^2 + x + c = 0$ and give a necessary and sufficient condition for the existence of a solution. Using this lemma we explain how to solve any quadratic equation.

**Lemma 2.12.** *The quadratic equation*

$$x^2 + x + c = 0 \tag{2.14}$$

*with $c \in \mathbb{F}_{2^m}$ has a solution if and only if $Tr(c) = 0$.*

*Proof.* Consider the linear mapping $\phi : y \mapsto y^2 + y$. If $\phi(a) = \phi(b)$ for some elements $a, b \in \mathbb{F}_{2^m}$, then $\phi(a + b) = 0$. Since there are no elements of order two, this is only true if $a + b = 0$ or $a + b = 1$. This means that $\mathbb{F}_{2^m}$ can be partitioned into disjoint pairs of elements and that each pair is mapped to a different element. Hence we know that $|\phi(\mathbb{F}_{2^m})| = 2^{m-1}$. Every element $c \in \phi(\mathbb{F}_{2^m})$ can be written as $c = a^2 + a$ for some $a \in \mathbb{F}_{2^m}$. Applying the trace function to both sides of this equation gives $\mathrm{Tr}(c) = \mathrm{Tr}(a^2 + a) = \mathrm{Tr}(a)^2 + \mathrm{Tr}(a) = 0$. It follows that $\phi(\mathbb{F}_{2^m})$ is a subset of $\mathrm{Tr}_0$. Because $\mathrm{Tr}(x)$ is a polynomial of degree $2^{m-1}$ it has at most $2^{m-1}$ zeros. But we already have that many zeros, namely the elements of $\phi(\mathbb{F}_{2^m})$. Therefore we conclude that $\phi(\mathbb{F}_{2^m}) = \mathrm{Tr}_0$. The equation has a solution if and only if $\mathrm{Tr}(c) = 0$. □

We refer to the solutions of the quadratic equations in Lemma 2.12 as set of roots $\mathrm{RT}(c)$.

**Definition 2.13.** *For any $c \in \mathbb{F}_{2^m}$, the solutions to the quadratic equation $x^2 + x + c$ are given by*

$$RT_{2^m}(c) := \{x \in \mathbb{F}_{2^m} \mid x^2 + x + c = 0\}. \tag{2.15}$$

The next corollary follows from this definition and Lemma 2.12.

**Corollary 2.14.** *For $c \in \mathbb{F}_{2^m}$ we have $RT_{2^m}(c) \neq \emptyset$ if and only if $Tr(c) = 0$.*

Now we turn our attention back to the general quadratic equation.

**Lemma 2.15.** *The solutions to the quadratic equation*

$$ax^2 + bx + c = 0 \tag{2.16}$$

*where $a \in \mathbb{F}_{2^m}^*$ and $b, c \in \mathbb{F}_{2^m}$, are given by*

$$\begin{array}{ll} (\frac{c}{a})^{2^{m-1}} & , \text{ if } b = 0 \\ \frac{b}{a} RT_{2^m}\left(\frac{ac}{b^2}\right) & , \text{ if } b \neq 0. \end{array} \tag{2.17}$$

*Proof.* If $b = 0$ we easily get $x^2 = c/a$ and thus $x = (x^2)^{2^{m-1}} = (c/a)^{2^{m-1}}$. Now assume that $b \neq 0$. Multiplying the equation with $a/b^2$ gives $\frac{a^2}{b^2}x^2 + \frac{a}{b}x + \frac{ac}{b^2} = 0$. Substituting $y = \frac{a}{b}x$ now yields the familiar form $y^2 + y + \frac{ac}{b^2} = 0$. The solutions to this equation are $\mathrm{RT}_{2^m}(\frac{ac}{b^2})$, by Definition 2.13. The solutions to the original equation are thus $\frac{b}{a}\mathrm{RT}(\frac{ac}{b^2})$. □

The next lemma has nothing to do with quadratic equations, but we need it later on when we will count codewords.

**Lemma 2.16.** *The number of elements $x \in \mathbb{F}_{2^m}$ for which $Tr(x) = Tr(\lambda x)$ is*

$$|\{x \mid Tr(x) = Tr(\lambda x) = 0\}| = \begin{cases} 2^{m-1} & \text{if } \lambda \in \{0, 1\} \\ 2^{m-2} & \text{if } \lambda \notin \{0, 1\}. \end{cases} \tag{2.18}$$

*Proof.* The problem is to count how many zeros the polynomials $\mathrm{Tr}(x)$ and $\mathrm{Tr}(\lambda x)$ have in common. This is easy if $\lambda \in \{0, 1\}$, since those zeros are just the zeros of $\mathrm{Tr}(x)$. There are $2^{m-1}$ of them. Now consider the case that $\lambda \notin \{0, 1\}$ and assume that $s$ is a zero of $\mathrm{Tr}(x)$. If $s$ is a zero of $\mathrm{Tr}(\lambda x)$ as well, then it is also a zero of $f(x) := \lambda^{2^{m-1}}\mathrm{Tr}(x) + \mathrm{Tr}(\lambda x)$. The degree of $f(x)$ is $2^{m-2}$, so there are at most $2^{m-2}$ common zeros. If $s$ is not a zero of $\mathrm{Tr}(\lambda x)$ then it is a zero of $\mathrm{Tr}(\lambda x) + 1$ and thus also of $g(x) := \lambda^{2^{m-1}}\mathrm{Tr}(x) + \mathrm{Tr}(\lambda x) + 1$. The degree of $g(x)$ is $2^{m-2}$ as well. Every $s$ is a zero of either $f(x)$ or $g(x)$, so it follows that both $f(x)$ and $g(x)$ must have precisely $2^{m-2}$ distinct zeros. And hence we have proven that $\mathrm{Tr}(x)$ and $\mathrm{Tr}(\lambda x)$ have $2^{m-2}$ zeros in common when $\lambda \notin \{0, 1\}$. □

## 2.6 Additive characters

An additive character is a mapping $\psi : \mathbb{F}_{2^m} \to \mathbb{C} \setminus \{0\}$ with the property that $\psi(a + b) = \psi(a)\psi(b)$ for any $a, b \in \mathbb{F}_{2^m}$.

Every character can be described by

$$\psi_a(x) = (-1)^{\mathrm{Tr}(ax)} \tag{2.19}$$

for some $a \in \mathbb{F}_{2^m}$. We call $\psi_0$ the trivial character because it maps every element to 1.

**Lemma 2.17.** *The character sum has the property that*

$$\sum_{x \in \mathbb{F}_{2^m}} \psi_a(x) = \begin{cases} 2^m & \text{if } a = 0 \\ 0 & \text{if } a \neq 0. \end{cases} \tag{2.20}$$

For more information on characters see, for example, [21].

# Chapter 3

# Shifting

## 3.1 Introduction

In the mid eighties of the previous century van Lint and Wilson [22] introduced a new procedure, called shifting, to compute a lower bound on the minimum distance of a cyclic code. It is still one of the best methods these days. However, there is nothing known on how to do this shifting efficiently. The only viable option seems to be a brute force approach.

Hollmann [17] has recently extended the shifting method to the setting of arbitrarely linear codes. To study such generalization was the initial goal of the project.

First we will give a stripped down version of the shifting method. Then we give a tight lower bound on the minimum distance of Reed-Muller codes. This lower bound is not very surprising, but the technique (shifting) is.

## 3.2 Some preliminary definitions

Let $\mathbb{F}$ be a field and let $a, b \in \mathbb{F}^n$, then

$$a * b = (a_1 b_1, \ldots, a_n b_n). \tag{3.1}$$

If $B \subset \mathbb{F}^n$, then

$$a * B = \{a * b \mid b \in B\}. \tag{3.2}$$

Let $c \in \mathbb{F}^n$ and suppose that $\chi(c) = \{i_1, \ldots, i_m\}$, then we define the restriction of $a \in \mathbb{F}^n$ to $c$ as

$$a|_c = (a_{i_1}, \ldots, a_{i_m}). \tag{3.3}$$

We can also define the restriction of $A \subset \mathbb{F}^n$ to $c$

$$A|_c = \{a|_c \mid a \in A\}. \tag{3.4}$$

## 3.3 A couple of theorems

The goal of shifting is to find a lower bound on the weight of a codeword $c$. This is done by constructing a subspace of $\mathbb{F}^n$ whose dimension is as large as possible when restricted to the support of $c$. We find such a subspace using the notion of independent sets with respect to $c$. An inductive definition follows.

1. (*empty set*): $\emptyset$ is independent with respect to $c$.
2. (*extension*): If $A$ is independent with respect to $c$, $A \subset c^\perp$ and $b \notin c^\perp$ then $A \cup \{b\}$ is independent with respect to $c$.
3. (*shifting*): If $A$ is independent with respect to $c$ and there is a $d \in \mathbb{F}^n$ with $d_i \neq 0$ for all $1 \leq i \leq n$, then $d * A$ is independent with respect to $c$.

**Theorem 3.1.** *Let $c \in \mathbb{F}^n$, then the weight (number of non zeroes) of $c$ satisfies*

$$w(c) \geq |A| \tag{3.5}$$

*for every set $A \subseteq \mathbb{F}^n$ that is independent with respect to $c$.*

*Proof.* By induction on the size of $A$ we will prove that if $A$ is independent with respect to $c$, then the vectors in $A|_c$ are linearly independent over $\mathbb{F}$. Since the dimension of a subspace of $\mathbb{F}^{w(c)}$ is at most $w(c)$, the result follows.

The statement is obviously true if $A$ is the empty set. Now suppose that $A \subset c^\perp$ is independent with respect to $c$ and $b \notin c^\perp$. Since $b$ is not orthogonal to $c$, it is certainly not in the span of $A$, so $(A \cup \{b\})|_c$ is a linear independent set of vectors. Assume that $A$ is independent with respect to $c$ and let $d \in \mathbb{F}^n$ with $d_i \neq 0$ for all $i$. The set $(d * A)|_c$ is obtained by applying the linear transformation represented by the diagonalmatrix $\Lambda(d|_c)$ to the set $A|_c$. Since this mapping is invertible, the set $(d * A)|_c$ is linearly independent as well. □

By imposing additional requirements on the independent sets, it might become easier to shift. For example, we could require that all elements of an independent set are in the group generated by the vector $\rho = (1, \alpha, \ldots, \alpha^{n-1})$, where $\alpha$ is a primitive $n$-th root of unity. The choice of this group is well suited for cyclic codes since orthogonality to $\rho^i$ simply means that $\alpha^i$ is a zero of the codeword, when viewed as a polynomial. It is this setting in which the concept of shifting was introduced by van Lint and Wilson [22].

In the following theorem we will show that there exists a group with which we can give a sharp lower bound on the minimum distance for Reed-Muller codes.

**Theorem 3.2** ([17])**.** *For any $q = 2^k > 2^m = n$ there exists a vector $\rho \in \mathbb{F}_q^n$, whose elements are all distinct, such that the dual of $\mathcal{RM}(r, m)$ contains $\rho^0, \rho, \ldots, \rho^{2^{m-r}-2}$. This implies that the minimum weight of $\mathcal{RM}(r, m)$ is at least $2^{m-r}$.*

*Proof.* Define the first degree binary polynomials $f_i(x_1, \ldots, x_m) := x_i$ for $1 \leq i \leq m$ and $f_0(x_1, \ldots, x_m) = 1$. These polynomials then give rise to codewords $v_i = (f_i((0)_2), \ldots, f_i((n-1)_2)) \in \mathcal{RM}(r, m)$ (see Definition 2.9). Let $\alpha$ be a primitive element in $\mathbb{F}_q$. We take $\rho = \sum_{j=0}^{m} \alpha^j v_j$. Element $\rho_i$ is the inner product of $(1, \alpha, \ldots, \alpha^m)$ and $(i)_2$. Since all the binary representations are different and $m < k$, all elements of $\rho$ are distinct. Now the only thing left to prove is that all powers up to $2^{m-r} - 2$ of $\rho$ are in the dual, $\mathcal{RM}(m-r-1, m)$. The degree of $\rho^s$ for any $0 \leq s \leq 2^{m-r} - 2$ therefore has to be at most $m-r-1$. We can write any integer $s$ as $s = \sum_{i \in I_s} 2^i$. If $s < 2^{m-r} - 1$, we have $|I_s| < m-r$. We upperbound the degree of $\rho^s$ as follows.

$$\begin{aligned}
\deg(\rho^s) &= \deg\left(\left(\sum_{j=0}^{m} \alpha^j v_j\right)^s\right) \\
&= \deg\left(\left(\sum_{j=0}^{m} \alpha^j v_j\right)^{\sum_{i \in I_s} 2^i}\right) \\
&= \deg\left(\prod_{i \in I_s} \left(\sum_{j=0}^{m} \alpha^j v_j\right)^{2^i}\right) \\
&\leq \sum_{i \in I_s} \deg\left(\sum_{j=1}^{m} \alpha^{j 2^i} v_j\right) \\
&= \sum_{i \in I_s} 1 \\
&\leq m - r - 1.
\end{aligned}$$

So indeed we have $\rho^s \in RM(m-r-1, m)$. Let $A$ be the $(2^{m-r}-1) \times 2^m$-matrix whose $(i+1)$-th row is $\rho^i$. If we take any $2^{m-r} - 1$ columns of $A$ we get an invertible Vandermonde matrix, since all elements of $\rho$ are distinct. Hence there is not a codeword of weight at most $2^{m-r} - 1$, and thus the minimum weight is at least $2^{m-r}$. □

Kasami, Lin and Peterson [19] proved in a very similar way that Reed-Muller codes punctured on the coordinate corresponding to $(0)_2$ are equivalent to primitive narrow-sense BCH codes with designed distance equal to the distance of the Reed-Muller code.

At the start of the project, we wanted, given a code $C$, to construct a group generated by a single element such that this group would give a sharp lower bound on the minimum distance. The following theorem states that this is equivalent to finding a subcode of a certain cyclic code.

**Theorem 3.3.** *Let $C$ be a binary $[n, k]$-code and let $\rho \subset \mathbb{F}_{2^m}^n$ be a vector whose components are distinct and non-zero. Define $D$ as the primitive cyclic code of*

*length* $2^m - 1$ *with defining set*

$$I = \{i \mid \rho^i \in C^\perp\}. \tag{3.6}$$

*Then, after padding the codewords in* $C$ *with sufficiently many zeroes, the code* $C$ *is equivalent to a subcode of* $D$*.*

*Proof.* We extend the codewords of $C$ by adding $2^m - 1 - n$ zeros at the end. We also construct a vector $\hat{\rho}$. The first part of $\hat{\rho}$ is simply the vector $\rho$. The last part consists of all elements of $\mathbb{F}_{2^m}^*$ that are not already in $\rho$. Orthogonality of the codewords to $\hat{\rho}^i$, for $i \in I$, is maintained since it only depends on the first $n$ coordinates. After a suitable coordinate permutation, we have $\hat{\rho} = (1, \alpha, \ldots, \alpha^{n-1})$. The zero-padded and permuted codewords are thus codewords of $D$. □

# Chapter 4

# Generalized Hamming weights

## 4.1 Introduction

The initial goal of the project was to investigate a generalization of shifting (see Chapter 3). As a first example, we applied this shifting technique to a certain small code and thereby concluded that in fact we found a generalized Hamming weight of a 2-error correcting BCH code. We pursued the subject and in this chapter we collect the results we obtained for general linear codes. In Chapter 5 we will focus on primitive narrow-sense two error correcting BCH codes.

The main result, which has already been proved by Barbero and Munuera [3], that we will work up to is a minor extension of a theorem by Wei [32] on the relation between the weight hierarchies of a code and its dual. We achieve the result by introducing the concept of *optimal* subcodes and subsequently show a connection between optimal subcodes of the code and optimal subcodes in its dual. To do so we need a result on the dimension of a shortened code. We did not find this lemma stated explicitly in the literature, but it is an immediate consequence of some work by Forney [11] and Simonis [28].

Another result that we will present is a relation between the generalized Hamming weights of a code and its extended code in case the automorphism group of the extended code is transitive. An example of such codes are the primitive narrow-sense BCH codes.

We start by giving an overview of the known results concerning linear codes in general.

## 4.2 Known results

In his paper on generalized Hamming weights Wei [32] states a couple of theorems. The first one says that the weights in the hierarchy form a strictly in-

creasing sequence. It follows immediately by shortening a subcode of minimum weight.

**Theorem 4.1** (monotonicity,[32]). *For an $[n, k]$ code $C$ with $k > 0$, we have*

$$1 \leq d_1(C) < d_2(C) < \ldots < d_k(C) \leq n. \tag{4.1}$$

The most important theorem in his paper concerns the relation between the weight hierarchies of the code and its dual.

**Theorem 4.2** (duality,[32]). *Let $C$ be a $[n, k]$ code. Then*

$$\{d_r(C)|1 \leq r \leq k\} = \{1, 2, \ldots, n\} \setminus \{n + 1 - d_r(C^\perp)|1 \leq r \leq n\}. \tag{4.2}$$

This theorem basically says that the set $\{1, 2, \ldots, n\}$ can be partitioned into two disjoint subsets: one of them corresponding with the weight hierarchy of the code, the other with the weight hierarchy of the dual code. From this perspective, it is clear that, given $d_r(C)$ for some $r$, there are precisely $d_r(C)-r$ generalized Hamming weights $d_s(C^\perp)$ of the dual for which $n + 1 - d_s(C^\perp) < d_r(C)$. The other weights of the dual satisfy $n+1-d_s(C^\perp) > d_r(C)$. From this observation we get the following corollary by Feng, Tzeng and Wei.

**Corollary 4.3** ([9]). *Let $0 \leq r \leq k$. If $d_r(C) \geq x > r$, then*

$$d_{n-k-x+r+1}(C^\perp) \geq n + 2 - x. \tag{4.3}$$

*If $d_r(C) \leq x < n$, then*

$$d_{n-k-x+r}(C^\perp) \leq n - x. \tag{4.4}$$

## 4.3 Some preliminary definitions

We define $C_I$ as the collection of codewords from $C$ that have a zero in the positions indexed by $I$, that is,

$$C_I := \{c \in C \mid \chi(c) \cap I = \emptyset\}. \tag{4.5}$$

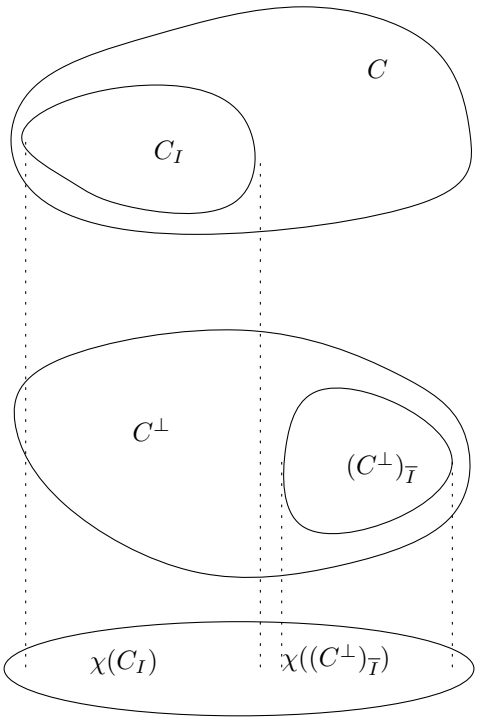
We will refer to $C_I$ as the code obtained from $C$ by shortening on $I$. The difference with the usual notion of shortening is that we do not delete any coordinates.

The unit vector $e_i$ is the vector that is zero on all indices, except for index $i$, where it is equal to 1. Note that adding $e_i$ to the dual of $C$ has the effect of shortening $C$ on position $i$. With this observation we can give an expression for the dual of a shortened code $C_I$:

$$(C_I)^\perp = \langle C^\perp \cup \{e_i \mid i \in I\}\rangle. \tag{4.6}$$

The subcodes $C_I$ and $(C^\perp)_{\overline{I}}$ of $C$ and $C^\perp$ respectively are depicted in Figure 4.3.

Figure 4.1: Illustration of the subcodes $C_I$ and $(C^\perp)_{\overline{I}}$



## 4.4 The dimension of a shortened code

Suppose we are playing a variant on bingo. A bingo card can have any amount of numbers on it. Whenever someones card is full he gets a prize and the game continues. We happen to have a special collection of cards. On each card is written the support of a codeword in the dual of a code $C$. Just for the fun of it, we also shorten the code $C$ on the position whose number is called. What happens if we win a prize? Just before the last call, all but one of the numbers on our winning card were checked. This means that the shortened code is zero at these positions. It is also orthogonal to the codeword on the card (which is in the dual), so the shortened code must also be zero on the last number called. Hence, shortening on this position does not lower the dimension! Apart from the bingo prize we therefore also 'won' a dimension in the shortened code.

Intuitively it seems that winning a prize is equivalent to 'winning' a dimension when shortening a code. A mathematical proof is contained in the following theorem. It provides us with a relation between the dimensions of the shortened code $C_I$ and the subcode $(C^\perp)_{\overline{I}}$ of the dual. The number of dimensions you gain when shortening on $I$ is precisely the dimension of $(C^\perp)_{\overline{I}}$.

**Theorem 4.4.** *For all $I \subset \{1, \ldots, n\}$, the dimension of a $[n, k, d]$-code $C$ shortened on $I$ is given by*

$$\dim(C_I) = k - |I| + \dim((C^\perp)_{\overline{I}}). \tag{4.7}$$

*Proof.* The subspace $(C^\perp)_{\overline{I}}$ is spanned by some vectors $h_1, \ldots, h_a$, where $a = \dim((C^\perp)_{\overline{I}})$. Note that the supports of all these vectors are contained in $I$ by definition of shortening. Extend this basis to a basis $h_1, \ldots, h_{n-k}$ for $C^\perp$. Any proper combination of the vectors $h_{a+1}, \ldots, h_{n-k}$ is not disjoint from $\overline{I}$, since otherwise it would be an element of $(C^\perp)_{\overline{I}}$. The dual of the shortened code $C_I$ is spanned by $h_1, \ldots, h_{n-k}$ and the vectors $e_i$ for $i \in I$. The vectors $h_1, \ldots, h_a$ are all in the span of the $e_i$, but $h_{a+1}, \ldots, h_{n-k}$ are not. We conclude that $h_{a+1}, \ldots, h_{n-k}$ and $e_i$, where $i \in I$, form a basis for $(C_I)^\perp$. Therefore we have that $\dim((C_I)^\perp) = \dim(C^\perp) - \dim((C^\perp)_{\overline{I}}) + |I|$. Substituting the duality relations $\dim((C_I)^\perp) = n - \dim(C_I)$ and $\dim(C^\perp) = n - \dim(C)$ ends the proof. □

The above theorem can also be obtained by combining the following two duality lemmas formulated by Forney [11]. He does not only consider shortened codes $C_I$, but also projected codes $P_I(C)$ and he then relates their dimensions. A projected code $P_I(C)$ is the image of a projection $P_I$ which zeroes out components outside of $I$. Forney's first lemma follows from the fact that $C_I$ is the kernel of the projection $P_I$ on $C$. The second lemma is a consequence of the duality of $C_{\overline{I}}$ and $P_I(C)$.

**Lemma 4.5** (first duality lemma [11])**.** *If $C$ is a $[n, k]$ code, then*

$$\dim(C_I) + \dim(P_I(C)) = k. \tag{4.8}$$

**Lemma 4.6** (second duality lemma [11])**.** *If $C$ is a $[n, k]$ code, then*

$$\dim(C_{\overline{I}}) + \dim(P_I(C^\perp)) = |I|. \tag{4.9}$$

It is remarkable that Forney gives a lower bound on the dimension of a shortened code as a corollary to Lemma 4.5 while he also could have given a precise expression.

## 4.5 Optimal subcodes

The weight hierarchy of a code is completely determined by the weights $d_r(C)$ for which $d_{r+1}(C) > d_r+1$ (the remaining weights can be found using $d_{r+1}(C) = d_r(C)$). We will call subcodes for which these weights are attained optimal.

**Definition 4.7.** *A $k'$-dimensional subcode of $C$ and of weight $n'$ is optimal if there does not exist a*

- *$k'$ dimensional subcode of $C$ of weight $n' - 1$;*
- *$k' + 1$ dimensional subcode of $C$ of weight $n' + 1$.*

Next we will show that $C_I$ is optimal if and only if $(C^\perp)_{\overline{I}}$ is optimal. To this end we need the following lemma.

**Lemma 4.8.** *If $|\chi(C_I)| < n - |I|$ for some index set $I$, then there is a set $J \supset I$ such that $C_J = C_I$ and $|\chi(C_J)| = n - |J|$.*

*Proof.* If $|\chi(C_I)| < n - |I|$, then there are $t = n - |I| - |\chi(C_I)|$ different indices $a_1, \ldots, a_t \notin I$ where all codewords of $C_I$ are zero. Shortening the code $C$ on index set $J := I \cup \{a_1, \ldots, a_t\}$ gives the same code $C_I$, so $\dim(C_J) = \dim(C_I)$, and

$$\begin{aligned} n - |J| &= n - (|I| + t) \\ &= n - (n - \chi(C_I)) \\ &= \chi(C_I) \\ &= \chi(C_J) \end{aligned}$$

So index set $J$ has the desired properties. □

Barbero and Munuera [3] also proved the next theorem.

**Theorem 4.9.** *The code $C_I$ is optimal in $C$ if and only if $(C^\perp)_{\overline{I}}$ is optimal in $C^\perp$.*

*Proof.* It is sufficient to prove that $C_I$ is optimal if $(C^\perp)_{\overline{I}}$ is optimal. The converse then follows by switching roles: the code is the dual of its dual. Suppose that $(C^\perp)_{\overline{I}}$ is optimal and let $t := \dim(C^\perp)_{\overline{I}}$. By Theorem 4.4 we know that $s := \dim(C_I) = k - |I| + t$. Note that because of the optimality of $(C^\perp)_{\overline{I}}$, we also know that $\chi((C^\perp)_{\overline{I}}) = I$. First we will show that there does not exist a

subcode of the same dimension as $C_I$ and weight smaller than $n - |I|$. This gives us the weight of $C_I$ and also proves part of the optimality. We conclude the proof by showing that there is not a subcode of $C$ of dimension $s+1$ and weight $n - |I| + 1$.

Suppose there exists a $s$-dimensional subcode of $C$ whose weight is smaller than $n - |I|$. This subcode is contained in a shortened code $C_J$ of dimension $s' \geq s$. As a consequence of Lemma 4.8 we may assume without loss of generality that $|\mathrm{supp}(C_J)| = n - |J| < n - |I|$. Theorem 4.4 gives us an expression for the dimensions of the shortened subcodes $C_I$ and $C_J$. Using this expression we get

$$\begin{aligned} \dim((C^\perp)_{\overline{J}}) - \dim((C^\perp)_{\overline{I}}) &= \dim(C_J) - k + |J| - (\dim(C_I) - k + |I|) \\ &= \dim(C_J) - \dim(C_I) + |J| - |I| \\ &\geq |J| - |I|. \end{aligned}$$

The weight of $(C^\perp)_{\overline{J}}$ is at most $|J|-|I|$ larger than that of $(C^\perp)_{\overline{I}}$. Its dimension is at least $|J| - |I|$ larger. Each shortening operation on $(C^\perp)_{\overline{J}}$ decreases the weight by at least one. Doing so until there are $\dim((C^\perp)_{\overline{I}})+1$ dimensions left, gives a subcode of weight at most $|I| + 1$. This contradicts the optimality of $(C^\perp)_{\overline{I}}$. So there does not exist a subcode of dimension $s$ and weight smaller than $n - |I|$. The weight of the shortened code $C_I$ is therefore exactly $n - |I|$.

The only thing left to proof is that there is not a subcode of $C$ of weight $n - |I| + 1$ and dimension $s + 1$. This follows easily from Theorem 4.2, since it gives us $d_t(C^\perp) = |I| \neq n + 1 - d_{s+1}(C)$. And thus we have proved that $C_I$ is optimal as well. □

The duality Theorem 4.2 by Wei relates the generalized Hamming weights of a code and its dual, but does not say anything about subcodes for which these weights are attained. Our theorem does so, however only for optimal subcodes. It could be used to search for optimal subcodes in the code and its dual simultaneously.

## 4.6 Extended codes

Initially, we found a generalized Hamming weight of an extended BCH code, but in the literature people were only interested in the non-extended code. Therefore we wanted to find a way to translate our result to one on the non-extended code. It was quite easy to do so and we observed that the only necessary condition was that the automorphism group of the extended code had to be transitive on the coordinates. The next theorem expresses this.

**Theorem 4.10.** *Let $C$ be a code and suppose that the automorphism group of the extended code $C^{ext}$ is transitive. Then we have*

$$d_r(C^{ext}) = d_r(C) + 1 \tag{4.10}$$

*for all $r$.*

*Proof.* First we will show that $d_r(C^{\text{ext}}) \geq d_r(C)+1$ and then $d_r(C^{\text{ext}}) \leq d_r+1$. Label the coordinates of $C$ by the set $\{1,\ldots,n\}$ and the coordinates of $C^{\text{ext}}$ by $\{0,1,\ldots,n\}$ (the symbol on position 0 is thus the parity check symbol). Suppose that $D$ is a $r$-dimensional subcode of $C^{\text{ext}}$ of minimal weight $d_r(C^{\text{ext}})$. Using the transitivity of the automorphism group we may assume without loss of generality that at least one of the codewords in $D$ is nonzero at coordinate zero. Puncturing the subcode $D$ on this coordinate is the same as projecting $D$ on $I =: \{1,\ldots,n\}$. By Lemma 4.5 we have $\dim(D) = \dim(P_I(D)) + \dim(D_I)$. If $\dim(D_I) = 1$, then there is a vector of weight 1 in $C^{\text{ext}}$, which is obviously a contradiction. So by puncturing the subcode $D$ on position 0 we get a subcode $D'$ of $C$ whose weight is precisely one smaller and whose dimension is the same. Hence we have that $d_r(C) + 1 \leq d_r(C^{\text{ext}})$.

Suppose that $D$ is a $r$-dimensional subcode of $C$ of minimal weight $d_r(C)$. Extending this subcode $D$ gives a subcode $D'$ of $C^{\text{ext}}$ whose weight is the same as the weight of $D'$ if all codewords in $D$ are of even weight and it is one more if one of the codewords in $D$ is of odd weight. In both cases we get the desired $d_r(C^{\text{ext}}) \leq d_r(C) + 1$.

Combining the two inequalities proves the theorem. □

Without the transitivity of the automorphism group of the extended code we are only able to claim that $d_r(C)$ and $d_r(C^{\text{ext}})$ differ by at most one, which Wei and Yang [31] already proved.

## 4.7 Some applications of generalized Hamming weights

### 4.7.1 Wiretap channel of type II

The wiretap channel of type II is a noiseless channel over which a sender wants to transmit $k$ bits of information without using encryption. The adversary is able to intercept any $s$ bits of his choosing. The goal is of course to minimize the amount of information that the adversary is able to gain. Simply transmitting the $k$ bits gives the attacker the most information he is able to get ($s$ bits), so this is not a good idea to do. Ozarow and Wyner [26] suggested the following scheme using a linear $[n, n-k]$ code $C$.

Regard the $k$ bits of information as a syndrome and randomly select a word $x$ (of length $n$) from the corresponding coset. This word $x$ is then transmitted over the channel. The question is how much information the adversary is able to extract from the $s$ intercepted bits. This happens to be the maximum possible dimension of a subcode of $C^{\perp}$ of length $s$.

### 4.7.2 Trellis complexity

A trellis is, roughly speaking, a directed labeled graph whose vertex set $V$ can be partitioned into subsets $V = V_0 \cup V_1 \cup \ldots \cup V_n$ with the following properties. The

subset $V_0$ contains only one vertex: the source. Edges are labelled with either 0 or 1 and are only present between successive subsets $V_i$ and $V_{i+1}$. Concatenating the labels of the edges on a path of length $n$ from the source to a vertex in $V_n$ gives a word of the same length. For well-constructed trellises, all these possible words are codewords of a linear code $C$. Such a trellis can then be used for decoding by assigning a cost to each edge: 0 if the label of the edge is the same as the corresponding bit in the received word and 1 otherwise. The problem of finding the codeword closest to the received word is then reduced to finding a shortest path in the trellis. A more elaborate and precise treatment on trellises can be found in [25] and [24].

There are many ways to construct a trellis for a code. Important properties of a trellis are, for example, the total number of vertices, the total number of edges and the state complexity $\max_i(|V_i|)$ (as defined by Forney [11]). The BCJR/minimal trellis (see [2] and [10]) minimizes these properties simultaneously. Permuting the coordinates gives essentially the same code, but the parameters of the minimal trellis change. It is therefore worthwile to find the permutation for which the minimal trellis has the best characteristics. Generalized Hamming weights can be used to find a good permutation that almost minimizes the state complexity. It should be noted that the permutation for which a certain parameter is minimal, is not necessarily the same as the permutation that minimizes another parameter.

### 4.7.3 Curves with many points

Some codes can be constructed from a vector space of polynomials. The codewords are the binary vectors obtained by evaluating the trace of such polynomial in a set of points in $\mathbb{F}_{2^m}$. Examples of such codes are the dual Melas codes, dual BCH(2,m) and the dual BCH(3,m) codes. A codeword $c$ derived from a polynomial $f(x)$ can be associated with an algebraic curve $\mathcal{K}_c$ over $\mathbb{F}_{2^m}$ with defining equation

$$\mathcal{K}_c : y^2 + y = f(x). \tag{4.11}$$

The number of points on this curve can be counted by observing that $\mathrm{Tr}(a) = 0$ if and only if there exists an element $b \in \mathbb{F}_{2^m}$ such that $b^2 + b = a$ (see for example [21]). To have many points on the curve we therefore need to have many values of $x$ for which $\mathrm{Tr}(f(x)) = 0$. This is equivalent to searching for a codeword of low weight. Subcodes correspond with so called fibre products of curves. In this case it is also true that the lower the weight of the subcode, the more points on the curve.

See [13] for a detailed explanation of how to construct such curves.

### 4.7.4 Shortening a code

By Theorem 4.4 we know that we can affect the dimension of a shortened code in two different ways. First we can choose on how many positions we shorten the code. The more positions we pick, the lower the dimension of the shortened

code. Once we have fixed the size of $|I|$ we can try to maximize the dimension of $((C^{\perp})_{\overline{I}})$ in order to keep the dimension of $C_I$ as high as possible. The latter is of course closely related to generalized Hamming weights since these weights give an upper bound on the maximum dimension possible.

# Chapter 5

# BCH codes

## 5.1 Introduction

In his paper on generalized Hamming weights, Wei [32] determined the weight hierarchy of several classes of well-known codes. The primitive narrow-sense double-error-correcting BCH code was not one of those. Among the first to publish on this subject, Chung [7] showed that $d_2(BCH(2,m)^\perp)$ achieved the Griesmer bound whenever $m$ is not a multiple of four. Together with Shim [27] he later managed to prove that $m = 4$ is the only case for which this bound is not achieved. Feng, Tzeng and Wei [9] showed that $d_2(BCH(2,m)) = 8$ for all $m \geq 4$ and with the help of some computer search in the dual code they produced the complete weight hierarchy for BCH(2,5) and BCH(2,6). They also gave a lower bound for $d_r(BCH(2,m))$ which is very slightly better than the bounds from the table of best known linear codes by Brouwer and Verhoeff [6]. The last results on this subject are by van der Geer and van der Vlugt [12] who proved that $d_3(BCH(2,m)) = 10$. Van der Vlugt [30] then gives upperbounds for the first six generalized Hamming weights of $BCH(2,m)$; he also describes the complete weight hierarchy of $BCH(2,4)$. As a direct consequence it followed that $d_4(BCH(2,m)) = 11$ for $m \equiv 0 \bmod 4$.

In the next section we will first give an alternative description of codewords in both $BCH(2,m)$ and $EBCH(2,m)$. Subsequently we describe their automorphism group. These two ingredients are needed to explain how we can find a specific code as a subcode of $BCH(2,m)$. Then we identify codewords of minimum weight with lines in a geometry and derive some properties. This approach is essentially the same as taken by van der Geer en van der Vlugt [12], but the formulation is nicer. In this setting we provide another proof of $d_3(BCH(2,m)) = 10$ for odd $m$. Using a completely different technique we derive a new result for even $m$, namely $d_4(BCH(2,m)) = 11$. Based on computer search we believe that $d_4(BCH(2,m)) = 12$ for odd $m$, but we have not been able to prove this. The first three generalized Hamming weights are the same for even and odd $m$, but they differ in the fourth weight. We will discuss

possible reasons for this distinction and mention some approaches we took to tackle the case of odd $m$.

## 5.2 An alternative description of BCH(2,m) codes

### 5.2.1 The non-extended code

A codeword $c = (c_0, c_1, \ldots, c_{n-1})$ is the representation of the polynomial $c(x) = \sum_{i=0}^{n-1} c_i X^i \in \mathbb{F}_{2^m}[X]/(X^n - 1)$. This polynomial has by definition of the BCH(2,m) code $\alpha$ and $\alpha^3$ as zeros, where $\alpha$ is a (fixed) primitive element in $\mathbb{F}_{2^m}$. Every codeword therefore satisfies the equations

$$\sum_{i=0}^{n-1} c_i \alpha^i = 0 \tag{5.1}$$

and

$$\sum_{i=0}^{n-1} c_i \alpha^{3i} = 0. \tag{5.2}$$

Another way to represent a codeword $c$ is by a set $X_c \subseteq \mathrm{F}_{2^m}$ consisting of the terms of $c(\alpha)$. A formal definition of this representation is

$$X_c := \{\alpha^i \mid i \in \chi(c)\}. \tag{5.3}$$

With this representation in mind we give another definition of a codeword of $BCH(2, m)$, in terms of subsets of $\mathbb{F}_{2^m}^*$.

**Definition 5.1.** *A subset $X \subseteq \mathbb{F}_{2^m}^*$ is a codeword of $BCH(2, m)$ if and only if*

$$\sum_{x \in X} x = 0 \tag{5.4}$$

*and*

$$\sum_{x \in X} x^3 = 0. \tag{5.5}$$

From now on we will only use $X_c$ if we want to emphasize that the codeword $c \in \mathbb{F}_{2^m}^n$ is associated with this set $X_c$. Otherwise we will simply use $X$ to denote a codeword.

### 5.2.2 The extended code

A codeword $\tilde{c}$ of $EBCH(2, m)$ can be written as $\tilde{c} = (c_0, c_1, \ldots, c_{n-1}, c_n)$ where $(c_0, c_1, \ldots, c_{n-1}) = c \in BCH(2, m)$ and $c_n = \sum_{i=0}^{n-1} c_i$. Since we are dealing with binary codewords, $c_n = 1$ if and only if the weight of $c$ is odd. The only role of $c_n$ is to make the weight of the extended codeword even. The zero element

of $\mathbb{F}_{2^m}$ can do the same in a representation of an extended codeword considered as a subset $X_{\tilde{c}} \subseteq \mathbb{F}_{2^m}$ as follows.

$$X_{\tilde{c}} = \begin{cases} X_c & \text{if } c_n = 0 \\ X_c \cup \{0\} & \text{if } c_n = 1 \end{cases} \tag{5.6}$$

This leads us to another definition of an extended codeword.

**Definition 5.2.** *A subset $X \subseteq \mathbb{F}_{2^m}$ is a codeword of $EBCH(2, m)$ if and only if $|X|$ is even,*

$$\sum_{x \in X} x = 0 \tag{5.7}$$

*and*

$$\sum_{x \in X} x^3 = 0. \tag{5.8}$$

The differences between this definition and that of non-extended codeword are that we allow a subset to contain the zero element and that the size of the set has to be even. The latter property will turn out to be very useful in proving a theorem on the automorphisms of the EBCH(2,m) code.

### 5.2.3 Automorphisms

One of the interesting properties of a code is its automorphism group. This section is devoted exclusively to this property.

**Theorem 5.3.** *The mapping $\phi := x \mapsto ax^{2^i} + b$ where $i \in \{0, 1, \ldots, m-1\}$, $a \in \mathbb{F}_{2^m}^*$ and $b \in \mathbb{F}_{2^m}$ is an automorphism of the extended $BCH(2, m)$ code.*

*Proof.* A codeword $X$ is mapped to $\phi(X) := \{\phi(x) \mid x \in X\}$. It is easy to see that $\phi$ is a bijection, so distinct codewords have distinct images and $|\phi(X)| = |X|$. The mapping is an automorphism if $\phi(X)$ is a codeword as well, so it has to satisfy the requirements of Definition 5.2. The first one is obviously met. Using the property that $|X|$ is even, we have

$$\begin{aligned} \sum_{y \in \phi(X)} y &= \sum_{x \in X} (ax^{2^i} + b) \\ &= a \sum_{x \in X} x^{2^i} + b \sum_{x \in X} 1 \\ &= a \left( \sum_{x \in X} x \right)^{2^i} + 0 \\ &= 0, \end{aligned}$$

hence requirement (5.7) is also satisfied. Finally, again using that the cardinality of $X$ is even, we have

$$\sum_{y \in \phi(X)} y^3 = \sum_{x \in X} (ax^{2^i} + b)^3$$

$$
\begin{aligned}
&= \sum_{x\in X}(a^3x^{3.2^i} + a^2x^{2.2^i}b + ax^{2^i}b^2 + b^3)\\
&= a^3\left(\sum_{x\in X}x^3\right)^{2^i} + a^2b\left(\sum_{x\in X}x\right)^{2^{i+1}} + ab^2\left(\sum_{x\in X}x\right)^{2^i} + b^3\sum_{x\in X}1\\
&= 0.
\end{aligned}
$$

All requirements are met, so $\phi(X)$ is a codeword as well. And hence we have proven that $\phi$ is an automorphism of the extended BCH(2,m) code. □

Define the mapping $\phi(i,a,b) := x \mapsto ax^{2^i} + b$ and consider the set $G$ of all such mappings, that is

$$G = \{\phi(i,a,b) \mid 0 \le i < m,\ a \in \mathbb{F}_{2^m}^* \text{ and } b \in \mathbb{F}_{2^m}\} \tag{5.9}$$

then we have the following theorem.

**Theorem 5.4.** *$G$ is a group of automorphisms of $EBCH(2,m)$ where multiplication is the composition of the mappings.*

*Proof.* First we will show that the set $G$ is indeed closed under multiplication. Then we prove that the mapping $\phi(0,1,0) = x \mapsto x$ is the unit element. We get the associativity for free since multiplication is the same as composing two mappings.

Let $\phi(i,a,b)$ and $\phi(j,c,d)$ be elements of $G$, then

$$
\begin{aligned}
\phi(i,a,b)\circ\phi(j,c,d) &= (x \mapsto ax^{2^i} + b)\circ(x \mapsto cx^{2^j} + d)\\
&= x \mapsto a(cx^{2^j} + d)^{2^i} + b\\
&= x \mapsto ac^{2^i}x^{2^{j+i}} + ad^{2^i} + b\\
&= \phi(i+j, ac^{2^i}, ad^{2^i} + b),
\end{aligned}
$$

which proves that $G$ is closed under multiplication. Now we will verify that the inverse of $\phi(i,a,b)$ is $\phi(m-i, a^{-2^{m-i}}, (a^{-1}b)^{2^{m-i}})$. It is indeed a right inverse since

$$
\begin{aligned}
\phi(i,a,b)\circ\phi(m-i, a^{-2^{m-i}}, (a^{-1}b)^{2^{m-i}}) &=\\
\phi(i+m-i, a(a^{-2^{m-i}})^{2^i}, a((a^{-1}b)^{2^{m-i}})^{2^i} + b) &=\\
\phi(0, a.a^{-2^m}, a.a^{-2^m}b^{2^m} + b) &=\\
\phi(0, a.a^{-1}, b + b) &=\\
\phi(0,1,0).
\end{aligned}
$$

In a similar way, we show that it is a left inverse as well:

$$
\begin{aligned}
\phi(m-i, a^{-2^{m-i}}, (a^{-1}b)^{2^{m-i}})\circ\phi(i,a,b) &=\\
\phi(m-i+i, a^{-2^{m-i}}.a^{2^{m-i}}, a^{-2^{m-i}}b^{2^{m-i}} + (a^{-1}b)^{2^{m-i}}) &=\\
\phi(0,1,0).
\end{aligned}
$$

This proves that $G$ is a group of automorphisms of EBCH(2,m). It is in fact the full automorphism group if $m > 4$ (see [4]). □

The full automorphism group of BCH(2,m) consists of the mappings $\phi : x \mapsto ax^{2^i}$ (see [4]).

**Theorem 5.5.** *The group $G$ is 2-transitive on $\mathbb{F}_{2^m}$.*

*Proof.* Take two arbitrary pairs of distinct points, say $(s,t)$ and $(u,v)$. With Lagrange interpolation we can find a first degree polynomial $f : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ such that $f(s) = u$ and $f(t) = v$. Since $G$ contains all first degree polynomials (just take $i = 0$) it follows that $f \in G$. And thus we conclude that $G$ is 2-transitive. □

Now that we have established transitivity, we get the following corollary from Theorem 4.10.

**Corollary 5.6.** *For every dimension $r$ we have*

$$d_r(EBCH(2,m)) = d_r(BCH(2,m)) + 1. \tag{5.10}$$

## 5.3 A note on codewords

The following lemma tells us how to find the remaining elements of a codeword if it is only partially known.

**Lemma 5.7.** *A codeword $X = \{x_1, \ldots, x_t\}$ of BCH(2,m) or EBCH(2,m) is completely determined by all but two of its elements. Let $a := \sum_{i=1}^{t-2} x_i$ and $b := \sum_{i=1}^{t-2} x_i^3$. The last two elements, say $x_{t-1}$ and $x_t$, are zeros of the second-degree polynomial*

$$f(z) := az^2 + a^2 z + a^3 + b \tag{5.11}$$

*and are given by*

$$\{x_{t-1}, x_t\} = aRT_{2^m}\left(1 + b/a^3\right). \tag{5.12}$$

*Proof.* The set $X$ is a codeword, so its elements satisfy $\sum_{i=1}^{t} x_i = \sum_{i=1}^{t} x_i^3 = 0$. Using the definitions of $a$ and $b$ the equations simplify to

$$\begin{aligned} x_{t-1} + x_t + a &= 0 \\ x_{t-1}^3 + x_t^3 + b &= 0. \end{aligned}$$

Since $x_t \neq x_{t-1}$ we have $a \neq 0$. Substitute the linear equation into the cubic equation to get the following equation in $x_t$:

$$\begin{aligned} 0 &= x_t^3 + (x_t + a)^3 + b \\ &= x_t^3 + x_t^3 + ax_t^2 + a^2 x_t + a^3 + b \\ &= f(x_t). \end{aligned}$$

The zeros of the polynomial $f(z) = az^2 + a^2z + a^3 + b = a^3\left((\frac{z}{a})^2 + \frac{z}{a} + \frac{a^3+b}{a^3}\right)$ can be found by solving the quadratic equation $y^2 + y + 1 + \frac{b}{a^3} = 0$, where $y = z/a$. By Definition 2.13, the solutions to this equation are $\mathrm{RT}_{2^m}(1 + b/a^3)$. Thus we get $\{x_{t-1}, x_t\} = a\mathrm{RT}_{2^m}(1 + b/a^3)$, which concludes the proof. □

## 5.4 How to find a specific code as a subcode of BCH(2,m)

Every subcode of BCH(2,m) is a code in itself, but how can we determine whether a code is a subcode of BCH(2,m)?

Let $C$ be a $[n, k, d]$ code spanned by $c_1, \ldots, c_k$. Each of the $c_i$ has to be a codeword of $BCH(2, m)$, so it has to correspond with a subset $X_{c_i} \subseteq \mathbb{F}_{2^m}^*$ satisfying Definition 5.2. A position $t$ in the support of $c_i$ corresponds with an element of $X_{c_i}$, say $x_t$. If position $t$ is also in the support of another codeword $c_j$, then $x_t$ is also an element of $X_{c_j}$. Thus we need $n$ variables $x_j$, where $1 \leq j \leq n$ to describe the subsets $X_{c_i}$, where $1 \leq i \leq k$ as follows

$$X_{c_i} = \{x_j \mid j \in \chi(c_i)\}. \tag{5.13}$$

Each subset $X_{c_i}$ has to satisfy the equations of Definition 5.1, that is

$$\sum_{j \in \chi(c_i)} x_j = 0 \tag{5.14}$$

and

$$\sum_{j \in \chi(c_i)} x_j^3 = 0. \tag{5.15}$$

The question is whether this system of $k$ linear and $k$ cubic equations has a solution for which all $x_j$ are distinct. If it does exist, then the code $C$ is a subcode of BCH(2,m).

To find a specific code as a subcode of the extended BCH(2,m) we only have to add the requirement that $|X_{c_i}|$ is even for all $i$.

## 5.5 A geometric approach

In this section we will prove some results on generalized Hamming weights of EBCH(2,m) codes by looking at the problem from a geometric point of view. We take the elements of $\mathbb{F}_{2^m}$ as the points of our geometry. The lines correspond with the codewords of minimal weight in the extended BCH(2,m) code. An obvious property of this geometry is that two lines intersect in at most three points, since otherwise we would get a codeword of weight four. But there are more properties to discover. First we will compute the number $\lambda$ of lines through any two points and subsequently the number $\mu$ of lines through any three points. Note that Kasami [20] already determined these numbers. With that knowledge we will determine the third generalized Hamming weight.

**Theorem 5.8** ([20]). *The number $\lambda$ of lines through any two points $x_1$ and $x_2$ is*

$$\lambda = \begin{cases} (2^m-2)(2^m-8)/24 & \text{if } m \text{ odd} \\ (2^m-4)^2/24 & \text{if } m \text{ even.} \end{cases} \tag{5.16}$$

*Proof.* Because of the 2-transitivity of EBCH(2,m) it suffices to count the number of lines through 0 and 1. Using Lemma 5.7, we will count the number of 2-sets $\{x, y\}$ that, joined with $\{0, 1\}$, yield a codeword $X$. However, any 2-set of $X \setminus \{0, 1\}$ gives the same result. So this number of 2-sets is $\binom{4}{2} = 6$ times the desired answer.

Let $\{x, y\} \subset \mathbb{F}_{2^m}$ and define $a := x+y$ and $b := x^3+y^3$. If $\{0, 1, x, y\}$ is part of a codeword $X$ (of weight six), then by Lemma 5.7 the remaining elements, say $\{u, v\}$, are given by $\{u, v\} = (a + 1)\mathrm{RT}(1 + (b + 1)/(a + 1)^3)$. This set of roots is non empty if and only if $\mathrm{Tr}(1 + (b + 1)/(a + 1)^3) = 0$, which gives us our first requirement on $a$ and $b$ and thus on $x$ and $y$. It must also hold that $u + v = 0 + 1 + x + y = a + 1$ and $u^3 + v^3 = 0^3 + a^3 + x^3 + y^3 = b + 1$. The set $\{0, 1, u, v\}$ is also part of codeword $X$, so again by Lemma 5.7 we get $\{x, y\} = a\mathrm{RT}(1 + b/a^3)$. This gives us our second requirement that $\mathrm{Tr}(1 + b/a^3) = 0$.

For convenience, define $s := 1 + \frac{b}{a^3}$ and $t := 1 + \frac{b+1}{(a+1)^3}$. We can write the expressions for $s$ and $t$ as $b = (s+1)a^3$ and $b+1 = (t+1)(a+1)^3$. Using these equations, we get

$$\begin{aligned} a^3 s &= b + a^3 \\ &= a^3 + 1 + (t+1)(a+1)^3 \\ &= (a+1)^3 + a^2 + a + (t+1)(a+1)^3. \end{aligned}$$

So $s = \frac{1}{a} + \frac{1}{a^2} + \left(\frac{a+1}{a}\right)^3 t$, and also $t = \frac{1}{a+1} + \frac{1}{(a+1)^2} + \left(\frac{a}{a+1}\right)^3 s$. If $s = 0$ or $t = 0$ we get $0 \in \mathrm{RT}(s)$ or $0 \in \mathrm{RT}(t)$ and thus $0 \in \{x, y, u, v\}$. Since this would not yield a codeword, we must have $s \notin \{0, 1/a+1/a^2\}$. If $a = 0$, then we get $x = y$ and if $a = 1$, then $u = v$. Therefore we also have the constraint $a \notin \{0, 1\}$. We have to count the number of pairs $(a, s)$ such that

$$\mathrm{Tr}(s) = \mathrm{Tr}\left((\frac{a}{a+1})^3 s\right) = 0, \tag{5.17}$$

$$a \notin \{0, 1\}, \tag{5.18}$$

$$s \notin \{0, 1/a + 1/a^2\}. \tag{5.19}$$

We can apply Lemma 2.16 to compute this number, but before we can use this lemma we need to find out when $a^3/(a + 1)^3 = 1$. We have $a^3 = (a + 1)^3 = a^3 + a^2 + a + 1$ if and only if $a^2 + a + 1 = (a^3 + 1)/(a + 1) = 0$. The latter holds if and only if $a^3 = 1$.

First consider the case that $m$ is odd. Then there are no third roots of unity in $\mathbb{F}_{2^m}$. We can choose $a$ in $2^m - 2$ different ways and $s$ in $2^m/4 - 2$ different ways. This gives us $(2^m - 2)(2^m/4 - 2) = (2^m - 2)(2^m - 8)/4$ possible

sets $\{x, y\}$ that are part of a codeword containing $\{0, 1\}$ as well. As we have argued before this number is six times the number of lines through 0 and 1, so $\lambda = ((2^m - 2)(2^m - 8)/24$, if $m$ is odd.

Now assume that $m$ is even. We can choose $a$ in two different ways, such that $a^3 = 1$. $s$ can then be choosen in $2^m/2 - 2$ different ways. If $a$ is not a third root of unity, then there are $2^m/4 - 2$ choices for $s$. This gives $2(2^m/2 - 2) + (2^m - 4)(2^m/4 - 2) = (2^m - 4)(2^m/4 - 1) = (2^m - 4)^2/4$ possible choices for $\{x, y\}$ in total. The number of codewords is hence $(2^m - 4)^2/24$. □

**Theorem 5.9** ([20])**.** *For odd $m$, the number $\mu$ of lines through any three points is equal to*

$$\mu = \frac{2^m - 8}{6}. \tag{5.20}$$

*Proof.* By definition of a line, this number $\mu$ is the same as the number of codewords whose support contains the three selected points. Because of the 2-transitivity of the automorphism group we may assume without loss of generality that each of these codewords is a set $\{0, 1, \gamma, x, y, z\}$ of distinct elements and where $\gamma$ is fixed. We will count the number of $x$ for which there indeed does exist such codeword. As we have seen in Lemma 5.7, the elements $x$ and $y$ are given by $\{x, y\} = \mathrm{RT}(1 + (1 + \gamma^3 + x^3)/(1 + \gamma + x)^3)$. This set of roots is not empty if and only if $\mathrm{Tr}(1 + (1 + \gamma^3 + x^3)/(1 + \gamma + x)^3) = 0$. If $x \in H := \{0, 1, \gamma, \gamma + 1\}$ then some of the elements are equal, so we must exclude these values of $x$. Note that the set $H$ is closed under addition. We can compute the total number of $x$ satisfying these conditions with the help of a sum of additive characters.

Before we do so, we need to introduce the map $\phi : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ defined by $\phi(x) = 1/x^3$ for all $x \in \mathbb{F}_{2^m}^*$ and $\phi(0) = 0$. Since $m$ is odd, $\phi$ is a bijection. Let $N$ denote the number of $x$ for which $\mathrm{Tr}(x) = 0$, and let $M$ be the number of $x$ for which $\mathrm{Tr}(x) = 1$. Then, using this bijection, we have

$$\begin{aligned}
N - M &= \sum_{x \notin H} \psi\left(1 + \frac{1 + \gamma^3 + x^3}{(1 + \gamma + x)^3}\right) \\
&= (-1)^m \sum_{x \notin H} \psi\left(\frac{1 + \gamma^3 + x^3}{(1 + \gamma + x)^3}\right) \\
&= (-1)^m \sum_{y \notin H + 1 + \gamma} \psi\left(\frac{1 + \gamma^3 + (y + 1 + \gamma)^3}{y^3}\right) \\
&= (-1)^m \sum_{y \notin H} \psi\left(\frac{1 + \gamma^3 + (1 + \gamma)^3}{y^3} + \frac{1 + \gamma}{y} + \frac{(1 + \gamma)^2}{y^2} + 1\right) \\
&= (-1)^{2m} \sum_{y \notin H} \psi\left(\frac{\gamma + \gamma^2}{y^3}\right) \\
&= \sum_{u \notin \phi(H)} \psi((\gamma + \gamma^2)u)
\end{aligned}$$

$$
\begin{aligned}
&= 0 - \psi(0) - \psi(\gamma+\gamma^2) - \psi(\frac{\gamma+\gamma^2}{\gamma^3}) - \psi(\frac{\gamma+\gamma^2}{(\gamma+1)^3}) \\
&= -4.
\end{aligned}
$$

Because we excluded values in $H$, we also know that $N+M = 2^m-4$. So there are $N = (2^m-8)/2$ solutions. We did count every codeword thrice, hence the number of codewords with three fixed points is equal to $(2^m-8)/6$. □

**Corollary 5.10.** *The collection of supports of codewords of minimal weight 6 in EBCH(2,m), m odd, constitutes a* $3-(2^m-1,6,\mu)$ *design.*

It is well known (see for example [23], page 451) that for odd $m$ the dual of $BCH(2,m)$ has only codewords of weight $2^{m-1}$ or $2^{m-1}\pm 2^{(m-1)/2}$. The dual of EBCH(2,m) is spanned by the 1-vector and a basis for $BCH(2,m)^\perp$. Therefore we know that $EBCH(2,m)^\perp$ has only four weights. The corollary then also follows as an application of the Assmus-Mattson theorem [1].

Now that we have determined $\lambda$ and $\mu$, we can prove that the third generalized Hamming weight of the extended BCH(2,m) code is 11. As an immediate consequence we then have $d_3(BCH(2,m)) = 10$. This has already been proved by van der Geer and van der Vlugt [12] and our proof is essentially the same as theirs. But the parameters in the extended BCH(2,m) code are a bit nicer and representing codewords by lines makes the proof more readable.

We define the degree of a point with respect to a collection of lines as the number of lines through this point. The degree of a set of points is simply the sum of the degrees of its points.

The following theorem is on the third generalized Hamming weight in case $m$ is odd. The even case will be handled later on.

**Theorem 5.11.** *If m is odd, we have* $d_3(EBCH(2,m)) = 11$ *and as a consequence* $d_3(BCH(2,m)) = 10$.

*Proof.* Take any three distinct points, say $s$, $t$, and $u$. Let $\mathcal{L}_u$ be the collection of lines through these points. All these lines are disjoint apart from $\{s,t,u\}$. According to Theorem 5.9, the size of $\mathcal{L}_u$ is equal to $(2^m-8)/6$. Let $A$ be the set of points different from $\{s,t,u\}$ and on one of the lines in $\mathcal{L}_u$. Then $|A| = 3(q-8)/6 = (q-8)/2$, where we write $q = 2^m$ for convenience. Let $B$ be the set of remaining points. So $|B| = q-3-|A| = (2q-6-(q-8))/2 = (q+2)/2$.

The number of lines through $s$ and $t$, but not through $u$ is $\lambda-\mu = (q-2)(q-8)/24-(q-8)/6 = (q-6)(q-8)/24$. We will now show that at least one of these lines intersects two lines of $\mathcal{L}_u$ which will lead us to the desired result. Suppose it is not true. Then every line intersects $\mathcal{L}_u$ in at most one point and hence contributes at least three to the total degree of $B$. So $\deg(B) \geq 3(q-6)(q-8)/24 = \frac{3(q-6)}{4}\frac{q-8}{6}$. We can also directly compute the degree of $B$. The number of lines through $s$, $t$ and any point in $B$ is $(q-8)/6$, so this is also the degree of each point in $B$. The total degree of $B$ is therefore $|B|(q-8)/6 = \frac{q+2}{2}\frac{q-8}{6}$. It can easily be verified that $\frac{q+2}{2} < \frac{3(q-6)}{4}$ whenever $m \geq 5$, therefore these

Table 5.1: Cyclotomic cosets

| | cyclotomic coset |
|---|---|
| 1 | 1 |
| $\alpha$ | $\alpha, \alpha^2, \alpha^4, \alpha^8$ |
| $\alpha^3$ | $\alpha^3, \alpha^6, \alpha^{12}, \alpha^9$ |
| $\alpha^5$ | $\alpha^5, \alpha^{10}$ |
| $\alpha^7$ | $\alpha^7, \alpha^{14}, \alpha^{13}, \alpha^{11}$ |

two relations contradict each other. Hence our assumption was wrong and we conclude that there is a line that intersects two lines in $\mathcal{L}_u$. Since each of the three lines has a point that is not on of the other lines, the dimension of the subcode spanned by their characteristic vectors is three. The size of the support of this subcode is $6+3+2 = 11$. The Griesmer bound 2.6 states that the length has to be at least $6 + 3 + 2 = 11$. This proves that $d_3(EBCH(2, m)) = 11$. By Theorem 5.6 we then also have $d_3(BCH(2, m)) = 10$. □

## 5.6 On the fourth generalized Hamming weight of BCH(2)

**Theorem 5.12.** *The extended BCH(2,4) of length 16 has the* $[12, 4, 6]$ *code as a subcode. Moreover, all its codewords are contained in* $\mathbb{F}_{16} \setminus \mathbb{F}_4$.

*Proof.* The even weight BCH(2,4) code of length 15 has zeros 1, $\alpha$ and $\alpha^3$, where $\alpha$ is a primitive element of $\mathbb{F}_{16}$. The zeros of the parity check polynomial are hence $\alpha^5$ and $\alpha^7$ (see table 5.1). The generator polynomial of the dual is the reciprocal of the parity check polynomial. The zeros of the dual code are therefore $\alpha^{-5}$ and $\alpha^{-7}$, or equivalently, $\alpha$ and $\alpha^5$. Consider the polynomial $t(x) := 1 + x^5 + x^{10} = \frac{x^{15}+1}{x^5+1}$. It is easy to see that $t(\alpha) = t(\alpha^5) = 0$, so $t(x)$ is a codeword (viewn as a polynomial) of the dual. If we shorten the even weight BCH(2) on position $i$, all remaining codewords are zero at this position. Considered as subsets of $\mathbb{F}_{2^m}^*$ they therefore do not contain $\alpha^i$. Now we indeed do shorten the code on positions $\{0, 5, 10\} =: I$. The resulting subcode is contained in $\mathbb{F}_{16} \setminus \{0, 1, \alpha^5, \alpha^{10}\} = \mathbb{F}_{16} \setminus \mathbb{F}_4$. Theorem 4.4 on the dimension of a shortened code then says that the dimension of our subcode is at least $6 - 3 + 1 = 4$. The length is at most $15 - 3 = 12$. Because the $[12, 4, 6]$ code is optimal, the obtained subcode must be the $[12, 4, 6]$ code. □

The field $\mathbb{F}_{16}$ is a subfield of $\mathbb{F}_{2^m}$ if $m$ is a multiple of four. Hence all codewords of $[12, 4, 6]$ are also in such an extension field. Since there does not exist a $[11, 4, 6]$ code by the Griesmer bound we automatically obtain $d_4(EBCH(2, m)) = 12$ if $m$ is a multiple of four. But we can prove something even stronger! We are able to construct solutions for all even values of $m$ using the subcode of Theorem 5.12.

**Theorem 5.13.** *If* $m$ *is even, then* $d_4(BCH(2, m)) = 11$.

*Proof.* We will prove that $d_4(EBCH(2,m)) = 12$. The result then follows from Theorem 5.6. First we will derive some nice properties of the subcode in $\mathbb{F}_{16}$ and then use them to prove the theorem.

Because of Theorem 5.12 we have that the $[12,4,6]$ code is a subcode of EBCH(2,4) and that all of its codewords are contained in $\mathbb{F}_{16} \setminus \mathbb{F}_4$. We can therefore write the elements $x_i$ of $\mathbb{F}_{16} \setminus \mathbb{F}_4$ as $x_i = a_i\alpha + b_i$ where $a_i \in \mathbb{F}_4^*$, $b_i \in \mathbb{F}_4$ and $\alpha \notin \mathbb{F}_4$. Now every codeword $X$ of the $[12,4,6]$ subcode is a collection of these $x_i$. Let $I$ be the set of indices for which $x_i \in X$. The set $X$ is a codeword, so

$$\begin{aligned} 0 &= \sum_{i\in I} x_i \\ &= \sum_{i\in I}(a_i\alpha + b_i) \\ &= (\sum_{i\in I} a_i)\alpha + \sum_{i\in I} b_i. \end{aligned}$$

Since $\alpha \notin \mathbb{F}_4$ it must be true that $\sum_{i\in I} a_i = \sum_{i\in I} b_i = 0$. There is another equation that has to be satisfied:

$$\begin{aligned} 0 &= \sum_{i\in I} x_i^3 \\ &= \sum_{i\in I}(a_i\alpha + b_i)^3 \\ &= (\sum_{i\in I} a_i^3)\alpha^3 + (\sum_{i\in I} a_i^2 b_i)\alpha^2 + (\sum_{i\in I} a_i b_i^2)\alpha + \sum_{i\in I} b_i^3. \end{aligned}$$

Define $\mu_i$ as the coefficient of $\alpha^i$ in the above equation. The cardinality of $I$ is the same as that of $X$, so it is even. Because $a_i \in \mathbb{F}_4^*$, we know that $a_i^3 = 1$. Hence we conclude that $\sum_{i\in I} a_i^3 = \mu_3 = 0$. This is not necessarily true for $\mu_0 = \sum_{i\in I} b_i^3$, but since $b_i^3 \in \mathbb{F}_2$ we do know that $\mu_0$ is in $\mathbb{F}_2$ as well. Computing $\mu_1^2$ gives

$$\begin{aligned} \mu_1^2 &= (\sum_{i\in I} a_i b_i^2)^2 \\ &= \sum_{i\in I} a_i^2 b_i^4 \\ &= \sum_{i\in I} a_i^2 b_i \\ &= \mu_2. \end{aligned}$$

Putting it all into one equation, we get $\mu_1^2\alpha^2 + \mu_1\alpha = \mu_0$. Squaring this as well gives $\mu_1\alpha^4 + \mu_1^2\alpha^2 = \mu_0$. Adding up these equations results in $\mu_1(\alpha^4 + \alpha) = 0$. We choose $\alpha$ such that it is not in $\mathbb{F}_4$, so $\alpha^4 + \alpha \neq 0$. The only possibility left

is $\mu_1 = 0$. It immediately follows that $\mu_2 = \mu_0 = 0$. Now we have derived all properties we need to finish the proof.

Consider any finite field on $2^m$ elements and $m$ even. It does have $\mathbb{F}_4$ as a subfield. Pick $\alpha \in \mathbb{F}_{2^m} \setminus \mathbb{F}_4$. The set $X := \{a_i\alpha + b_i \mid i \in I\}$ does satisfy the equations $\sum_{x\in X} x = 0$ and $\sum_{x\in X} x^3 = 0$, since the coefficients of the powers of $\alpha$ you get when expanding the sums are always zero as we have just proven. The $[12, 4, 6]$ code is therefore a subcode of BCH(2,m) for even values of $m$. Thus the 4th-generalized Hamming weight for these codes is 12. □

Earlier we proved that $d_3(BCH(2, m)) = 10$ if $m$ is odd. For even values of $m$ we have the same result as a consequence of the previous theorem and the monotonicity of the weight hierarchy (see Theorem 4.1).

## 5.7 $d_4(BCH(2, m))$ for odd m

We have just proven that $d_4(EBCH(2, m)) = 12$ whenever $m$ is even. The case $m$ is odd remains an open problem. If the fourth generalized weight would be 12 for some $m$, then the code must have the unique $[12, 4, 6]$-code as a subcode. In Section 5.4 we discussed that a code $C$ is a subcode of EBCH(2,m) if and only if a system of equations derived from the basis vectors of $C$ has a solution. With the help of a computer program (see the following section) we did try to find the $[12, 4, 6]$-code as a subcode. We did not find a solution for $m \in \{5, 7, 9, 11, 13\}$. The finite field $\mathbb{F}_{2^m}$ has a third root of unity if and only if $m$ is even. This property manifests itself in several ways. That is why we believe that $d_4(EBCH(2, m)) > 12$ for odd $m$. Combined with the bound $d_4(BCH(2, m)) \leq 12$ given by van der Vlugt [30], we conjecture that $d_4(BCH(2, m)) = 12$ for odd $m$. But we have not been able to prove this conjecture. In Sections 5.9, 5.10 and 5.11 we will discuss some attempts we made to solve the case where $m$ is odd.

## 5.8 A computer program

In Section 5.4 we explained that in order to find a code as a subcode of $EBCH(2, m)$ we had to solve a system of equations. We do not know how to solve this system efficiently, therefore we will use a brute force method. There are some tricks that help us to do better than simply trying every possible solution. Suppose that we want to solve the system of equations associated with a $k$-dimensional code $C$ of length $n$; then we have variables $x_1, \ldots, x_n$ which must have distinct values. One of the most powerful tools at our disposal is the automorphism group. This group is 2-transitive on $\mathbb{F}_{2^m}$, so we can map any solution to a solution with $x_1 = 0$ and $x_2 = 1$. Hence we may assume without loss of generality that $x_1$ and $x_2$ indeed have these values. We can save more computing power with the automorphism $\phi : x \mapsto x^2$. This mapping keeps $x_1$ and $x_2$ in place and limits the choices for $x_3$ since we only have to select one element of every orbit of $\phi$. Most orbits have $m$ elements, so the time required to search

the solution space is reduced by about a factor $m$. If all but two variables of a codeword have been guessed or determined, then the remaining variables are uniquely determined by the roots of a quadratic equation (see Section 5.3). If only one variable is left, then it is equal to the sum of the other elements of the codeword (since the sum of the elements has to be zero). Note that the cubic equation does not necessarily hold in this case, so it has to be checked.

## 5.9 First approach: counting the number of solutions

### 5.9.1 Introduction

In this section we will try to count the total number of solutions to the equations of the $[12, 4, 6]$-subcode (see Section 5.4). This number also includes solutions that contain non-distinct values for the $x_i$. Counting solutions of the latter kind might be feasible (we have not tried so far) and subtracting this amount from the total number of solutions would give us the number of solutions we are really interested in. Hopefully this is zero for odd $m$.

In the next paragraph we will explain how to count the total number of solutions to a set of equations. Then we turn our focus to the specific equations produced by the $[12, 4, 6]$ code.

### 5.9.2 Number of solutions to a general set of equations

Let $f_i(x_1, \ldots, x_n) = 0$ for $1 \leq i \leq k$ be a system of $k$ polynomial equations in $n$ variables. Suppose that in each equation the variables are *separable*, that is $f_i(x_1, \ldots, x_n) = \sum_{j=1}^{n} f_{i,j}(x_j)$. The following property (see Lemma 2.17) of a character sum is used in counting the number of solutions

$$\sum_{s \in \mathbb{F}_{2^m}} \psi(sf(x_1, \ldots, x_n)) = \begin{cases} 2^m & \text{if } f(x_1, \ldots, x_n) = 0 \\ 0 & \text{if } f(x_1, \ldots, x_n) \neq 0 \end{cases} \tag{5.21}$$

We will count the total number of solutions $N$ to this system of equations in the same way as done in [16] or [21]. For convenience we write $x = (x_1, \ldots, x_n)$ and $a = (a_1, \ldots, a_k)$ whenever applicable.

$$\begin{aligned} (2^m)^k N &= \sum_{x \in \mathbb{F}_{2^m}^n} \left( \sum_{a_1 \in \mathbb{F}_{2^m}} \psi(a_1 f_1(x)) \right) \ldots \left( \sum_{a_k \in \mathbb{F}_{2^m}} \psi(a_k f_k(x)) \right) \\ &= \sum_{x \in \mathbb{F}_{2^m}^n} \sum_{a \in \mathbb{F}_{2^m}^k} \psi\left(a_1 f_1(x) + \ldots + a_k f_k(x)\right) \\ &= \sum_{a \in \mathbb{F}_{2^m}^k} \sum_{x \in \mathbb{F}_{2^m}^n} \psi\left( \sum_{i=1}^{n} \sum_{j=1}^{k} a_j f_{j,i}(x_i) \right) \end{aligned}$$

$$= \sum_{a \in \mathbb{F}_{2^m}^k} \sum_{x \in \mathbb{F}_{2^m}^n} \prod_{i=1}^{n} \psi \left( \sum_{j=1}^{k} a_j f_{j,i}(x_i) \right)$$

$$= \sum_{a \in \mathbb{F}_{2^m}^n} \prod_{i=1}^{n} \left( \sum_{x_i \in \mathbb{F}_{2^m}} \psi(a_1 f_{1,i}(x_i) + \ldots + a_k f_{k,i}(x_i)) \right)$$

Now we write for convenience $h_{i,a}(z) = a_1 f_{1,i}(z) + \ldots + a_k f_{k,i}(z)$, then we have to compute $\sum_{z \in \mathbb{F}_{2^m}} \psi(h_{i,a}(z))$ for all $1 \leq i \leq n$. If we succeed in doing so, we also know the number of solutions to the system of equations.

### 5.9.3 Equations of the $[12, 4, 6]$ code

The $[12, 4, 6]$ code is spanned by four codewords $c_1, \ldots, c_4$, so in order to find it as a subcode of $EBCH(2, m)$ we have to solve a system consisting of four linear and four cubic equations (see Section 5.4). We can write the linear equation for $c_j$ as

$$\sum_{i=1}^{n} c_{ij} x_j = 0 \tag{5.22}$$

and the cubic equation as

$$\sum_{i=1}^{n} c_{ij} x_j^3 = 0. \tag{5.23}$$

Multiplying the four cubic equations by $a_1, \ldots, a_4$, respectively, and the linear equations by $a_5, \ldots, a_8$, respectively, we get $h_{i,a}(x) = ux^3 + vx$, where $u = a_1 c_{1i} + \ldots + a_4 c_{i4}$ and $v = a_5 c_{1i} + \ldots + a_8 c_{4i}$. For all vectors $a \in \mathbb{F}_{2^m}^8$ we have to compute the character sum $S_{i,a} := \sum_{x \in \mathbb{F}_{2^m}} \psi(h_{i,a}(x))$ for every $1 \leq i \leq 12$. The product of the $S_{i,a}$ only contributes to the number of solutions if all $S_{i,a}$ are non-zero. We shall investigate which values $S_{i,a}$ can attain.

If $u$ and $v$ are both zero, then we have $S_{i,a} = q$. If precisely one of $u$ or $v$ is zero, then $S_{i,a} = 0$ since the mapping $x \mapsto x^3$ is a bijection if $m$ is odd and $\sum_{x \in \mathbb{F}_{2^m}} \psi(x) = 0$. We will derive a sufficient condition on $u$ and $v$ to have $\sum_{x \in \mathbb{F}_{2^m}} \psi(h_{i,a}(x) = 0$. Rewriting this charactersum by substituting $y = u^{1/3}x$ we get $\sum_{x \in \mathbb{F}_{2^m}} \psi(ux^3 + vx) = \sum_y \psi(y^3 + v/u^{1/3}y)$. We will try to find $c, d, \gamma \in \mathbb{F}_{2^m}$ such that $\psi(y^3 + v/u^{1/3}y) = \psi(y^3 + \gamma^2 y^2 + \gamma y + v/u^{1/3}y) = \psi((y+d)^3 + c)$. If we manage to do so, then it follows easily from $\mathrm{Tr}(\gamma^2 y^2) = \mathrm{Tr}(\gamma y)$ that the sum of characters $S_{i,a}$ is zero. Since $(y+d)^3 + c = y^3 + dy^2 + d^2 y + d^3 + c$ we get the following constraints on such values.

$$c + d^3 = 0 \tag{5.24}$$

$$d^2 = \gamma + v/u^{1/3} \tag{5.25}$$

$$d = \gamma^2 \tag{5.26}$$

Substituting 5.26 into 5.25, we get $\gamma^4 + \gamma + v/u^{\frac{1}{3}} = 0$. Such a $\gamma$ exists precisely when the polynomial $x^4 + x + v/u^{\frac{1}{3}}$ has a zero in $\mathbb{F}_{2^m}$. The following lemma shows that it does have a zero if and only if $\mathrm{Tr}(v/u^{\frac{1}{3}}) = 0$.

**Lemma 5.14.** *Let $s \in \mathbb{F}_{2^m}$ and $m$ odd, then the equation*

$$x^4 + x + s = 0 \tag{5.27}$$

*has a solution in $\mathbb{F}_{2^m}$ if and only if $Tr(s) = 0$.*

*Proof.* Define $\phi : \mathbb{F}_{2^m} \to \mathbb{F}$ by $\phi(x) = x^4 + x = x(x^3 + 1)$. This is a linear mapping because the characteristic of the field is two. If $\phi(a + b) = 0$, then it follows that either $a + b = 0$ or $(a + b)^3 = 1$. For odd values of $m$, $\mathbb{F}_{2^m}$ does not contain a third root of unity besides 1 itself, so in that case either $a = b$ or $b = a + 1$. Hence we know that $|\phi(\mathbb{F}_{2^m})| = 2^{m-1}$. If $s \in \phi(\mathbb{F}_{2^m})$ then $s = t^4 + t$ for some $t \in \mathbb{F}_{2^m}$ and $\mathrm{Tr}(s) = \mathrm{Tr}(t^4 + t) = \mathrm{Tr}(t) + \mathrm{Tr}(t) = 0$. Since $|\mathrm{Tr}_0| = 2^{m-1}$ we have $\phi(\mathbb{F}_{2^m}) = \mathrm{Tr}_0$. That is, the polynomial $x^4 + x + s$ has a solution if and only if $\mathrm{Tr}(s) = 0$. □

We showed that a vector $a \in \mathbb{F}_{2^m}^8$ possibly contributes to the total number of solutions only if $\mathrm{Tr}(v_i u_i^{1/3}) = 1$ for all $1 \le i \le 12$. We do not know whether this is a sufficient condition. It seems to be very complicated to determine for which vectors $a$ these trace equations are satisfied. To illustrate this a bit more we will write out some of those equations. As a generator matrix $G$ of the $[12, 4, 6]$-code we take

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{5.28}$$

The equations for the last three indices become

$$\begin{aligned} h_{10,a}(x) &= (a_3 + a_4)x^3 + (a_7 + a_8)x, \\ h_{11,a}(x) &= a_3 x^3 + a_7 x \\ h_{12,a}(x) &= a_4 x^3 + a_8 x. \end{aligned}$$

The corresponding trace equations then are

$$\mathrm{Tr}((a_7 + a_8)(a_3 + a_4)^{-1/3}) = 1 \tag{5.29}$$

$$\mathrm{Tr}(a_7 a_3^{-1/3}) = 1 \tag{5.30}$$

$$\mathrm{Tr}(a_8 a_4^{-1/3}) = 1. \tag{5.31}$$

It is not so hard to choose $a_3, a_4, a_7$ and $a_8$ such that the last two trace equations are met, but we have no clue how to pick them such that the first equation is satisfied as well. To make things worse, there are nine other trace equations to be met.

## 5.10 Second approach: trace equations

To prove that $d_4(BCH(2, m)) \neq 12$ if $m$ is odd, we have to show that the system of equations associated with the $[12, 4, 6]$-code (see Section 5.4) does not have a

solution. As a generator matrix $G$ of the $[12, 4, 6]$-code we take 5.28. Variable $x_i$ then corresponds with the $i$-th column of $G$. Because of the 2-transitivity of the automorphism group we may assume without loss of generality that $x_1 = 0$ and $x_2 = 1$. If $x_3$, $x_4$ and $x_7$ are known, then the other variables are roots of quadratic polynomials (see Lemma 5.7) or the sum of such roots. Expressing $x_5, x_6, x_8, \ldots, x_{12}$ in terms of $x_3, x_4$ and $x_7$, we obtain

$$\{x_5, x_6\} = (1 + x_3 + x_4)\mathrm{RT}_{2^m}\left(1 + \frac{1 + x_3^3 + x_4^3}{(1 + x_3 + x_4)^3}\right), \tag{5.32}$$

$$\{x_8, x_9\} = (1 + x_3 + x_7)\mathrm{RT}_{2^m}\left(1 + \frac{1 + x_3^3 + x_7^3}{(1 + x_3 + x_7)^3}\right), \tag{5.33}$$

$$\{x_{10}, x_{11}\} = (x_3 + x_4 + x_7)\mathrm{RT}_{2^m}\left(1 + \frac{x_3^3 + x_4^3 + x_7^3}{(x_3 + x_4 + x_7)^3}\right) \tag{5.34}$$

$$x_{12} = 1 + x_5 + x_8 + x_{10}. \tag{5.35}$$

A set of roots $\mathrm{RT}(s)$ is not empty if and only if $\mathrm{Tr}(s) = 0$ according to Lemma 2.12. Thus we have to find a solution for $x_3, x_4$ and $x_7$ such that all three traces corresponding to (5.32),(5.33) and (5.34) are zero. We have not been able to describe the solutions to each seperate trace equation, let alone the solutions to all three of them simultaneously.

## 5.11 Third approach: greatest common divider

Suppose that two codewords $X_1$ and $X_2$ have an element $x$ in common. Lemma 5.7 tells us how to obtain a second-degree polynomial from a codeword if all but two of its elements are known. The zeros of this polynomial are the remaining elements of the codeword. If we take $x$ as one of the unknowns, we thus get polynomials $f_1(z) = a_1 z^2 + a_1^2 z + a_1^3 + b_1$ and $f_2(z) = a_2 z^2 + a_2^2 z + a_2^3 + b_2$ with the property that $f_1(x) = f_2(x) = 0$. The greatest common divider of $f_1$ and $f_2$ therefore contains the factor $z - x$. This observation could enable us to derive additional constraints on the elements of the codewords.

As (another) generator matrix $G'$ of the $[12, 4, 6]$-code we could take

$$G' = \left(\begin{array}{ccccccc|c|cccc} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right). \tag{5.36}$$

The matrix $G'$ contains the all-one column followed by the identity matrix. We have to pick $x_1, \ldots, x_7$ such that the polynomials corresponding to each row of $G'$ have zeros in $\mathbb{F}_{2^m}$ (see Lemma 5.7). These zeros then determine $x_8, \ldots, x_{12}$. We could simply try to solve the trace equations to determine whether the set of roots are non-empty. By doing so, we overlook the fact that the polynomials must have a common zero, namely $x_8$.

# Chapter 6

# Finding a subcode of minimal weight

## 6.1 Introduction

Since it is not always possible to determine the generalized Hamming weights of a code in a purely mathematical way, we would like to design a computer algorithm that calculates $d_r(C)$. Janwa and Lal [18] already gave an algorithm for computing the weight hierarchy of cyclic codes, but it does not seem to go through all possible $r$-dimensional subspaces very effectively. We have developed an algorithm that avoids, as much as possible, examining a subspace twice. So far, we have been able to compute the complete weight hierarchy of BCH(2,7).

First we will give some necessary definitions and explain how the algorithm works. After that we will go into more detail of the implementation.

## 6.2 Some definitions

Let $C$ be a $[n, k, d]$ code and let $D = \{c_1, \ldots, c_N\}$ be a subset, not necessarily a subcode, of $C$. A tuple of codewords in $D$ is represented by the tuple of their indices. The set of indices $\{1, \ldots, N\}$ is also written as $I$.

A *sorted* $k$-tuple $x = (x_1, \ldots, x_k) \in I^k$ is a tuple with the property that $x_1 < \ldots < x_k$. The operator $S$ permutes the elements of a tuple $x \in I^k$ such that $S(x)$ is a sorted tuple (we also say that $S$ sorts $x$). For any two tuples $x \in I^k$ and $y \in I^m$ we define their concatenation $(x, y) := (x_1, \ldots, x_k, y_1, \ldots, y_m) \in I^{k+m}$. A tuple $y$ is called a *child* of tuple $x$ if there is a tuple $z$ such that $y = (x, z)$ (or: $x$ is a parent of $y$). The tuples are subject to the lexicographical ordering, defined as follows.

**Definition 6.1.** *For any two tuples $x \in I^k$ and $y \in I^m$ we write $x < y$ if there is a $t \leq \min(k, m)$ such that $x_i = y_i$ for all $i < t$ and $x_t < y_t$ or if $y$ is a child of $x$.*

Tuples correspond to a set of codewords and can thus be identified with subcodes of $C$ in a natural way.

**Definition 6.2.** *The subcode associated with a tuple $x \in I^k$ is denoted by $V(x)$ and is defined by*

$$V(x) = \langle \{c_{x_1}, \ldots, c_{x_k}\} \rangle. \tag{6.1}$$

Automorphisms of the code act on tuples $x$ by their action on the corresponding codewords.

## 6.3 The search algorithm

In order to compute $d_r(C)$ of a $k$-dimensional code, we could try out all subspaces of $C$ spanned by any $r$ codewords, but this rapidly becomes computationally unfeasible. Theorem 2.8 by Dodunekov and Manev [8] states that if the weight of a (sub)code exceeds the Griesmer bound by $t$, then it is spanned by codewords of weight at most $d + t$. This means that we only have to examine subspaces spanned by codewords of low weight. The value of $t$ is not known in advance, so we have to guess it. When the search is over we know whether our guess was good enough. If it was not, we only have an upperbound on $d_r(C)$.

We let $D$ be the set of all codewords of weight at most $d + t$, that is

$$D := \{c \in C \mid w(c) \leq d + t\}. \tag{6.2}$$

If the cardinality of $D$ is $N$, the index set is $I = \{1, \ldots, N\}$. Our goal is to find a $r$-tuple $x \in I^r$ such that $\dim(V(x)) = r$ and $w(V(x))$ is as small as possible. Without loss of generality we may assume that this tuple is a sorted tuple. We will search for the best tuple with the help of backtracking. For each sorted tuple of size at most $r$ we have a vertex in the search tree; the sorted $k$-tuples are placed at depth $k$ in the search tree. Edges only exist between a $k$-tuple and all its children of size $k + 1$. Traversing the tree is done in lexicographical order.

The performance of the algorithm heavily depends on how well we can prune the search tree. We will use one of the following four reasons to stop searching a branch that grows from tuple $x = (x_1, \ldots, x_k)$.

1. The dimension of $V(x)$ is smaller than $k$.
2. The weight of $V(x)$ is at least as large as the best weight found so far.
3. There is another basis for $V(x)$ and this new basis can be represented by a sorted tuple smaller than $x$.
4. There is an automorphism $\sigma$ such that $\sigma(V(x))$ can be represented by a sorted tuple smaller than $x$.

The first two reasons are pretty self explanatory. Any $r$-tuple $y$ that has $x$ as a parent only corresponds with a $r$-dimensional subspace if the codewords

$c_{x_1}, \ldots, c_{x_k}, c_{y_{k+1}}, \ldots, c_{y_r}$ are linear independent. Of course this does not hold if $\dim(V(x)) < k$. As for the weight of $V(y)$, every non-zero position in $V(x)$ is also a non-zero position in $V(y)$, so we have $w(V(y)) \geq w(V(x))$.

The third and fourth rule depend on finding a smaller tuple representing the same space as $V(x)$. Because we traverse the tree in lexicographical order, we have already decided whether such smaller tuple is the one we are looking for.

The next lemma gives us a sufficient condition to prune, using the third rule.

**Lemma 6.3.** *If $c_a \in c_{x_k} + V((x_1, \ldots, x_{k-1}))$ and $a < x_k$, then there is a sorted tuple $x'$ such that $V(x) = V(x')$ and $x' < x$ and for each child $y$ of $x$ there is a sorted tuple $y'$ such that $V(y) = V(y')$ and $y' < y$.*

*Proof.* De subspace $V(x)$ is spanned by $c_{x_1}, \ldots, c_{x_k}$. If we replace $c_{x_k}$ by $c_a$ we still have a basis for $V(x)$, because of the assumption. Therefore we can represent $V(x)$ by the sorted tuple $x' = S((x_1, \ldots, x_{k-1}, a))$, which is obviously smaller than $x$.

Each child $y = (x, z)$ of $x$ represents the same subspace as $y' = (x', b)$ and because $x' < x$ we also have $y' < y$. □

The last pruning rule is stated in the following lemma.

**Lemma 6.4.** *If there is an automorphism $\sigma$ such that $S(\sigma(x)) < x$, then for every child $y$ of $x$ we have $S(\sigma(y)) < y$.*

*Proof.* We can write every child as $y = (x, z)$. Then

$$\begin{aligned} S(\sigma(y)) &= S((\sigma(x), \sigma(z))) \\ &< (S(\sigma(x)), S(\sigma(z))) \\ &< x \\ &< (x, z) \\ &= y \end{aligned}$$

□

In a later section we will explain how to find an automorphism that maps a tuple to a smaller sorted tuple.

The algorithm in its most basic form is given in Algorithm 1. It is initialized with the smallest non-empty tuple $x = (1)$ and an estimate $\hat{d}_r$ for $d_r(C)$. In lines 4 to 7 we try to cut off the search tree in tuple $x = (x_1, \ldots, x_k)$. Whenever $x$ fails one the tests, the remaining tests need not to be performed. The algorithm can be modified at line 9 to also return the subcode for which $d_r(C)$ is attained.

## 6.4 Implementation of the algorithm

### 6.4.1 Representation of binary vectors on a computer

An obvious way to represent a binary vector in a computer program is by an array of the same length, say $n$. Adding two vectors then would require a

**Algorithm 1** Computing $d_r(C)$

**Input:** $D = \{c_1, \ldots, c_N\}$, dimension $r$.
**Output:** $d_r(C)$

```
1:  x ← (1)
2:  d̂_r ← ∞
3:  while search is not over do
4:      Test dim(V(x)) = k.
5:      Test w(V(x)) < d̂_r.
6:      Test {a | a < x_k ∧ c_a ∈ c_{x_k} + V((x_1, ..., x_{k-1}))} = ∅.
7:      Test {σ | S(σ(x)) < x} = ∅.
8:      if x passed all tests and k = r then
9:          d̂_r = w(V(x))
10:     else if passed all tests and k < r then
11:         x ← (x, x_k + 1)
12:     else
13:         x ← backtrack(x)
14:     end if
15: end while
16: return d̂_r
```

multiple of $n$ elementary computer instructions. Computing the weight of a vector would demand about the same amount of time. Since the algorithm can easily take several hours to complete, it is wise to use a speedier representation.

Most computers are able to perform operations on 32-bit words (or even more) in a single instruction. If we chop a binary vector in pieces of 32 bits, we can store it in an array whose size is 32 times smaller than it was before. In this way we significantly reduce the time needed to add two vectors. Counting the number of 1's in a vector is also faster, as it no longer depends on the length of the vector, but on the weight.

### 6.4.2 Storage of codewords and automorphisms

Storing additional data in memory could increase the speed of the algorithm. This idea is commonly known as time-memory trade-off. Since speed is the bottleneck, we will sacrifice memory in favor of performance.

For each codeword we store the following (redundant) data.

- **message:** the vector $a \in \mathbb{F}_2^k$.
- **encoded message:** the vector $c = aG$, where $G$ is the generator matrix.
- **index:** index of the codeword in the list $D$.
- **minimal automorphisms:** there is a codeword $c_{min} \in D$ of smallest index in the same orbit; the minimal automorphisms are the automorphisms that map the codeword to $c_{min}$.

An automorphism $\sigma$ of the code is a linear mapping from $C$ onto itself, but it can also be seen as a linear mapping from the message space $\mathbb{F}_2^k$ onto itself. We can thus write $\sigma(a.G) = a.P.G$ for some $k \times k$-matrix $P$. Storing this matrix $P$ allows for faster computation of the image of a codeword.

### 6.4.3 Determining the index of a codeword

As input to the algorithm we have a list $D$ from which we have to choose $r$ codewords that span a $r$-dimensional subspace of minimal weight. We have to check regularly whether a codeword is in $D$, and if so, what its index in this list is. If the dimension $k$ of the code is not too large, we can perform this check in constant time by using an array of length $2^k$. Each codeword is mapped to a position in this array; the value at this position is either the index in $D$ of the codeword or infinity if it is not in the list.

If the dimension of the code is too large, we have to resort to binary search, which worsens the look-up time to $\mathcal{O}(\log(N))$. Note that in this case the running time is already quite impractical due to the likely large size of $D$.

### 6.4.4 Determining the dimension

The algorithm only examines a tuple $x$ if $\dim(V((x_1, \ldots, x_{k-1}))) = k - 1$. Because of this observation, we know for sure that if $\dim(V(x)) < k$, it must be $c_{x_k}$ that can be written as the sum of some $c_{x_i}$. Thus there exists an $x_i < x_k$ such that $c_{x_i} \in c_{x_k} + V((x_1, \ldots, x_{k-1}))$. The algorithm already does this check when searching for another basis, so there is no need to seperately test for $\dim(V(x)) < k$.

### 6.4.5 Computing the weight

Counting the number of 1's in a vector is rather straightforward, but still we can use a little trick. Instead of computing the whole weight of $V(x)$, we count the number of ones that are in the support of $V(x)$, but not in the support of $V((x_1, \ldots, x_{k-1}))$. We do so with the aid of a bitmask stored in a vector $mask(k)$ defined by

$$\text{mask}(k)_i = \begin{cases} 0 & \text{if } i \in \chi(V((x_1, \ldots, x_k))) \\ 1 & \text{if } i \notin \chi(V((x_1, \ldots, x_k))). \end{cases} \tag{6.3}$$

With this bitmask we can compute the weight of $V(x)$ and the mask itself in a recursive fashion, using the bitwise operators AND and NOT,

$$w(V(x)) = w(V((x_1, \ldots, x_{k-1}))) + w(\text{mask}(k-1) \text{ AND } c_{x_k}) \tag{6.4}$$

and

$$\text{mask}(k) = \text{mask}(k-1) \text{ AND NOT } c_{x_k}. \tag{6.5}$$

### 6.4.6 Change of basis

We store all codewords of $V((x_1, \ldots, x_k))$ in an array named *span* and of length $2^r$. The first $2^i$ elements are precisely the codewords in $V((x_1, \ldots, x_i))$. To determine all codewords in $V(x)$ we only have to compute $c_{x_k} + V((x_1, \ldots, x_{k-1}))$. Whenever we encounter a codeword whose index is smaller than $x_k$ we can stop searching because of Lemma 6.3. This part of the algorithm is listed in Algorithm 2.

**Algorithm 2** Test $\{a \mid a < x_k \wedge c_a \in c_{x_k} + V((x_1, \ldots, x_{k-1}))\} = \emptyset$

```
for i = 1 to 2^{k-1} do
    v ← c_{x_k} + span(i)
    if index(v) < x_k then
        return false
    else
        span(2^{k-1} + i) ← v
    end if
end for
```

### 6.4.7 Finding the best automorphism

We proved in Lemma 6.4 that if there is an automorphism that maps $x$ to a smaller sorted tuple, then we can prune the search tree. We would like to find the automorphism $\sigma$ that maps $x$ to the smallest tuple, say $z$, possible. One of the elements of $x$ has to be mapped to $z_1$. If we define the *minimal* automorphisms of an index $i$ as the automorphisms that map $i$ to the smallest index possible, then we are certain that we can find $\sigma$ in the collection of minimal automorphism of $x_1, \ldots, x_k$. Algorithm 3 shows in pseudocode how to find such $\sigma$.

**Algorithm 3** Testing $\{\sigma \mid S(\sigma(x)) < x\} = \emptyset$

**Input:** sorted tuple $x = (x_1, \ldots, x_k)$, list of codewords.
**Output:** true if and only if there is such $\sigma$

```
1: for i = 1 to k do
2:     list ← c_{x_i}.minimalAutomorphisms
3:     for all automorphism σ in list do
4:         if S(σ(x)) < x then
5:             return true
6:         end if
7:     end for
8: end for
9: return false
```

Precomputing the minimal automorphisms can be done using some kind of sieve method, as depicted in Algorithm 4.

**Algorithm 4** Computing the minimal automorphisms

**Input:** a list of codewords

1: **repeat**
2: select smallest unmarked index $i$
3: **for all** automorphism $\sigma$ **do**
4: $c \leftarrow \sigma(c_i)$
5: $c$.addMinimalAutomorphism($\sigma^{-1}$)
6: mark index of $c$
7: **end for**
8: **until** all indices are marked

## 6.5 Some results

So far only the first three generalized Hamming weights of both BCH(2,7) and its dual were known. With the help of another computer program developed in Chapter 5 we have determined that $d_4(BCH(2,7)) = 12$. By Theorem 4.2, it is sufficient to compute $d_r(BCH(2,7)^\perp)$ for $r = 4, 5, 6$. Instead of searching in $BCH(2,7)^\perp$ we did so in $EBCH(2,7)^\perp$ and translated the obtained results using Theorem 4.10. The list of codewords out of which we had to select a basis nearly doubled. But the number of automorphisms was raised to the square. This made pruning the search tree far more effective. It took about 24 hours on an average computer to determine $d_6(EBCH(2,7)^\perp)$. Since then we have been able to speed up the implementation a bit, so it can be done even faster.

All values we have been able to compute are listed in the table below.

| | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| $BCH(2,7)$ | 5 | 8 | 10 | 12 | | |
| $BCH(2,7)^\perp$ | 56 | 84 | 98 | 105 | 110 | 114 |
| $BCH(2,8)$ | 5 | 8 | 10 | 11 | | |
| $BCH(2,8)^\perp$ | 112 | 168 | 198 | 215 | | |
| $BCH(3,5)$ | 7 | 11 | | | | |
| $BCH(3,5)^\perp$ | 8 | 12 | 14 | 15 | 16 | 20 |
| $BCH(3,6)$ | 7 | 11 | 13 | 14 | 15 | |
| $BCH(3,6)^\perp$ | 16 | 24 | 32 | 39 | 43 | |
| $BCH(3,7)^\perp$ | 48 | 72 | 84 | 90 | | |

# Chapter 7

# Conclusion

In the beginning of this project we wanted to gain knowledge on shifting for arbitrary linear codes. In the end we spend most of our time on generalized Hamming weights. Therefore our initial objective is not accomplished. Although there is no indication that there is anything to be learned on this matter, without further research the status quo is certainly maintained.

The technique used so far to determine generalized Hamming weights of BCH(2,m) is geometric in nature. Codewords of minimal weight are identified with lines. By counting, it is then shown that there exist subcodes that meet the Griesmer bound. This approach works fine for the first three weights. Based on computer search we conjecture that, if $m$ is odd, $d_4(BCH(2,m)) = 12$, which exceeds the Griesmer bound. New tools are needed to proof this conjecture and determine more weights in the hierarchy. It might be wise to examine the extended BCH(2,m) code and translate results obtained there back to BCH(2,m) by $d_r(EBCH(2,m)) = d_r(BCH(2,m)) + 1$. The advantage of this approach is that the automorphism group of EBCH(2,m) is significantly larger. Unfortunately we have no further clues. The good news is that we have been able to proof that $d_4(BCH(2,m)) = 11$ if $m$ is even.

We have also designed an algorithm with which we have been able to compute the complete weight hierarchy of BCH(2,7) and several other weights. Due to a lack of time, we did not study this algorithm extensively. It is therefore worthwile to look for improvements. Especially when the automorphism group is small, the performance suffers. There is also work to be done on the implementation of the algorithm. At the moment it is fully functional, but certainly not user friendly. It would be a good idea to develop an application that computes the weight hierarchy of a code, or at least returns upperbounds on the weights, with the push of several buttons. Currently the only supported codes are the cyclic codes and the duals of extended BCH codes. More codes could be added to this list.

# Bibliography


[1] E. F. Assmus, Jr., H. F. Mattson, Jr., "New 5-designs", J. Comb. Theory, vol. 6, pp. 122-151, 1969.

[2] L. Bahl, J. Cocke, F. Jelinek, J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate", IEEE Trans. Inform. Theory, vol.20, no. 2, pp. 284-287, March 1974.

[3] Angela I. Barbero, Carlos Munuera, "The weight hierarchy of Hermitian codes", SIAM J. Discrete Math., vol. 13, no. 1, pp. 79-104, January 2000.

[4] T. Berger, "The automorphism group of double-error-correcting BCH codes", IEEE Trans. Inform. Theory, vol. 40, no. 2, pp. 538-542, March 1994.

[5] Elwyn R. Berlekamp, Robert J. McEliece, Henk C.A. van Tilborg, "On the inherent intractability of certain coding problems", IEEE Trans. Inform. Theory, vol IT-24, no. 3, pp. 384-386, May 1978.

[6] A. E. Brouwer, Tom Verhoeff, "An updated table of minimum-distance bounds for binary linear codes", IEEE Trans. Inform. Theory, vol. 39, no. 2, pp. 662-677, March 1993.

[7] H. Chung, "The 2nd generalized Hamming weight of double-error-correcting binary BCH codes and their dual codes", Lecture Notes in Computer Science, vol. 539, New York: Springer-Verlag, pp. 118-129, 1991.

[8] S. M. Dodunekov, N. L. Manev, "An improvement of the Griesmer bound for some small minimum distances", Discr. Appl. Math., vol. 12, pp. 103-114, 1985.

[9] G. L. Feng, K. K. Tzeng, V. K. Wei, "On the generalized Hamming weights of several classes of cyclic codes", IEEE Trans. Inform. Theory, vol.38, no. 3, pp. 1125-1130, May 1992.

[10] G. D. Forney, "Coset codes - part II: binary lattices and related codes", IEEE Trans. Inform. Theory, vol. 34, no. 5, pp. 1152-1187, September 1988.

[11] G. D. Forney, Jr., "Dimension/ length profiles and trellis complexity of linear block codes", IEEE Trans. Inform. Theory, vol. 40, no. 6, pp. 1741-1752, November 1994.

[12] G. van der Geer, M. van der Vlugt, "On generalized Hamming weights of BCH codes", IEEE Trans. Inform. Theory, vol. 40, no. 2, pp. 543-546, March 1994.

[13] Gerard van der Geer, Marcel van der Vlugt, "How to construct curves over finite fields with many points", arXiv:alg-geom/9511005 v2, May, 1996.

[14] J. H. Griesmer, "A bound for error-correcting codes", IBM J. Res. Develop., vol. 4, pp. 532-542, November 1960.

[15] Tor Helleseth, Torleiv Kløve, Øyvind Ytrehus, "Generalized Hamming weights of linear codes", IEEE Trans. Inform. Theory, vol. 38, no. 3, pp. 1133-1140, May 1992.

[16] Tor Helleseth, "On the covering radius of cyclic linear codes and arithmic codes", Discr. Appl. Math., vol. 11, pp. 157-173, 1985.

[17] H. D. L. Hollmann, personal communication.

[18] H. Janwa, A. K. Lal, "On the generalized Hamming weights of cyclic codes", IEEE Trans. Inform. Theory, vol. 43, no. 1, pp. 299-308, January 1997.

[19] Tadao Kasami, Shu Lin, W. Wesley Peterson, "New generalizations of the Reed-Muller codes part I: primitive codes", IEEE Trans. Inform. Theory, vol IT-14, no. 2, pp. 189-199, March 1968.

[20] T. Kasami, "Weight distributions of Bose-Chandhuri-Hocquenghem codes", Chapter 20 in R.C. Bose and T.A. Dowling (eds.) Proceedings of the conference on combinatorial mathematics and its applications (April 10-14, 1967), The University of North Carolina Press, Chapel Hill, N.C., 1968.

[21] Rudolf Lidl, Harald Niederreiter, "Finite fields", volume 20 of encyclopedia of mathematics and its applications, Addison-Wesley, 1983.

[22] Jacobus H. van Lint, Richard M. Wilson, "On the minimum distance of cyclic codes", IEEE Trans. Inform. Theory, vol. 32, no. 1, pp. 23-40, January 1986.

[23] F. J. MacWilliams, N. J. A. Sloane, "The theory of error-correcting codes", North Holland Publishing company,1977.

[24] Robert J. McEliece, "On the BCJR trellis for linear block codes", IEEE Trans. Inform. Theory, vol. 42, no. 4, pp. 1072-1092, July 1996.

[25] Douglas Muder, "Minimal trellises for block codes", IEEE Trans. Inform. Theory, vol. 34, no. 5, pp. 1049-1053, September 1988.

[26] L. H. Ozarow, A. D. Wyner, "Wire-tap-channel II", AT&T Bell labs technical journal, vol. 63, pp. 2135-2157, 1984.

[27] C. Shim, H. Chung, "On the second generalized Hamming weight of the dual code of a double-error-correcting binary BCH code", IEEE Trans. Inform. Theory, vol. 41, no. 3, pp. 805-808, May 1995.

[28] J. Simonis, "The effective length of subcodes", AAECC, pp. 371-377, 1994.

[29] Henk van Tilborg, "Error-correcting codes - a first course", Studentlitteratur, Lund, 1993.

[30] M. van der Vlugt, "A note on generalized Hamming weights of BCH(2)", IEEE Trans. Inform. Theory, vol. 42, no. 1, pp. 254-256, January 1996.

[31] Victor K. Wei, Kyeoncheol Yang, "On the generalized Hamming weights of product codes", IEEE Trans. Inform. Theory, vol. 39, no. 5, pp. 1709-1713, September 1993.

[32] Victor K. Wei, "Generalized Hamming weights for linear codes", IEEE Trans. Inform. Theory, vol. 37, no. 5, pp. 1412-1418, September 1991.