Master's Thesis

Title

Methods to Determine the Local Weight Distribution of Binary Linear Block Codes Using the Automorphism Group and Code Structure

Supervisor
Professor Toru Fujiwara

by
Kenji Yasunaga

February 17, 2005

Department of Multimedia Engineering
Graduate School of Information Science and Technology
Osaka University

Master's Thesis

Methods to Determine the Local Weight Distribution of Binary Linear Block Codes Using the Automorphism Group and Code Structure

Kenji Yasunaga

Abstract

For a binary linear block codes, the local weight distribution is the weight distribution of zero neighbors in the code. Knowledge of the local weight distribution is valuable for the error performance analysis of the code. Using the local weight distribution instead of the (global) weight distribution we could give the tighter upper bound on the error probability for soft decision decoding over AWGN channel.

In this thesis, some methods to determine the local weight distribution of binary linear codes are presented. Two approaches are studied: a computational approach and a theoretical approach. As a computational method, the algorithm for computing the local weight distribution of codes using the automorphism group of the codes is proposed. In this algorithm, a code is considered as the set of cosets of a subcode, and the set of cosets is partitioned into equivalence classes. Then, the weight distributions of zero neighbors only for each representative cosets of equivalence classes are computed. As a theoretical method, relations between the local weight distribution of a code, its extended code, and its even weight subcode are studied.

As a result, the local weight distribution of some of extended primitive BCH codes, Reed-Muller codes, primitive BCH codes, punctured Reed-Muller codes, and even weight subcodes of primitive BCH codes and punctured Reed-Muller codes are determined.

Keywords

local weight distribution, binary linear code, automorphism group, zero neighbor, coset, primitive BCH code, Reed-Muller code

Contents

1 Introduction

In a binary linear code, a zero neighbor is a codeword whose Voronoi region shares a facet with that of the all-zero codeword [1]. The *local weight distribution* [2] (or *local distance profile* [1, 6, 13, 14, 15]) of a binary linear code is defined as the weight distribution of zero neighbors in the code. Knowledge of the local weight distribution of a code is valuable for the error performance analysis of the code. On the problem of error performance evaluation for soft decision decoding over AWGN channel, the *union bound* is commonly used. The union bound gives an upper bound of error probability, in most cases the *global weight distribution* (or *weight distribution*) of a code is used. Usage of the local weight distribution instead of the global weight distribution could provide a tighter upper bound than the usual union bound [6].

Formulas for local weight distribution are only known for certain classes of codes, Hamming codes and second-order Reed-Muller codes. Although an efficient method to examine zero neighborship of codeword is presented in [1], the computation for obtaining the local weight distribution is a very time-consuming task. Thus, we can determine the local weight distributions only for the codes with small dimensions.

In this thesis, some methods to determine the local weight distribution of binary linear block codes are studied. Two approaches are studied: A computational approach and a theoretical approach. The idea of the computational approach is based on that of the algorithm of computing the weight distribution for primitive BCH codes in [7]. In this algorithm, a code is considered a set of cosets of a subcode, and the set of cosets are partitioned into equivalence classes with an invariance property. The weight distributions only for each representative cosets are computed. Thereby the computational complexity is reduced. This idea can be applied to local weight distribution. The local weight distributions for some of extended primitive BCH codes and Reed-Muller codes are obtained using this computational approach. As a theoretical approach, relations between the local weight distributions of a code, its extended code, and its even weight subcode are studied. As a result, the local weight distributions for some of primitive BCH codes, punctured Reed-Muller codes, and their even weight subcodes are obtained.

The outline of this thesis is as follows. In Chapter 2, definitions, properties, and known results with respect to local weight distribution are given.

In Chapter 3, the algorithm for computing the local weight distribution is proposed. The algorithm uses the automorphism group of the code. Agrell pointed out that the automorphism group of a code can help reduce the complexity for computing the local weight distribution [1]. Using the automorphism group of cyclic codes, i.e. cyclic permutations,

Mohri et al. obtained the local weight distributions of the primitive BCH codes of length 63 [13, 14, 15]. In this algorithm, representative codewords of cyclic permutations in a code are generated efficiently, and it was a key subalgorithm. An algorithm for the other known automorphism group of codes, e.g. the affine group of extended primitive BCH codes and the general affine group of Reed-Muller codes, is studied in this chapter. No efficient way of generating the representative codewords for these groups of permutations is known. To overcome this problem, in the proposed algorithm, *cosets* of a linear subcode are used. This idea is used in [7] for computing the weight distribution of primitive BCH codes.

In Chapter 4, two methods for improving the algorithm proposed in Chapter 3 are presented. First one uses the trellis structure of a code. Second one intends a full use of the automorphism group of a code.

In Chapter 5, a theoretical method to determine the local weight distribution using code structure is presented. Relations between the local weight distributions of a code, its extended code, and its even weight subcode are given. Some local weight distributions are obtained using these relations.

In Chapter 6, the local weight distributions that are obtained using the methods described in the previous chapters are presented. For primitive BCH codes, the local weight distributions of the $(127, k)$ codes for $k \leq 50$, their extended codes, and their even weight subcodes are obtained. For Reed-Muller codes, the local weight distributions of the third-order Reed-Muller code of length 128, its punctured code, and the even weight subcode of the punctured code are obtained.

Chapter 7 represents conclusions of this thesis and future research directions.

2 Local Weight Distribution of Binary Linear Codes

In this chapter, the local weight distribution of binary linear block codes is defined. Some properties, known results, and applications for local weight distribution are presented.

2.1 Definitions

Let C be a binary (n, k) linear block code. Let $\mathbf{R}^n$ be the n-dimensional Euclidean space. $\mathbf{R}^1 = \mathbf{R}$ is the set of real numbers. Define a mapping s from $\{0, 1\}$ to $\mathbf{R}$ as $\mathrm{s}(0) = 1$ and $\mathrm{s}(1) = -1$. The mapping s is naturally extended to one from $\{0, 1\}^n$ to $\mathbf{R}^n$, denoted by $\mathbf{s}$. A codeword $\boldsymbol{v} = (v_1, v_2, \ldots, v_n) \in C$ is transmitted as $\mathbf{s}(\boldsymbol{v}) = (\mathrm{s}(v_1), \mathrm{s}(v_2), \ldots, \mathrm{s}(v_n))$ with BPSK modulation (antipodal signaling) on an AWGN channel. $S = \{\mathbf{s}(\boldsymbol{v}) : \boldsymbol{v} \in C\}$ is called a signal set. $\mathbf{R}^n$ is partitioned into *Voronoi regions* of codewords.

Definition 1 (Voronoi region)**.** The Voronoi region of $\boldsymbol{v} \in C$ is a set of closest vectors in $\mathbf{R}^n$ to $\boldsymbol{v}$, that is,

$$\{\boldsymbol{x} \in \mathbf{R}^n : d_E(\boldsymbol{x}, \boldsymbol{v}) \leq d_E(\boldsymbol{x}, \boldsymbol{v}'), \quad \forall \boldsymbol{v}' \in C \setminus \{\boldsymbol{v}\}\}, \tag{2.1}$$

where $\mathbf{0} = (0, 0, \ldots, 0)$ and $d_E(\boldsymbol{x}, \boldsymbol{y})$ is the squared Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathbf{R}^n$.

The shape of the Voronoi region determines almost all significant properties for communications [6]. If the signal set S is *geometrically uniform* [6], all the Voronoi regions have the same shape. A signal set of a binary linear code with BPSK modulation is *geometrically uniform*. Then, for a binary linear code C, the Voronoi region of the all-zero codeword can be used as the representative of all codewords in C. A codeword whose Voronoi region shares the facet with that of the all-zero codeword is said to be a *zero neighbor* (refer to Figure 2.1).

Definition 2 (Zero neighbor)**.** For $\boldsymbol{v} \in C$, define $\boldsymbol{m}_0 \in \mathbf{R}^n$ as $\boldsymbol{m}_0 = \frac{1}{2}(\mathbf{s}(\mathbf{0}) + \mathbf{s}(\boldsymbol{v}))$. The codeword $\boldsymbol{v}$ is a zero neighbor if and only if

$$d_E(\boldsymbol{m}_0, \mathbf{s}(\boldsymbol{v})) \;=\; d_E(\boldsymbol{m}_0, \mathbf{s}(\mathbf{0})) < d_E(\boldsymbol{m}_0, \{\mathbf{s}(\boldsymbol{v}')), \quad \text{for any } \boldsymbol{v}' \in C \setminus \{\mathbf{0}, \boldsymbol{v}\}, \tag{2.2}$$

The all-zero codeword itself is not a zero neighbor. A zero neighbor is called a *minimal* codeword in [3]. A set of all zero neighbors in C is called the *projecting set* of C in [8].

The local weight distribution is defined as follows:

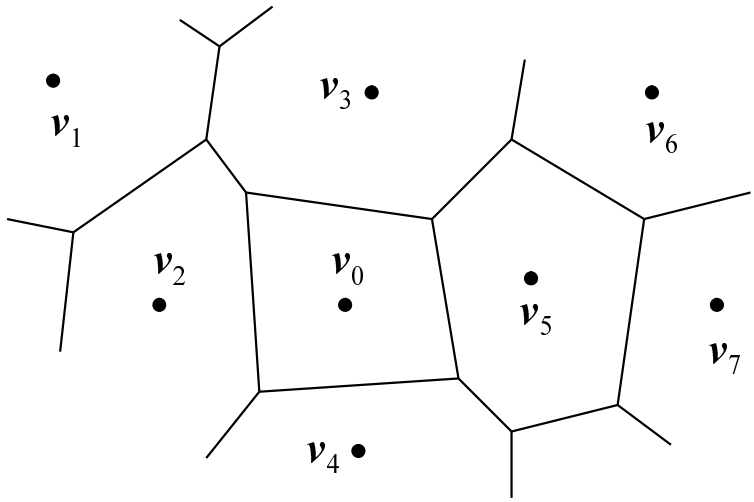


Figure 2.1: Eight vectors in $\mathbf{R}^n$. v_0 is the all-zero codeword. $v_2 \sim v_5$ are zero neighbors. v_1, v_6, and v_7 are not zero neighbors.

Definition 3 (Local weight distribution)**.** Let $L_w(C)$ be the number of zero neighbors with weight w in C. The local weight distribution of C is defined as the $(n+1)$-tuple $(L_0(C), L_1(C), \ldots, L_n(C))$.

Local weight distribution is called *local distance profile* in [1, 6, 13, 14, 15]. Knowledge of the local weight distribution of a code is valuable for the error performance analysis of the code. For example, the local weight distribution of a code gives a tighter upper bound on error probability for soft decision decoding over an AWGN channel than the usual union bound [6]. The details are represented in section 2.4.

2.2 Determination of local weight distribution

Useful properties to determine the local weight distribution of a code are shown. First, we define the *support set* of a codeword.

Definition 4 (Support set)**.** A set of positions of nonzero elements in $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$ is called the support set of $\boldsymbol{v}$, denoted by $\mathrm{Supp}(\boldsymbol{v})$. That is, $\mathrm{Supp}(\boldsymbol{v}) = \{i : v_i \neq 0\}$.

The following lemma is helpful to examine whether a given codeword is a zero neighbor or not.

Lemma 1. $\boldsymbol{v} \in C$ is a zero neighbor if and only if there does not exist $\boldsymbol{v}' \in C \setminus \{\mathbf{0}\}$ such that $\mathrm{Supp}(\boldsymbol{v}') \subsetneq \mathrm{Supp}(\boldsymbol{v})$.

For example, suppose that a binary linear code C consists of $\{0000, 0110, 1001, 1111\}$. 1111 is not a zero neighbor because $\mathrm{Supp}(0110) = \{2, 3\}$ is a subset of $\mathrm{Supp}(1111) =$

$\{1, 2, 3, 4\}$. 0110 is a zero neighbor because there exists no codeword $\boldsymbol{v} \in C \setminus \{\mathbf{0}\}$ such that $\mathrm{Supp}(\boldsymbol{v}) \subsetneq \mathrm{Supp}(0110)$. 1001 is also a zero neighbor. Then, the local weight distribution of C is $(0, 0, 2, 0, 0)$.

The complexity of checking whether a given codeword is a zero neighbor or not is $O(n^2k)$ [1] when a generator matrix is used and $O(n^2(n-k))$ when a parity check matrix is used. Specific algorithms and the details are presented in section 3.3.2. Since there are 2^k codewords in C, the whole complexity for computing the local weight distribution is $O(n^2k \cdot 2^k)$. Thus, computing the local weight distribution is very time-consuming, especially for codes with large dimensions.

On the local weight distribution, we have the following lemma [2, 3].

Lemma 2. Let $A_w(C)$ be the number of codewords with weight w in C and d be the minimum distance of C.

$$L_w(C) = \begin{cases} A_w(C), & w < 2d, \\ 0, & w > n - k + 1. \end{cases} \tag{2.3}$$

The $(n+1)$-tuple $(A_0(C), A_1(C), \ldots, A_n(C))$ is called the *global weight distribution*, or simply *weight distribution*, of C. From this lemma, if the weight distribution is known, only $L_w(C)$ with $2d \leq w \leq n - k + 1$ need to be computed to obtain the local weight distribution. Generally, the complexity for computing the local weight distribution is larger than that for computing the weight distribution. Therefore, Lemma 2 is useful for obtaining local weight distributions. Moreover, when all the weights w in a code is confined in $w < 2d$ and $w > n - k + 1$, the local weight distribution can be obtained from the weight distribution straightforwardly. For example, the local weight distribution of the (n, k) primitive BCH code of length 63 for $k \leq 18$, of length 127 for $k \leq 29$, and of length 255 for $k \leq 45$ can be obtained from their weight distributions.

2.3 Known results

In this section, known results for obtaining the local weight distributions are presented.

A. Hamming codes

Let C be a binary Hamming code of length $n = 2^m - 1$ with dimension $k = n - m$. The local weight distribution of C is given in the following theorem [3].

Theorem 1.

$$L_w(C) = \begin{cases} \frac{1}{w!} \prod_{i=0}^{w-2}(2^m - 2^i), & 3 \le w \le m+1 \\ 0, & 0 \le w < 3, \quad m+1 < w \le 2^m - 1. \end{cases} \tag{2.4}$$

A similar argument yields the following result for extended Hamming codes [3].

Theorem 2. Let C be a binary $(2^m, 2^m - m)$ extended Hamming code.

$$L_w(C) = \begin{cases} \frac{1}{w!}\, 2^m \prod_{i=0}^{w-3}(2^m - 2^i), & 4 \le w \le m+2 \\ 0, & 0 \le w < 4, \quad m+2 < w \le 2^m. \end{cases} \tag{2.5}$$

B. Reed-Muller codes

Let $\mathrm{RM}(r, m)$ be a binary r^{th} order Reed-Muller code of length $n = 2^m$. First, we show the formula for computing the local weight distribution of the second-order Reed-Muller code. For the second-order Reed-Muller code, the dimension k is $1 + m + \binom{m}{2}$ and the minimum distance d is 2^{m-2}. The local weight distribution of $\mathrm{RM}(2, m)$ is given in the following theorem [3].

Theorem 3.

$$L_w(\mathrm{RM}(2,m)) = \begin{cases} 0, & w = 2^{m-1} + 2^{m-1-h} \ (h = 0, 1, 2), \\ & w > 2^m - k + 1, \\ A_w - 2^{m+1} + 2 - A_d(2^{m-1} - 2), & w = 2^{m-1}, \\ A_w, & \text{otherwise} \end{cases} \tag{2.6}$$

where

$$A_{2^{m-1} \pm 2^{m-1-i}} = 2^{i(i+1)} \cdot \frac{(2^m - 1)(2^{m-1} - 1) \cdots (2^{m-2i+1} - 1)}{(2^{2i} - 1)(2^{2i-2} - 1) \cdots (2^2 - 1)}, \quad \text{for } 1 \le i \le [\tfrac{1}{2}m], \tag{2.7}$$

$$A_{2^{m-1}} = 2^{1+m+\binom{m}{2}} - \sum_{j \ne 2^{m-1}} A_j. \tag{2.8}$$

Borissov et al. derived partial results for the local weight distributions of r^{th} order Reed-Muller codes [5]. They are given as Theorems 4 and 5.

Theorem 4. The number of non-zero neighbors with weight $2d = 2^{m-r+1}$ in $\mathrm{RM}(r, m)$ is

$$N_{\mathrm{r,m}} = A_{r,m} + B_{r,m} + P_{r,m} - S_{r,m}, \tag{2.9}$$

where

$$A_{r,m} = 2^{r-1} \begin{bmatrix} m \\ m-r+1 \end{bmatrix}, \tag{2.10}$$

$$B_{r,m} = \frac{2^{r+1}-4}{4} \begin{bmatrix} m \\ m-r+1 \end{bmatrix} \binom{2^{r+1}}{3}, \tag{2.11}$$

$$P_{r,m} = 2^{r-1} \begin{bmatrix} m \\ m-r \end{bmatrix} \left(2^r \begin{bmatrix} m \\ m-r \end{bmatrix} - Q_{r,m} \right), \tag{2.12}$$

$$Q_{r,m} = \sum_{k=\max\{0,m-2r\}}^{m-r} 2^{(m-r-k)(m-r-k+1)} \begin{bmatrix} m-r \\ k \end{bmatrix} \begin{bmatrix} r \\ m-r-k \end{bmatrix}, \tag{2.13}$$

$$\begin{aligned} S_{r,m} &= 2^{r-1}(2^{m-r+1}-1) \begin{bmatrix} m \\ m-r+1 \end{bmatrix} \\ &\quad + \frac{1}{8} \cdot 2^{r+1}(2^{r+1}-1)(2^{r+1}-2)(2^{r+1}-4) \begin{bmatrix} m \\ m-r+1 \end{bmatrix}, \end{aligned} \tag{2.14}$$

$\begin{bmatrix} m \\ i \end{bmatrix}$ is the *2-ary gaussian coefficient*, defined by:

$$\begin{bmatrix} m \\ i \end{bmatrix} = \prod_{j=0}^{i-1} \frac{2^m - 2^i}{2^i - 2^j}, \quad \begin{bmatrix} m \\ 0 \end{bmatrix} = 1, \tag{2.15}$$

for $i = 1, 2, 3, \ldots, m$.

Theorem 5. For RM(r, m) code of order $r \geq 3$, any codeword $\boldsymbol{v} \in \mathrm{RM}(r, m)$ of $\mathrm{wt}(\boldsymbol{v}) > 2^m - 2^{m-r+1}$ is not a zero neighbor.

C. Random codes

Suppose that C is a random linear code whose parity check matrix has independent equiprobable entries. Let $L_w(C)$ be the number of zero neighbors with weight w in C and $\mathbf{E}L_w(C)$ be its average number of the ensemble over random linear codes. We have the following theorem [3].

Theorem 6.

$$\mathbf{E}L_w(C) = \begin{cases} \binom{n}{w} \frac{1}{2^{n-k}} \prod_{i=0}^{w-2} (1 - 2^{-(n-k-i)}), & w \leq n-k+1 \\ 0, & \text{otherwise.} \end{cases} \tag{2.16}$$

D. Long codes

Agrell introduced the *neighbor ratio* and gave an asymptotic analysis for long codes [2]. The *neighbor ratio* is defined as

$$\Gamma(C) = \frac{\sum_{i=0}^{n} L_w(C)}{\sum_{i=0}^{n} A_w(C)}. \tag{2.17}$$

The neighbor ratio satisfies $0 \leq \Gamma(C) \leq 1$. Suppose that a constant rate $0 < R(= k/n) \leq 1$ is given. Consider an infinite sequence of codes C_i, $i = 1, 2, \ldots$, whose parameters satisfy that the length and the rate are tend to infinity and R, respectively, as i tend to infinity. Then the *asymptotic neighbor ratio* is defined as

$$\Gamma_\infty(R) = \lim_{i \to \infty} \Gamma(C_i). \tag{2.18}$$

The following theorems are derived [2].

Theorem 7. The asymptotic neighbor ratio satisfies

$$\Gamma_\infty(R) = 0, \qquad \text{if} \quad R > 1/2 \tag{2.19}$$

for any sequence of codes such that $\Gamma_\infty(R)$ exists.

Theorem 8. Let $\mathcal{L}(n, R)$ be the set of all binary linear codes for which the length is n and the rate is R. For any $R < 1/2$ and any $\epsilon > 0$, the proportion of codes $C \in \mathcal{L}(n, R)$ that satisfy $\Gamma(C) > 1 - \epsilon$ tends to 1 as $n \to \infty$.

Above theorems show that the neighbor ratio is close to 0 for all sufficient long codes with $R > 1/2$, and close to 1 for almost all code with $R < 1/2$.

E. Cyclic codes

As described in Section 2.2, the complexity for computing the local weight distribution is very large. Agrell noted in [1] that the automorphism group of codes helps reduce the complexity. Using the automorphism group of cyclic codes, i.e. cyclic permutations, Mohri et al. obtained the local weight distributions of the $(63, k)$ primitive BCH codes for $k \leq 45$ [14, 15]. The algorithm uses the following invariance property for cyclic permutations.

Theorem 9. Let C be a binary cyclic code. Any cyclic permuted codeword of $\boldsymbol{v}$ is a zero neighbor if and only if a codeword $\boldsymbol{v} \in C$ is a zero neighbor.

Corollary 1. Let C be a binary cyclic code, and $\sigma^i \boldsymbol{v}$ be an i times cyclic-permuted codeword of $\boldsymbol{v} \in C$. Consider a set $S = \{\boldsymbol{v}, \sigma\boldsymbol{v}, \sigma^2\boldsymbol{v}, \ldots, \sigma^{p(\sigma,\boldsymbol{v})-1}\boldsymbol{v}\}$, where $p(\sigma, \boldsymbol{v})$ is the

period of σ, which is the minimum i such that $\sigma^i \boldsymbol{v} = \boldsymbol{v}$. Then, (1) if $\boldsymbol{v}$ is a zero neighbor, all codewords in the set S are zero neighbors; and (2) otherwise, all codewords in S are not zero neighbors.

In this algorithm, the representative codeword of cyclic permutations (a representative codeword of S in Corollary 1) and the number of the equivalent codewords (the size of S) are generated efficiently. The complexity is about $1/n$ as that of the brute force method. For the $(63, k)$ primitive BCH codes with $k = 51, 57$, Mohri et al. also obtained the local weight distributions using another algorithm [13, 14]. The latter algorithm is based on the same idea as the former, i.e. using the invariance property, and generates the representative codewords with large weight more efficiently than the former although the representative codewords may not be generated once.

The following corollary implies that the algorithms in [13, 14, 15] could be applied to extended cyclic codes straightforwardly.

Corollary 2. Let C and $C_{\rm ex}$ be a binary cyclic code and its extended code, respectively. For $\boldsymbol{v} \in C$, let $\boldsymbol{v}^{(\rm ex)}$ be the corresponding codeword in $C_{\rm ex}$. Then, for any cyclic permuted codeword $\sigma^i \boldsymbol{v}$ of $\boldsymbol{v}$, $(\sigma^i \boldsymbol{v})^{(\rm ex)}$ is a zero neighbor in $C_{\rm ex}$ if and only if $\boldsymbol{v}^{(\rm ex)}$ is a zero neighbor in $C_{\rm ex}$.

Proof. (If part) Suppose that $(\sigma^i \boldsymbol{v})^{(\rm ex)}$ is not a zero neighbor in $C_{\rm ex}$. There exists $\boldsymbol{u} \in C$ such that $\mathrm{Supp}((\sigma^i \boldsymbol{u})^{(\rm ex)}) \subsetneq \mathrm{Supp}((\sigma^i \boldsymbol{v})^{(\rm ex)})$. Then, $\mathrm{Supp}(\boldsymbol{u}^{(\rm ex)}) \subsetneq \mathrm{Supp}(\boldsymbol{v}^{(\rm ex)})$, and this contradicts the fact that $\boldsymbol{v}^{(\rm ex)}$ is a zero neighbor in $C_{\rm ex}$. (Only if part) Suppose that $\boldsymbol{v}^{(\rm ex)}$ is not a zero neighbor in $C_{\rm ex}$. There exists $\boldsymbol{u} \in C$ such that $\mathrm{Supp}(\boldsymbol{u}^{(\rm ex)}) \subsetneq \mathrm{Supp}(\boldsymbol{v}^{(\rm ex)})$. Then, $\mathrm{Supp}((\sigma^i \boldsymbol{u})^{(\rm ex)}) \subsetneq \mathrm{Supp}((\sigma^i \boldsymbol{v})^{(\rm ex)})$, and this contradicts the fact that $(\sigma^i \boldsymbol{v})^{(\rm ex)}$ is a zero neighbor in $C_{\rm ex}$. □

2.4 Applications

An exact expression for the error probability P_e of soft decision decoding can not be obtained normally. We use the *union bound* for an approximation, which is

$$\begin{aligned} P_e &\leq \sum_{\boldsymbol{v} \in C \setminus \{\boldsymbol{0}\}} Q\left(\sqrt{\mathrm{wt}(\boldsymbol{v}) \frac{2E_b}{N_0}}\right) \\ &= \sum_{i=1}^{n} A_i(C)\, Q\left(\sqrt{i \frac{2E_b}{N_0}}\right) \end{aligned} \tag{2.20}$$

assuming equiprobable codewords and biorthogonal modulation (say, BPSK or QPSK) with bit energy E_b. The variance of the discrete-time Gaussian noise is $N_0/2$, $\mathrm{wt}(\boldsymbol{v})$ is the Hamming weight of $\boldsymbol{v}$, and $Q(x)$ is the integral $\int_x^\infty (2\pi)^{-1/2} \exp(-z^2/2) dz$.

Using the local weight distribution, (2.20) yields another bound,

$$\begin{aligned} P_e &\leq \sum_{\boldsymbol{v} \in N_0(C)} Q\left(\sqrt{\mathrm{wt}(\boldsymbol{v})\frac{2E_b}{N_0}}\right) \\ &= \sum_{i=1}^{n} L_i(C)\, Q\left(\sqrt{i\frac{2E_b}{N_0}}\right), \end{aligned} \tag{2.21}$$

where $N_0(C)$ is the set of zero neighbors in C. This bound is tighter than the usual union bound [1, 2, 6]. Agrell pointed out that [1] other bounds, related to the union bound, such as Berlekamp's tangential union bound [4], may be improved in a similar fashion.

Zero neighbors in a linear code have a link to secret-sharing schemes using error-correcting codes. Massey showed that the set of zero neighbors in the dual code completely specifies the access structure of the secret-sharing scheme [12].

3 A Computational Method to Determine Local Weight Distribution Using the Automorphism Group

In this chapter, a method for computing the local weight distribution using the automorphism group of the code is presented. In [14, 15], the complexity for computing the local weight distribution is reduced by using an invariance property for cyclic permutations. This invariance property can be generalized to that for any group of permutations. Using the invariance property for the larger group of permutations, we may reduce the number of representative codewords. However, it is not easy to obtain the representative codewords and the number of the equivalent codewords.

In order to use the generalized invariance property, the invariance property is applied to the set of cosets of a subcode rather than the set of codewords. This reduce the complexity of finding the representatives, which is much smaller than that for checking whether every representative is a zero neighbor or not. This idea is used in [7] for computing the (global) weight distribution of extended binary primitive BCH codes.

3.1 Invariance property

For a permutation π and a set of vectors D, the set of the permuted vectors $\pi[D]$ is defined as

$$\pi[D] = \{\pi \boldsymbol{v} : \boldsymbol{v} \in D\}. \tag{3.1}$$

The automorphism group of a code C is the set of all permutations by which C is permuted into C, and denoted by $\mathrm{Aut}(C)$, i.e.,

Definition 5 (Automorphism group of a code)**.**

$$\mathrm{Aut}(C) = \{\pi : \pi[C] = C\}. \tag{3.2}$$

An invariance property under the automorphism group of a code is given in the following theorem.

Theorem 10 (Invariance property)**.** For $\pi \in \mathrm{Aut}(C)$ and $\boldsymbol{v} \in C$, $\pi\boldsymbol{v}$ is a zero neighbor if and only if $\boldsymbol{v}$ is a zero neighbor.

Proof. Suppose that $\boldsymbol{v}$ is a zero neighbor and $\pi\boldsymbol{v}$ is not a zero neighbor. There exists a nonzero codeword $\boldsymbol{v}' \in C$ such that $\mathrm{Supp}(\pi\boldsymbol{v}) \supsetneqq \mathrm{Supp}(\boldsymbol{v}')$ from Lemma 1. Since $\mathrm{Aut}(C)$ is a group, there exists $\boldsymbol{v}'' \in C$ such that $\boldsymbol{v}' = \pi\boldsymbol{v}''$. Thus $\mathrm{Supp}(\pi\boldsymbol{v}) \supsetneqq \mathrm{Supp}(\pi\boldsymbol{v}'')$, and $\mathrm{Supp}(\boldsymbol{v}) \supsetneqq \mathrm{Supp}(\boldsymbol{v}'')$. This contradicts being the zero neighbor of $\boldsymbol{v}$, from Lemma 1. □

This theorem derives the following corollary:

Corollary 3. For $\boldsymbol{v} \in C$, consider a set $S = \{\pi\boldsymbol{v} : \forall\pi \in \mathrm{Aut}(C)\}$. Then, (1) if $\boldsymbol{v}$ is a zero neighbor, all codewords in S are zero neighbors; and (2) otherwise, all codewords in S are not zero neighbors.

It is not easy to devise a similar algorithm as that in [14, 15]. This is because, for almost all groups of permutations, no efficient way is known for generating the representative codewords and obtaining the number of the equivalent codewords. In order to use this generalized invariance property, we apply the invariance property to the set of cosets of a subcode rather than the set of codewords.

3.2 Local weight subdistribution for cosets of subcode

For a binary (n, k) linear code C and its linear subcode with dimension k', let C/C' denote the set of cosets of C' in C, that is, $C/C' = \{\boldsymbol{v} + C' : \boldsymbol{v} \in C \setminus C'\}$. Then,

$$|C/C'| = 2^{k-k'}, \quad \text{and} \quad C = \bigcup_{D \in C/C'} D. \tag{3.3}$$

Definition 6 (Local weight subdistribution for cosets)**.** The local weight subdistribution for a coset $D \in C/C'$ is the weight distribution of zero neighbors of C in D. The local weight subdistribution for D is $(|LS_0(D)|, |LS_1(D)|, \ldots, |LS_n(D)|)$, where

$$\begin{aligned} LS_w(D) \;=\; & \{\boldsymbol{v} \in D : \mathrm{Supp}(\boldsymbol{v}') \nsubseteq \mathrm{Supp}(\boldsymbol{v}) \;\text{ for any }\; \boldsymbol{v}' \in C \setminus \{\boldsymbol{0}, \boldsymbol{v}\}, \\ & \text{and the Hamming weight of } \boldsymbol{v} \text{ is } w\}, \end{aligned} \tag{3.4}$$

with $0 \le w \le n$.

Then, from (3.3), the local weight distribution of C is given as the sum of the local weight subdistributions for the cosets in C/C', that is,

$$L_w = \sum_{D \in C/C'} |LS_w(D)|. \tag{3.5}$$

The following theorem gives an invariance property under permutations for cosets.

Theorem 11 (Invariance property for cosets)**.** For $D_1, D_2 \in C/C'$, the local weight subdistribution for D_1 and that for D_2 are the same if there exists $\pi \in \mathrm{Aut}(C)$ such that $\pi[D_1] = D_2$.

Proof. For any codewords $\boldsymbol{v} \in D_1$, $\pi\boldsymbol{v} \in D_2$, from Theorem 10, $\pi\boldsymbol{v}$ is a zero neighbor if and only if $\boldsymbol{v}$ is a zero neighbor. Therefore the local weight subdistribution for D_1 and that for D_2 are the same. □

This theorem is a condition for cosets having the same local weight subdistribution. The following lemma gives the set of all permutations by which every coset in C/C' is permuted into one in C/C'.

Lemma 3. For a linear code C and its linear subcode C',

$$\{\pi : \pi[D] \in C/C' \text{ for any } D \in C/C'\} = \mathrm{Aut}(C) \cap \mathrm{Aut}(C'). \tag{3.6}$$

Proof. Let $\pi \in \mathrm{Aut}(C) \cap \mathrm{Aut}(C')$. For a coset $\boldsymbol{v}_1 + C' \in C/C'$, suppose that $\pi \boldsymbol{v}_1 \in \boldsymbol{v}_2 + C'$. For any codeword $\boldsymbol{v}_1 + \boldsymbol{u}_1 \in \boldsymbol{v}_1 + C'$,

$$\begin{aligned} \pi(\boldsymbol{v}_1 + \boldsymbol{u}_1) &= \pi\boldsymbol{v}_1 + \pi\boldsymbol{u}_1 \\ &= \boldsymbol{v}_2 + \boldsymbol{u}_2 + \pi\boldsymbol{u}_1, \quad \boldsymbol{u}_2 \in C', \\ &= \boldsymbol{v}_2 + (\boldsymbol{u}_2 + \pi\boldsymbol{u}_1) \ \in \ \boldsymbol{v}_2 + C'. \end{aligned} \tag{3.7}$$

Thus, $\pi[\boldsymbol{v}_1 + C'] = \boldsymbol{v}_2 + C' \in C/C'$. Then, $\{\pi : \pi[D] \in C/C' \text{ for any } D \in C/C'\} \supseteq \mathrm{Aut}(C) \cap \mathrm{Aut}(C')$.

Let $\pi \in \{\rho : \rho[D] \in C/C' \text{ for any } D \in C/C'\}$. For any codeword $\boldsymbol{v} \in C$, $\boldsymbol{v}$ must be in either coset in C/C', and then $\pi\boldsymbol{v} \in C$. Thus, $\pi \in \mathrm{Aut}(C)$. C' itself is one of cosets in C/C'. For any codeword $\boldsymbol{u} \in C'$, $\pi\boldsymbol{u} \in C'$ because $\pi[C'] = C'$. Thus, $\pi \in \mathrm{Aut}(C')$. Then, $\{\pi : \pi[D] \in C/C' \text{ for any } D \in C/C'\} \subseteq \mathrm{Aut}(C) \cap \mathrm{Aut}(C')$. □

$\mathrm{Aut}(C) \cap \mathrm{Aut}(C')$ (or even $\mathrm{Aut}(C)$) is generally not known. Only subgroups of $\mathrm{Aut}(C) \cap \mathrm{Aut}(C')$ are known. Therefore, we use a subgroup.

Definition 7. Let Π be a subset of $\mathrm{Aut}(C) \cap \mathrm{Aut}(C')$. For $D_1, D_2 \in C/C'$, we denote $D_1 \sim_\Pi D_2$ if and only if there exists $\pi \in \Pi$ such that $\pi[D_1] = D_2$.

Lemma 4. The relation "$\sim_\Pi$" is an equivalence relation on C/C' if Π forms a group.

Proof. Let $D_1, D_2, D_3 \in C/C'$.
(Reflexive: $D_1 \sim D_1$) From the definition of $\mathrm{Aut}(C)$, the identity permutation $\pi_0 \in \mathrm{Aut}(C)$. Then, $\pi_0[D_1] = D_1$, and $D_1 \sim D_1$.
(Symmetric: $D_1 \sim D_2 \to D_2 \sim D_1$) Suppose that $D_1 \sim D_2$ and $\pi[D_1] = D_2$ for $\pi \in \mathrm{Aut}(C)$. There exists $\rho \in \mathrm{Aut}(C)$ such that $\rho[\pi[D_1]] = D_1$ since $\mathrm{Aut}(C)$ forms a group. Then, $\rho[D_2] = D_1$, and $D_2 \sim D_1$.
(Transitive: $D_1 \sim D_2, D_2 \sim D_3 \to D_1 \sim D_3$) Suppose that $D_1 \sim D_2, D_2 \sim D_3$. There exists $\pi, \rho \in \mathrm{Aut}(C)$ such that $\pi[D_1] = D_2$, $\rho[D_2] = D_3$. Then, $D_3 = \rho[D_2] = \rho\pi[D_1]$. Since $\rho\pi \in \mathrm{Aut}(C)$, $D_1 \sim D_3$. □

When the set of cosets are partitioned into equivalence classes by the relation "$\sim_\Pi$", the local weight subdistributions for cosets which belong to the same equivalence class are the same.

We give a useful theorem for partitioning the set of cosets into equivalence classes by the relation "$\sim_\Pi$."

Theorem 12. Let Π be a subset of $\mathrm{Aut}(C) \cap \mathrm{Aut}(C')$. For $D_1, D_2 \in C/C'$ and $\pi \in \Pi$, we have $D_1 \sim_\Pi D_2$ if $\pi \boldsymbol{v}_1 \in D_2$ for any $\boldsymbol{v}_1 \in D_1$.

Proof. Let $\pi \boldsymbol{v}_1 = \boldsymbol{v}_2 \in D_2$. Any codeword in D_1 is represented by $\boldsymbol{v}_1 + \boldsymbol{v}$ ($\boldsymbol{v} \in C'$). Then,

$$\begin{aligned} \pi(\boldsymbol{v}_1 + \boldsymbol{v}) &= \pi \boldsymbol{v}_1 + \pi \boldsymbol{v} \\ &= \boldsymbol{v}_2 + \pi \boldsymbol{v}. \end{aligned} \tag{3.8}$$

Since $\pi \in \mathrm{Aut}(C')$, $\pi \boldsymbol{v}$ is in C'. Thus $\pi[D_1] = D_2$. □

From Theorem 12, in order to partition the set of cosets into equivalence classes, we only need to check whether the representative codeword of a coset is permuted into another coset. After partitioning into equivalence classes, the local weight subdistribution for only one coset in each equivalence class needs to be computed. Thereby the computational complexity is reduced.

3.3 An algorithm for computing the local weight distribution

On the basis of the method for partitioning the set of cosets described in the previous section, we can compute the local weight distribution as follows (refer to Figure 3.1):

1. Choose a subcode C' and a subgroup Π of permutations of $\mathrm{Aut}(C) \cap \mathrm{Aut}(C')$.

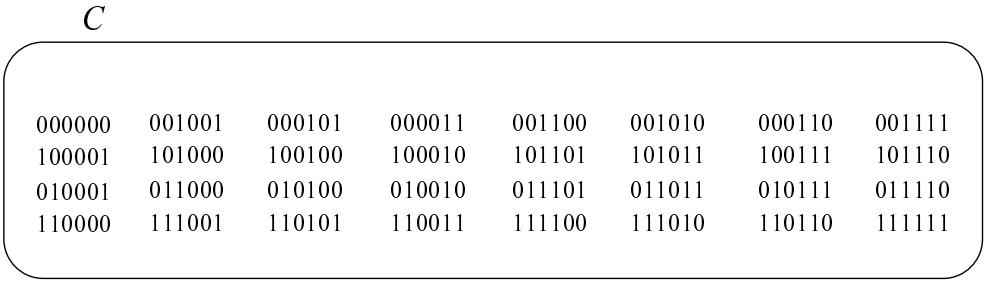
2. Partition C/C' into equivalence classes with permutations in Π, and obtain the number of codewords in each equivalence class.

3. Compute the local weight subdistributions for the representative cosets in each equivalence class.

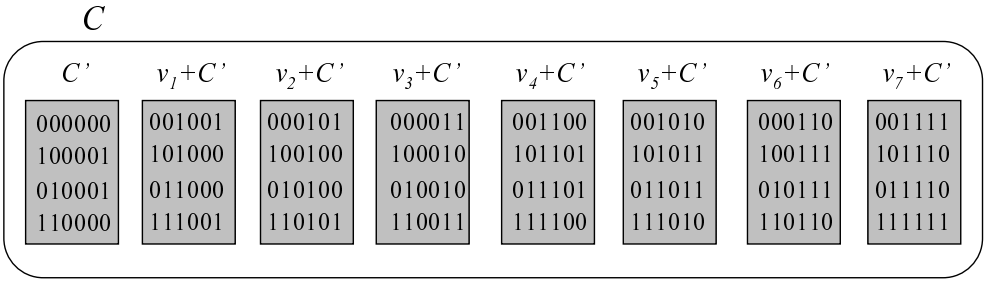
4. Sum up all the local weight subdistributions.

The algorithm is shown in Figure 3.2.

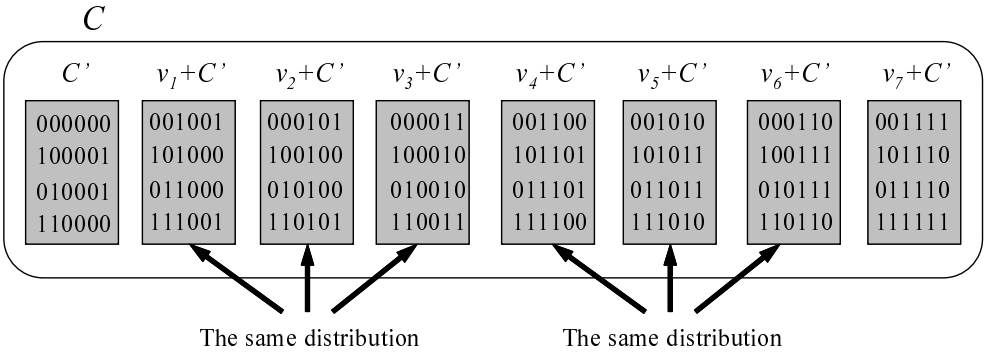
Let C be a binary linear code.



STEP1: Choose a subcode C'. Consider C the set of cosets of the subcode C'.



STEP2: Partition C/C' into equivalence classes.



STEP3: Compute the local weight subdistributions for the representative cosets in each equivalence class.

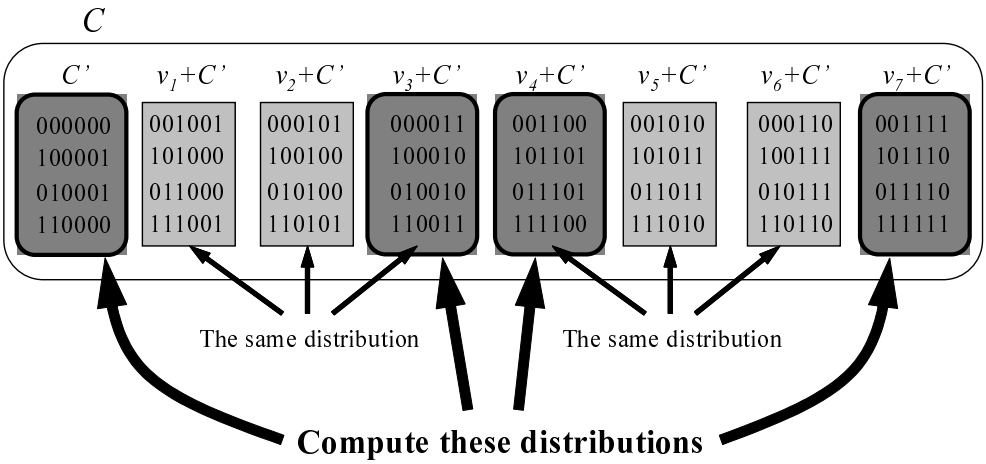


Figure 3.1: Diagram of the procedure of the proposed algorithm.

Input: G, a generator matrix of an (n, k, d) linear code C.
G', a generator matrix of a linear subcode C'.

Output: $L[i], (0 \leq i \leq n)$, the number of zero neighbors with weight i.

Algorithm:

For $i \leftarrow 0$ to n:
    $L[i] \leftarrow 0$.
Generate the cosets C/C'.
Partition the cosets into equivalent classes.
For every representative coset D:
    $class \leftarrow$ the number of the cosets equivalent to D.
    For every codeword $\boldsymbol{u}$ in D:
        $w \leftarrow$ the Hamming weight of $\boldsymbol{u}$.
        If $w < 2d$:
            $L[w] \leftarrow L[w] + class$.
        If $\boldsymbol{u}$ is turned out to be a zero neighbor:
            $L[w] \leftarrow L[w] + class$.

Figure 3.2: An algorithm for computing the local weight distribution using the automorphism group of a code.

3.3.1 Partitioning cosets into equivalence classes

The implementation of step (2) of the algorithm is based on Theorem 12 and Lemma 4. In order to partition cosets efficiently, we use the following one-to-one corresponding between the set of the cosets and $\{0,1\}^{k-k'}$. We choose a parity check matrix H' of C' with

$$H' = \begin{pmatrix} H_0 \\ H \end{pmatrix}, \tag{3.9}$$

where H is a parity check matrix of C, and H_0 is an $n \times (k-k')$ matrix. Then, the one-to-one corresponding is

$$\boldsymbol{v} + C' \leftrightarrow \boldsymbol{v}H_0^T,$$

where H_0^T means the transpose of H_0. $\boldsymbol{v}H_0^T$ is a part of the syndrome of $\boldsymbol{v}$.

An algorithm for partitioning cosets into equivalence classes is presented in Figure 3.3. After partitioning cosets, class$[i]$ contains the number of equivalent cosets to the i-th coset, and synd2index$[s]$ contains the index of the coset whose syndrome is s.

3.3.2 Examining zero neighborship

First, we define the following set with respect to a linear code C and a codeword $\boldsymbol{v} \in C$:

$$C(\boldsymbol{v}) = \{\boldsymbol{u} : \boldsymbol{u} \in C,\ \mathrm{Supp}(\boldsymbol{u}) \subseteq \mathrm{Supp}(\boldsymbol{v})\}. \tag{3.10}$$

Lemma 5. For $\boldsymbol{v} \in C$, $C(\boldsymbol{v})$ is a linear subcode of C.

Proof. Let $\boldsymbol{v}_1, \boldsymbol{v}_2 \in C(\boldsymbol{v})$. Since $\mathrm{Supp}(\boldsymbol{v_1}), \mathrm{Supp}(\boldsymbol{v_2}) \subseteq \mathrm{Supp}(\boldsymbol{v})$, $\mathrm{Supp}(\boldsymbol{v}_1 + \boldsymbol{v}_2) \subseteq \mathrm{Supp}(\boldsymbol{v}_1) + \mathrm{Supp}(\boldsymbol{v}_2) \subseteq \mathrm{Supp}(\boldsymbol{v})$. Thus, $\boldsymbol{v}_1 + \boldsymbol{v}_2 \in C(\boldsymbol{v})$, and then $C(\boldsymbol{v})$ is a linear subcode of C. □

The following theorem is useful to examine whether a given codeword is a zero neighbor or not.

Theorem 13. For nonzero codeword $\boldsymbol{v} \in C$, the dimension of $C(\boldsymbol{v})$ is one or more. The dimension of $C(\boldsymbol{v})$ is one if and only if $\boldsymbol{v}$ is a zero neighbor in C.

Proof. Since $\boldsymbol{v} \in C(\boldsymbol{v})$, the dimension of $C(\boldsymbol{v})$ is one or more. For any $\boldsymbol{u} \in C(\boldsymbol{v})$, $\mathrm{Supp}(\boldsymbol{v}) \supseteq \mathrm{Supp}(\boldsymbol{u})$ from the definition. From Lemma 1, $\boldsymbol{v}$ is a zero neighbor if and only if $C(\boldsymbol{v})$ contains only $\mathbf{0}$ and $\boldsymbol{v}$. That is, $\boldsymbol{v}$ is a zero neighbor in C if and only if the dimension of $C(\boldsymbol{v})$ is one. □

To obtain the dimension of $C(\boldsymbol{v})$, we can use two methods: methods G and H.

Input: G, a generator matrix of C.

H, a parity check matrix of C.

G', a generator matrix of C'.

H', a parity check matrix of C'.

Π, a subgroup of $\mathrm{Aut}(C) \cap \mathrm{Aut}(C')$.

Output: class$[i]$ $(1 \leq i \leq 2^{k-k'})$.

Algorithm:

Generate the coset leaders, $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{2^{k-k'}}$ from G and G'.

Generate H_0 from H and H' to calculate the syndrome of codewords.

For $i \leftarrow 1$ to $2^{k-k'}$:

    class$[i] \leftarrow 0$.

    synd2index$[\boldsymbol{v}_i H_0^T] \leftarrow i$.

For $i \leftarrow 1$ to $2^{k-k'}$:

    If class$[i] = 0$:

        class$[i] \leftarrow 1$.

        For every $\pi \in \Pi$:

            If class[synd2index$[\boldsymbol{\pi v_i} H_0^T]$] $= 0$:

                class[synd2index$[\pi \boldsymbol{v}_i H_0^T]$] $= -1$.

                class$[i] \leftarrow$ class$[i] + 1$.

Figure 3.3: An algorithm for partitioning cosets into equivalence classes

Method G

Method G uses a generator matrix of C, denoted by G. The G method derives a generator matrix of $C(\boldsymbol{v})$ from G and checks the dimension of $C(\boldsymbol{v})$. The algorithm of the G method is shown in Figure 3.4. This algorithm is equivalent to the following G Rule presented in [2].

G Rule: Let $G_0(C, \boldsymbol{v})$ denote the matrix formed by the columns of a generator matrix G of C corresponding to positions where a given codeword $\boldsymbol{v} \in C$ has zeros. Then $\boldsymbol{v}$ is a zero neighbor if and only if rank $G_0(C, \boldsymbol{v}) = k - 1$ where k is the dimension of C.

Method H

Method H uses a parity check matrix of C, denoted by H. The H method derives a parity check matrix of $C(\boldsymbol{v})$ from H and checks the dimension of $C(\boldsymbol{v})$. The algorithm of the H method is shown in Figure 3.5. This algorithm is equivalent to the following H Rule presented in [2].

H Rule: Let $H_1(C, \boldsymbol{v})$ denote the matrix formed by the columns of a parity check matrix H of C corresponding to positions where a given codeword $\boldsymbol{v} \in C$ has ones. Then $\boldsymbol{v}$ is a zero neighbor if and only if rank $H_1(C, \boldsymbol{v}) = \mathrm{wt}(\boldsymbol{v}) - 1$ where $\mathrm{wt}(\boldsymbol{v})$ is the Hamming weight of $\boldsymbol{v}$.

3.4 Complexity

Here, we analyze computational complexity of the algorithm. Let C be an (n, k) linear code and C' be an (n, k') linear subcode of C. We compute the local weight distribution of C using C'.

Time complexity

We can use Method G and Method H to check whether a given codeword is a zero neighbor or not. The time complexity of checking one codeword is $O(n^2 k)$ and $O(n^2(n-k))$ for Method G and Method H, respectively. Thus, we should use G Method for codes with rate less than $1/2$. Since the number of codewords in each coset is $2^{k'}$, the total number of codewords to be checked by the procedure is $e2^k$, where e is the number of the equivalence classes. Hence, the time complexity of Step (3) of the proposed algorithm using Method G is $O(n^2 k \cdot e2^{k'})$. The time complexity of Step (2), partitioning into equivalence classes, is $O(n(k - k')2^{k-k'}|\Pi|)$.

Therefore, The time complexity of the entire algorithm is $O(n^2 k \cdot e2^{k'} + n(k - k')2^{k-k'}|\Pi|)$. When k' is chosen as $k' > k/2$, then $2^{k'} > 2^{k-k'}$, and the complexity of partitioning into equivalence classes is much smaller than of computing the local weight

Input: $\boldsymbol{v} \in C$, a codeword to examine.

$\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_k$, the rows of a generator matrix of C.

Output: 1, if $\boldsymbol{v}$ is a zero neighbor,

0, otherwise.

Algorithm:

$i \leftarrow 0$.
For every p in $\{1, 2, \ldots, n\} \setminus \mathrm{Supp}(\boldsymbol{v})$:
  $first \leftarrow 1$.
  For $j \leftarrow i + 1$ to n:
    If the p-th element in $\boldsymbol{g}_j$ is 0:
      If $first = 1$ then:
        $pivot \leftarrow j$.
        $first \leftarrow 0$.
      else:
        $\boldsymbol{g}_j \leftarrow \boldsymbol{g}_j \oplus \boldsymbol{g}_{pivot}$.
  If $first = 0$ then:
    Swap$(\boldsymbol{g}_i, \boldsymbol{g}_{pivot})$.
    $i \leftarrow i + 1$.
    If $i = k - 1$ then:
      return 1.
return 0.

Figure 3.4: Method G: An algorithm for examining the zero neighborship of $\boldsymbol{v}$ using a generator matrix of C.

Input: $\boldsymbol{v} \in C$, a codeword to examine.
$\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_{n-k}$, the rows of a parity check matrix of H.

Output: 1, if $\boldsymbol{v}$ is a zero neighbor,
0, otherwise.

Algorithm:

$i \leftarrow 0$.
For every p in Supp($\boldsymbol{v}$):
    $first \leftarrow 1$.
    For $j \leftarrow i+1$ to $n-k$:
        If the p-th element in $\boldsymbol{h}_j$ is 0:
            If $first = 1$ then:
                $pivot \leftarrow j$.
                $first \leftarrow 0$.
            else:
                $\boldsymbol{h}_j \leftarrow \boldsymbol{h}_j \oplus \boldsymbol{h}_{pivot}$.
    If $first = 0$ then:
        Swap($\boldsymbol{h}_i, \boldsymbol{h}_{pivot}$).
        $i \leftarrow i+1$.
If $i = w-1$ then:
    return 1.
else:
    return 0.

Figure 3.5: Method H: An algorithm for examining the zero neighborship of $\boldsymbol{v}$ using a parity check matrix of C.

subdistributions for cosets.

Space complexity

The space complexity of checking a zero neighborship is very small, because we need space to storage only a generator matrix or a parity check matrix of C. It is $O(n^2)$. On the other hand, the space complexity of partitioning cosets into equivalence classes is much large. We need space to storage the entries proportional to $2^{k-k'}$, as seen in Figure 3.3, and it is $O((k-k')2^{k-k'})$. The space complexity of the entire algorithm is $O((k-k')2^{k-k'})$.

3.5 Selection of subcodes

We should choose the subcode for which the number of permutations in Π is large.

If there are several such subcode with the same Π, then the subcode with the smaller dimension should be chosen to minimize the number of codewords that need to be checked, as long as the complexity of partitioning into equivalence classes is relatively small.

3.6 Application of the computational method

The proposed algorithm can be applied to codes that are closed under a group of permutations and whose subcodes are also closed under the same group of permutations. We applied the algorithm to extended primitive BCH codes and Reed-Muller codes. Extended primitive BCH codes are closed under the affine group and Reed-Muller codes are closed under the general affine group [11].

3.6.1 Extended primitive BCH codes

Let C be an extended code of length $n = 2^m$. Suppose that the parity bit is added in the n-th element. For a codeword $\boldsymbol{v} = (v_0, v_1, \ldots, v_n) \in C$, we map the elements of the codeword $v_0, v_1, v_2, \ldots, v_{n-1}, v_n$ into elements of GF(2^m), $\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{n-1}, 0$. The *affine group* is defined as follows [11]:

Definition 8 (The affine group)**.** Let $P_{\beta,\gamma}$ permute the elements of GF(2^m) by

$$P_{\beta,\gamma} : \alpha^i \to \beta\alpha^i + \gamma \tag{3.11}$$

where $\beta, \gamma \in \mathrm{GF}(2^m), \beta \neq 0$. The set of all $P_{\beta,\gamma}$ forms a group called the *affine group* on GF(2^m).

The affine group has order $2^m(2^m - 1)$ and doubly transitive [11].

Extended primitive BCH codes are closed under the affine group. We applied the algorithm to extended primitive BCH codes of length 128. As noted in section 3.5, the

number of codewords that need to be checked, i.e. the complexity for computing the local weight distribution after partitioning cosets, varies according to the selection of a subcode C'. We computed the number of codewords that need to be checked for many cases. The results are presented in Table 3.1.

3.6.2 Reed-Muller codes

Let $RM(r, m)$ be a binary r^{th} order Reed-Muller code of length $n = 2^m$. Codewords in $RM(r, m)$ have one-to-one corresponding to Boolean functions, that is

$$\boldsymbol{f} \leftrightarrow f(X_1, X_2, \ldots, X_m) \tag{3.12}$$

where $\boldsymbol{f} \in RM(r, m)$ and $f(X_1, X_2, \ldots, X_m)$ is a Boolean function which is a polynomial of degree at most r.

Definition 9 (The general affine group)**.** Let $T_{A,b}$ transform $(X_1, X_2, \ldots, X_m)$ by

$$T_{A,b} : (X_1, X_2, \ldots, X_m) \rightarrow A(X_1, X_2, \ldots, X_m) + b \tag{3.13}$$

where $A(a_{ij})$ is an invertible $m \times m$ binary matrix and b is a binary m-tuple. The set of all $T_{A,b}$ forms a group called the *general affine group*. $T_{A,b}$ permutes Boolean functions as

$$T_{A,b} : f(X_1, X_2, \ldots, X_m) \rightarrow f(\Sigma a_{1j} X j + b_1, \Sigma a_{2j} X j + b_2, \ldots, \Sigma a_{mj} X j + b_m). \tag{3.14}$$

The general affine group has order $2^m(2^m - 1)(2^m - 2)(2^m - 2^2) \cdots (2^m - 2^{m-1})$ and triply transitive [11].

Definition 10 (The general linear group)**.** The subgroup of the general affine group consisting of all transformations

$$T_A : (X_1, X_2, \ldots, X_m) \rightarrow A(X_1, X_2, \ldots, X_m) \tag{3.15}$$

is the *general linear group*.

The general linear group has order $(2^m - 1)(2^m - 2)(2^m - 2^2) \cdots (2^m - 2^{m-1})$ and doubly transitive [11].

Reed-Muller codes are closed under the general affine group. We also apply the proposed algorithm to the third-order Reed-Muller code of length 128. We use the second-order Reed-Muller code as the subcode. The representative codewords of cosets are presented in [9]. A method to obtain the number of equivalent cosets to the representative cosets are presented in [17]. Thus, the process of obtaining the representative cosets and the number of equivalent cosets are different from that for extended primitive BCH codes. Note that the computing time for this process is vanishingly small. The number of equivalent cosets is shown in Table 3.2.

k	k'	N_{equ}	N_{check}
1	1	0	0.00×10^{0}
8	8	1	2.56×10^{2}
	1	2	2.00×10^{0}
15	15	1	3.28×10^{4}
	8	2	5.12×10^{2}
	1	4	8.00×10^{0}
22	22	1	4.19×10^{6}
	15	2	6.55×10^{4}
	8	130	3.33×10^{4}
	1	386	7.72×10^{2}
29	29	1	5.37×10^{8}
	22	2	8.39×10^{6}
	15	4	1.31×10^{5}
	8	258	6.60×10^{4}
36	36	1	6.87×10^{10}
	29	2	1.07×10^{9}
	22	130	5.45×10^{8}
	15	386	1.26×10^{7}
43	43	1	8.80×10^{12}
	36	2	1.37×10^{11}
	29	3	1.61×10^{9}
	22	258	1.08×10^{9}
50	50	1	1.13×10^{15}
	43	2	1.76×10^{13}
	36	130	8.93×10^{12}
	29	258	1.39×10^{11}
	22	32898	1.38×10^{11}
57	57	1	1.44×10^{17}
	50	2	2.25×10^{15}
	43	130	1.14×10^{15}
	36	16514	1.13×10^{15}
	29	32898	1.77×10^{13}
64	64	1	1.84×10^{19}
	57	2	2.88×10^{17}
	50	3	3.38×10^{15}
	43	258	2.27×10^{15}
	36	32898	2.26×10^{15}
71	71	1	2.36×10^{21}
	64	2	3.69×10^{19}
	57	130	1.87×10^{19}

k	k'	N_{equ}	N_{check}
78	78	1	3.02×10^{23}
	71	2	4.72×10^{21}
	64	130	2.40×10^{21}
	57	16514	2.38×10^{21}
	50	32898	3.70×10^{19}
85	85	1	3.87×10^{25}
	78	2	6.04×10^{23}
	71	3	7.08×10^{21}
	64	258	4.76×10^{21}
	57	32898	4.74×10^{21}
92	92	1	4.95×10^{27}
	85	2	7.74×10^{25}
	78	130	3.93×10^{25}
	71	258	6.09×10^{23}
	64	32898	6.07×10^{23}
99	99	1	6.34×10^{29}
	92	2	9.90×10^{27}
	85	3	1.16×10^{26}
	78	258	7.80×10^{25}
	71	16642	3.93×10^{25}
106	106	1	8.11×10^{31}
	99	2	1.27×10^{30}
	92	130	6.44×10^{29}
	85	258	9.98×10^{27}
	78	32898	9.94×10^{27}
113	113	1	1.04×10^{34}
	106	2	1.62×10^{32}
	99	3	1.90×10^{30}
	92	258	1.28×10^{30}
	85	16642	6.44×10^{29}
120	120	1	1.33×10^{36}
	113	2	2.08×10^{34}
	106	130	1.05×10^{34}
	99	258	1.64×10^{32}
	92	32898	1.63×10^{32}

Table 3.2: The number of equivalent cosets ($\nu_i(3,7)$ in [17]).

i	$\nu_i(3,7)$
1	1
2	11,811
3	2,314,956
4	45,354,240
5	59,527,440
6	21,165,312
7	1,763,776
8	2,222,357,760
9	238,109,760
10	17,778,862,080
11	444,471,552
12	13,545,799,680

4 Improvements of the Computational Method

In this chapter, some improvements of the proposed algorithm for computing the local weight distribution are shown.

4.1 Trellis structure

We consider reducing the complexity of checking whether a codeword is a zero neighbor or not. For $\boldsymbol{v} \in C$, let $C(\boldsymbol{v}) = \{\boldsymbol{u} \,|\, \boldsymbol{u} \in C,\ \mathrm{Supp}(\boldsymbol{u}) \subseteq \mathrm{Supp}(\boldsymbol{v})\}$. Checking whether a codeword $\boldsymbol{v}$ is a zero neighbor or not is examining whether the dimension of $C(\boldsymbol{v})$, denoted by $\dim(C(\boldsymbol{v}))$, is one or not. For $\boldsymbol{v} \in C$ and i with $1 \leq i \leq n$, let

$$\mathrm{S}(\boldsymbol{v}, i) = \{(u_1, u_2, \ldots, u_n) \in C \,|\, u_j = v_j \text{ with } 1 \leq j \leq i\} \tag{4.1}$$

and

$$\mathrm{C}(\boldsymbol{v}, i) = \{\boldsymbol{u} \,|\, \boldsymbol{u} \in \mathrm{S}(\boldsymbol{v}, i),\ \mathrm{Supp}(\boldsymbol{u}) \cap \{1, \ldots, i\} \subsetneq \mathrm{Supp}(\boldsymbol{v}) \cap \{1, \ldots, i\}\}. \tag{4.2}$$

A typical implementation to check whether $\boldsymbol{v}$ is a zero neighbor or not computes $\mathrm{C}(\boldsymbol{v}, i)$ for $i = 1, 2, \ldots, n$, where $\mathrm{C}(\boldsymbol{v}, n) = \mathrm{C}(\boldsymbol{v})$. For any $\boldsymbol{u}, \boldsymbol{u}' \in \mathrm{S}(\boldsymbol{v}, i)$, $\mathrm{C}(\boldsymbol{u}, i) = \mathrm{C}(\boldsymbol{u}', i)$. That is, if we generate $\mathrm{C}(\boldsymbol{u}, i)$ once, we does not need to generate $\mathrm{C}(\boldsymbol{u}, i)$ for each $\boldsymbol{u} \in \mathrm{S}(\boldsymbol{v}, i)$ later. This can reduce the computational complexity. We should choose i properly, say $n/2$ or

$n/4$, in order to make $\mathrm{S}(\boldsymbol{v}, i)$ large and $\mathrm{C}(\boldsymbol{u}, i)(\boldsymbol{u} \in \mathrm{S}(\boldsymbol{v}, i))$ small. To obtain $\mathrm{S}(\boldsymbol{v}, i)$, we use the trellis structure of cosets. This method does not make the space complexity much larger, since a codeword in $\mathrm{S}(\boldsymbol{v}, i)$ is generated successively. The advantage of this method depends on the code. For extended binary primitive BCH codes, permuting the symbol positions of codewords properly makes $\mathrm{S}(\boldsymbol{v}, i)$ larger [10]. We can obtain the dimension of $\mathrm{S}(\boldsymbol{v}, i)$ by row operations of the generator matrix G of C (refer to Figure 4.1).

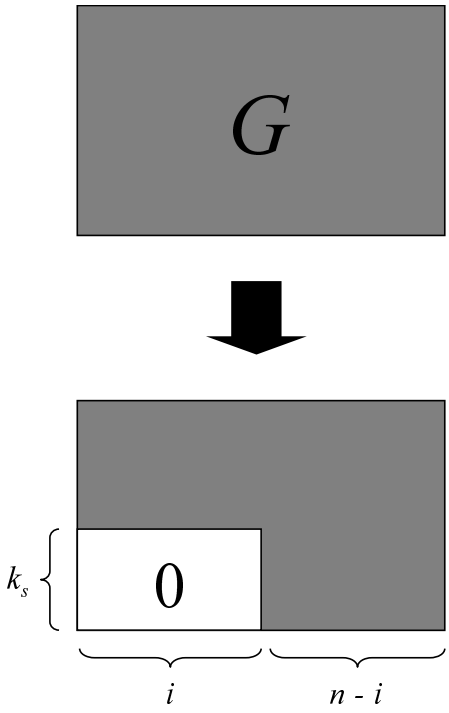


Figure 4.1: The way of obtaining the dimension of $\mathrm{S}(\boldsymbol{v}, i)$, denoted by k_s, by row operations of a generator matrix G

It is not easy to estimate precisely how the computational complexity is reduced. We will estimate the effect roughly in the case of the $(128, 50)$ extended BCH code. In this case, the $(128, 29)$ code is chosen as the subcode and the number of representative cosets is 258. We pick up $2^{15} \times 258$ codewords by choosing 2^{15} codewords randomly from each of the 258 representative coset. For every codeword $\boldsymbol{v}$ in such codewords, we examined a position at which the $\dim(\mathrm{C}(\boldsymbol{v}))$ is found out to be one or not. Then, the average was 100. Assuming that the complexity of obtaining $\mathrm{C}(\boldsymbol{v}, i+1)$ from $\mathrm{C}(\boldsymbol{v}, i)$ is proportional to the dimension of $\mathrm{C}(\boldsymbol{v}, i)$, we estimated the relative computational complexity to that without using the above technique when i_0 is chosen as i $(0 \leq i \leq n)$: $R_{i_0,k_s} = ((100 - i_0)/100)^2 + (1 - ((100 - i_0)/100)^2)/2^{k_s}$, where k_s is the dimension of $\mathrm{S}(\boldsymbol{v}, i_0)$. It turned

out that the complexity would reduced mostly by 1/2 for $i = 33, 47$, and 48 (refer to Table 4.1). Therefore, we choose 33 as i for the $(128, 50)$ code. Actually, for the $(128, 50)$ extended BCH code and the $(128, 29)$ extended BCH subcode, the complexity is reduced by about 1/2.

In the proposed algorithm, since the codewords that need to be checked are generated coset by coset, the subset of $\mathrm{S}(\boldsymbol{v}, i)$ in a coset is used instead of $\mathrm{S}(\boldsymbol{v}, i)$. If the dimension of the subcode is small, k_s may become small and the effect of using the trellis structure is small. We should choose the subcode considering the effect of using the trellis structure.

Table 4.1: Effect of using the trellis structure for the (128,50) extended primitive BCH code using the (128,29) primitive BCH code as a subcode. i is the selected position. k_s is the dimension of $\mathrm{S}(\boldsymbol{v}, i)$. R_{i,k_s} is the rate of reducing the complexity

i	k_s	R_{i,k_s}	i	k_s	R_{i,k_s}	i	k_s	R_{i,k_s}	i	k_s	R_{i,k_s}
1	28	0.9801	18	12	0.6725	35	3	0.4947	52	1	0.6152
2	27	0.9604	19	11	0.6563	36	3	0.4834	53	1	0.6105
3	26	0.9409	20	10	0.6404	37	2	0.5477	54	1	0.6058
4	25	0.9216	21	9	0.6248	38	2	0.5383	55	1	0.6013
5	24	0.9025	22	8	0.6099	39	2	0.5291	56	1	0.5968
6	23	0.8836	23	7	0.5961	40	2	0.5200	57	1	0.5925
7	22	0.8649	24	7	0.5809	41	2	0.5111	58	1	0.5882
8	21	0.8464	25	6	0.5693	42	2	0.5023	59	1	0.5841
9	20	0.8281	26	6	0.5547	43	2	0.4937	60	1	0.5800
10	19	0.8100	27	6	0.5402	44	2	0.4852	61	1	0.5761
11	18	0.7921	28	6	0.5259	45	2	0.4769	62	1	0.5722
12	17	0.7744	29	6	0.5118	46	2	0.4687	63	1	0.5685
13	16	0.7569	30	6	0.4980	47	2	0.4607	64	1	0.5648
14	15	0.7396	31	6	0.4843	48	2	0.4528	65	0	1.0000
15	14	0.7225	32	6	0.4708	49	1	0.6301	66	0	1.0000
16	14	0.7056	33	5	0.4661	50	1	0.6250			
17	13	0.6889	34	4	0.4709	51	1	0.6201			

4.2 Invariance property in cosets

In the proposed algorithm, the invariance property for zero neighborship is applied to the set of cosets of a subcode rather than the set of codewords. This reduce the complexity of finding the representatives. However, we do not use the invariance property sufficiently. That is, the invariance property is not used for codewords in cosets. Then, in computing the local weight subdistribution for a coset, we can apply the invariance property to codewords in the coset. An invariance property in a coset is given in the following theorem.

Theorem 14. For a coset $\boldsymbol{v} + C' \in C/C''$, $\pi \in \{\rho : \rho\boldsymbol{v} \in \boldsymbol{v} + C'\}$, and $\boldsymbol{u} \in \boldsymbol{v} + C'$, $\pi\boldsymbol{u}$ is a zero neighbor in C if and only if $\boldsymbol{u}$ is a zero neighbor in C.

No efficient way is known for generating the representative codewords in a coset as in a code. Then we use a similar fashion. As we applied the invariance property to the set of cosets in a code rather than the set of codewords in the code, we apply the invariance property to the set of cosets in a coset rather than the set of codewords in the coset. This means we consider a coset $\boldsymbol{v} + C' \in C/C'$ the set of cosets of C''.

For a coset $\boldsymbol{v} + C' \in C/C'$, let $(\boldsymbol{v} + C')/C''$ denote the set of all cosets of C'' in $\boldsymbol{v} + C'$, that is, $(\boldsymbol{v} + C')/C'' = \{\boldsymbol{v} + \boldsymbol{u} + C'' : \boldsymbol{u} \in C' \setminus C''\}$. Then,

$$|(\boldsymbol{v} + C')/C''| = 2^{k'-k''} \quad \text{and} \quad \boldsymbol{v} + C' = \bigcup_{E \in (\boldsymbol{v}+C')/C''} E,$$

where k' and k'' are the dimension of C' and C''. We also call the weight spectrum of zero neighbors in $E \in (\boldsymbol{v} + C')/C''$ the local weight subdistribution for E. The following theorem gives an invariance property for cosets in $(\boldsymbol{v} + C')/C''$.

Theorem 15. For $E_1, E_2 \in (\boldsymbol{v}+C')/C''$, the local weight subdistribution for E_1 and that for E_2 are the same if there exists $\pi \in \{\rho : \rho\boldsymbol{v} \in \boldsymbol{v} + C', \rho \in \mathrm{Aut}(C) \cap \mathrm{Aut}(C')\}$ such that $\pi[E_1] = E_2$.

We consider partitioning $(\boldsymbol{v} + C')/C''$ into equivalence classes. Permutations which are used to partition cosets into equivalence classes are presented in the following lemma.

Lemma 6. For a coset $\boldsymbol{v} + C' \in (\boldsymbol{v} + C')/C''$,

$$\begin{aligned} &\{\pi : \pi[E] \in (\boldsymbol{v} + C')/C'' \text{ for any } E \in (\boldsymbol{v} + C')/C''\} \\ &\quad = \{\rho : \rho\boldsymbol{v} \in \boldsymbol{v} + C',\ \rho \in \mathrm{Aut}(C) \cap \mathrm{Aut}(C') \cap \mathrm{Aut}(C'')\}. \end{aligned} \tag{4.3}$$

Proof. Let $\pi \in \{\rho : \rho\boldsymbol{v} \in \boldsymbol{v} + C',\ \rho \in \mathrm{Aut}(C) \cap \mathrm{Aut}(C') \cap \mathrm{Aut}(C'')\}$. For a coset $\boldsymbol{v} + \boldsymbol{v}_1 + C'' \in (\boldsymbol{v} + C')/C''$, suppose that $\pi\boldsymbol{v}_1 = \boldsymbol{v} + \boldsymbol{v}_2, \boldsymbol{v}_2 \in C'$ and $\pi\boldsymbol{v}_1 = \boldsymbol{v}_3 \in C'$. For any codeword $\boldsymbol{v} + \boldsymbol{v}_1 + \boldsymbol{u}_1 \in \boldsymbol{v} + \boldsymbol{v}_1 + C''$,

$$\begin{aligned} \pi(\boldsymbol{v} + \boldsymbol{v}_1 + \boldsymbol{u}_1) &= \pi\boldsymbol{v} + \pi\boldsymbol{v}_1 + \pi\boldsymbol{u}_1 \\ &= \boldsymbol{v} + \boldsymbol{v}_2 + \boldsymbol{v}_3 + \boldsymbol{u}_2, \quad \boldsymbol{u}_2 \in C'', \\ &= \boldsymbol{v} + (\boldsymbol{v}_2 + \boldsymbol{v}_3) + \boldsymbol{u}_2 \ \in\ \boldsymbol{v} + (\boldsymbol{v}_2 + \boldsymbol{v}_3) + C''. \end{aligned} \tag{4.4}$$

Thus, $\pi[\boldsymbol{v} + \boldsymbol{v}_1 + C''] = \boldsymbol{v} + (\boldsymbol{v}_2 + \boldsymbol{v}_3) + C'' \in (\boldsymbol{v} + C')/C''$. Then, $\{\pi : \pi[E] \in (\boldsymbol{v} + C')/C''$ for any $E \in (\boldsymbol{v} + C')/C''\} \supseteq \{\rho : \rho\boldsymbol{v} \in \boldsymbol{v} + C',\ \rho \in \mathrm{Aut}(C) \cap \mathrm{Aut}(C') \cap \mathrm{Aut}(C'')\}$.

Let $\pi \in \{\rho : \rho[E] \in (\boldsymbol{v} + C')/C''$ for any $E \in (\boldsymbol{v} + C')/C''\}$. For any codeword $\boldsymbol{v} + \boldsymbol{v}_1 \in \boldsymbol{v} + C'$, $\boldsymbol{v} + \boldsymbol{v}_1$ must be in either coset in $(\boldsymbol{v} + C')/C''$, and then $\pi(\boldsymbol{v} + \boldsymbol{v}_1) \in \boldsymbol{v} + C''$ and $\pi \in \mathrm{Aut}(C)$.

For $\boldsymbol{v} + \boldsymbol{v}_1 + C'' \in (\boldsymbol{v} + C')/C''$, let $\boldsymbol{v} + \boldsymbol{v}_1 + \boldsymbol{u}_1, \boldsymbol{v} + \boldsymbol{v}_1 + \boldsymbol{u}_2 \in \boldsymbol{v} + \boldsymbol{v}_1 + C''$. $\pi(\boldsymbol{v} + \boldsymbol{v}_1 + \boldsymbol{u}_1) = \pi\boldsymbol{v} + \pi\boldsymbol{v}_1 + \pi\boldsymbol{u}_1$ and $\pi(\boldsymbol{v} + \boldsymbol{v}_1 + \boldsymbol{u}_2) = \pi\boldsymbol{v} + \pi\boldsymbol{v}_1 + \pi\boldsymbol{u}_2$ must be in the same coset of $\boldsymbol{v} + \boldsymbol{v}_2 + C''$. Thus, $\pi \in \mathrm{Aut}(C'), \pi \in \mathrm{Aut}(C'')$. Then, $\{\pi : \pi[E] \in (\boldsymbol{v} + C')/C''$ for any $E \in (\boldsymbol{v} + C')/C''\} \subseteq \{\rho : \rho\boldsymbol{v} \in \boldsymbol{v} + C',\ \rho \in \mathrm{Aut}(C) \cap \mathrm{Aut}(C') \cap \mathrm{Aut}(C'')\}$. □

In order to partition cosets into equivalence classes, we should use permutations presented in Lemma 6. Although $\mathrm{Aut}(C)$, $\mathrm{Aut}(C')$, and $\mathrm{Aut}(C'')$ are known, we should obtain permutations π that satisfy $\pi\boldsymbol{v} \in \boldsymbol{v} + C'$.

For example, we consider the case of the third-order Reed-Muller code of length 256, denoted by $\mathrm{RM}(3, 8)$. The equivalent cosets in $\mathrm{RM}(3, 8)/\mathrm{RM}(2, 8)$ are presented in [9], and there are 32 equivalence classes. We choose $\mathrm{RM}(1, 8)$ as a subcode of $\mathrm{RM}(2, 8)$. Then, the general affine group is a subgroup of $\mathrm{Aut}(C) \cap \mathrm{Aut}(C') \cap \mathrm{Aut}(C'')$. For each cosets in $\mathrm{RM}(3, 8)/\mathrm{RM}(2, 8)$, the time complexity for computing the local weight subdistribution is about 1300 hours with the present algorithm, which is described in the previous chapter. The total time complexity is about 42000 hours. To compute the local weight distribution of $\mathrm{RM}(3, 8)$ in practical time, we can use the technique described in this section. For each coset $\boldsymbol{v} + \mathrm{RM}(2, 8)$ in $\mathrm{RM}(3, 8)/\mathrm{RM}(2, 8)$, we should find the permutations π that satisfy $\pi\boldsymbol{v} \in \boldsymbol{v} + \mathrm{RM}(2, 8)$ If we could find more than 50 permutations for each cosets, the local weight distribution of $\mathrm{RM}(3, 8)$ may be computable.

5 A Theoretical Method to Determine Local Weight Distribution Using Code Structure

In this chapter, we consider using code structure for determination of the local weight distribution. In other words, non-computational method is studied.

The algorithm proposed in Chapter 3 reduces the complexity for computing the local weight distribution using the automorphism group of the code. The algorithm is applied to extended primitive BCH codes, which are closed under the affine group of permutations. The size of the affine group is larger than that of the cyclic permutations. Then, the local weight distributions of the $(127, k)$ primitive BCH codes for $k \geq 36$ were not obtained although those of the corresponding $(128, k)$ extended primitive BCH codes are known. A method for obtaining the local weight distribution of a code from that of its extended code should be considered.

A relation between local weight distributions of a code, its extended code, and its even weight subcode is given. In this relation, *Only-odd decomposable* codewords are key codewords. First, a sufficient condition is shown under which no only-odd decomposable codeword exists. For a code that satisfies this sufficient condition, the local weight distribution of its extended code is determined from that of its original code. Second, for a transitive invariant extended code that satisfies the above sufficient condition, a relation between the code and its original code is presented. Extended binary primitive BCH codes and Reed-Muller codes are transitive invariant codes. Using the relation, the local weight distribution of a code is determined from that of its extended code. Finally, a relation between the local weight distributions of a code and its even weight subcode is given.

5.1 Determination of the local weight distribution of an extended code from that of the original code

Consider a binary linear code C of length n, its extended code $C_{\rm ex}$, and its even weight subcode $C_{\rm even}$. For a codeword $\boldsymbol{v} \in C$, let $\mathrm{wt}(\boldsymbol{v})$ be the Hamming weight of $\boldsymbol{v}$ and $\boldsymbol{v}^{(\rm ex)}$ be the corresponding codeword in $C_{\rm ex}$, that is, $\boldsymbol{v}^{(\rm ex)}$ is obtained from $\boldsymbol{v}$ by adding the over-all parity bit.

We define a *decomposable* codeword.

Definition 11 (Decomposable)**.** $\boldsymbol{v} \in C$ is called *decomposable* if $\boldsymbol{v}$ can be represented as $\boldsymbol{v} = \boldsymbol{v}_1 + \boldsymbol{v}_2$ where $\boldsymbol{v}_1, \boldsymbol{v}_2 \in C$ and $\mathrm{Supp}(\boldsymbol{v}_1) \cap \mathrm{Supp}(\boldsymbol{v}_2) = \emptyset$. From Lemma 1, $\boldsymbol{v}$ is not a zero neighbor if and only if $\boldsymbol{v}$ is decomposable.

For even weight codewords, we introduce *only-odd-decomposable* and *even-decomposable*.

Definition 12. Let $\boldsymbol{v} \in C$ be a decomposable codeword with even $\mathrm{wt}(\boldsymbol{v})$. That is, $\boldsymbol{v}$ is not a zero neighbor in C. $\boldsymbol{v}$ is said to be *only-odd-decomposable*, if all the decomposition of $\boldsymbol{v}$ is of the form $\boldsymbol{v}_1 + \boldsymbol{v}_2$ with the odd weight codewords $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. Otherwise, $\boldsymbol{v}$ is said to be *even-decomposable*.

When $\boldsymbol{v}$ is even-decomposable, there is a decomposition of $\boldsymbol{v}$, $\boldsymbol{v}_1 + \boldsymbol{v}_2$ such that both $\mathrm{wt}(\boldsymbol{v}_1)$ and $\mathrm{wt}(\boldsymbol{v}_2)$ are even. The relation between C and C_{ex} with respect to zero neighborship is given in the following theorem, which is also summarized in Table 5.1.

Theorem 16. 1. For a zero neighbor $\boldsymbol{v}$ in C, $\boldsymbol{v}^{(\mathrm{ex})}$ is a zero neighbor in C_{ex}.

2. For a codeword $\boldsymbol{v}$ which is not a zero neighbor in C, the following (a) and (b) hold.

   (a) When $\mathrm{wt}(\boldsymbol{v})$ is odd, $\boldsymbol{v}^{(\mathrm{ex})}$ is not a zero neighbor in C_{ex}.

   (b) When $\mathrm{wt}(\boldsymbol{v})$ is even, $\boldsymbol{v}^{(\mathrm{ex})}$ is a zero neighbor in C_{ex} if and only if $\boldsymbol{v}$ is only-odd-decomposable in C.

Proof. 1. Suppose that $\boldsymbol{v}^{(\mathrm{ex})}$ is not a zero neighbor in C_{ex}. $\boldsymbol{v}^{(\mathrm{ex})}$ is decomposable into $\boldsymbol{v}_1^{(\mathrm{ex})} + \boldsymbol{v}_2^{(\mathrm{ex})}$, then $\boldsymbol{v}$ is decomposable into $\boldsymbol{v}_1 + \boldsymbol{v}_2$. This contradicts the indecomposability of $\boldsymbol{v}$.

2. Suppose that $\boldsymbol{v}$ is decomposed into $\boldsymbol{v} = \boldsymbol{v}_1 + \boldsymbol{v}_2$. (a) Since $\mathrm{wt}(\boldsymbol{v})$ is odd, the sum of the parity bits in $\boldsymbol{v}_1^{(\mathrm{ex})}$ and $\boldsymbol{v}_2^{(\mathrm{ex})}$ is one. Also, the parity bit in $\boldsymbol{v}^{(\mathrm{ex})}$ is one. Then, $\boldsymbol{v}^{(\mathrm{ex})}$ is decomposable into $\boldsymbol{v}_1^{(\mathrm{ex})} + \boldsymbol{v}_2^{(\mathrm{ex})}$, and $\boldsymbol{v}^{(\mathrm{ex})}$ is not a zero neighbor in C_{ex}. (b) Since $\mathrm{wt}(\boldsymbol{v})$ is even, the parity bit in $\boldsymbol{v}^{(\mathrm{ex})}$ is zero. (If part) Suppose that $\boldsymbol{v}^{(\mathrm{ex})}$ is not a zero neighbor in C_{ex}. Then, there exists a decomposition $\boldsymbol{v}^{(\mathrm{ex})} = \boldsymbol{v}_1^{(\mathrm{ex})} + \boldsymbol{v}_2^{(\mathrm{ex})}$. Because the parity bit in $\boldsymbol{v}^{(\mathrm{ex})}$ is zero, the parity bits in $\boldsymbol{v}_1^{(\mathrm{ex})}$ and $\boldsymbol{v}_2^{(\mathrm{ex})}$ must be zero. This implies that $\boldsymbol{v}$ is even-decomposable into $\boldsymbol{v}_1 + \boldsymbol{v}_2$, and contradicts the assumption that $\boldsymbol{v}$ is only-odd-decomposable. (Only if part) Because $\boldsymbol{v}$ is even-decomposable, there is a decomposition such that the parity bits in both $\boldsymbol{v}_1^{(\mathrm{ex})}$ and $\boldsymbol{v}_2^{(\mathrm{ex})}$ are zero. For such the decomposition, $\boldsymbol{v}^{(\mathrm{ex})}$ is decomposable into $\boldsymbol{v}_1^{(\mathrm{ex})} + \boldsymbol{v}_2^{(\mathrm{ex})}$, and $\boldsymbol{v}^{(\mathrm{ex})}$ is not a zero neighbor in C_{ex}. □

For the local weight distributions of a code and its extended code, we have the following theorem, which is a direct consequence of Theorem 16.

Theorem 17. If there is no only-odd-decomposable codeword in C,

$$L_{2i}(C_{\mathrm{ex}}) \;=\; L_{2i-1}(C) + L_{2i}(C), \quad 0 \le i \le n/2. \tag{5.1}$$

Table 5.1: Zero neighborship of $\boldsymbol{v}$ in a linear block code, $\boldsymbol{v}_{\text{ex}}$ in its extended code, and $\boldsymbol{v}$ in its even weight subcode.

$\boldsymbol{v}$ in C			$\boldsymbol{v}^{(\text{ex})}$ in C_{ex}		$\boldsymbol{v}$ in C_{even}	
Zero neighborship	Weight	Decomposability	Zero neighborship	Theorem 16	Zero neighborship	Theorem 20
Yes	Odd	Not decomposable	Yes	1)	N/A	N/A
	Even				Yes	1)
No	Odd	Decomposable	No	2) - a)	N/A	N/A
	Even	Only-odd-decomposable	Yes	2) - b)	Yes	2)
	Even	Even-decomposable	No		No	

From Theorem 17, the local weight distributions of C_{ex} are obtained from that of C. Next, we give a useful sufficient condition under which no only-odd-decomposable codeword exists.

Theorem 18. If all the weights of codewords in C_{ex} are multiples of four, no only-odd-decomposable codeword exists in C.

Proof. If $\boldsymbol{v} \in C$ is an only-odd-decomposable codeword and decomposed into $\boldsymbol{v}_1 + \boldsymbol{v}_2$, the weights of $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ can be represented as $\text{wt}(\boldsymbol{v}_1) = 4i - 1$ and $\text{wt}(\boldsymbol{v}_2) = 4j - 1$ where i and j are integers. Then, $\text{wt}(\boldsymbol{v}) = \text{wt}(\boldsymbol{v}_1 + \boldsymbol{v}_2) = \text{wt}(\boldsymbol{v}_1) + \text{wt}(\boldsymbol{v}_2) = (4i - 1) + (4j - 1) = 4i + 4j - 2$. This contradicts the fact that $\text{wt}(\boldsymbol{v})$ is a multiple of four. □

For example, all the weights of codewords in the $(128, k)$ extended primitive BCH code with $k \leq 57$ are multiples of four. The parameters of Reed-Muller codes with which all the weights of codewords are multiples of four are given by Corollary 13 of Chapter 15 in [11]. The third-order Reed-Muller code of length 128, 256, and 512 are true for the case.

Although the local weight distribution of C_{ex} for these codes can be obtained from that of C by using Theorem 17 (see Figure 5.1), what we need is a method for obtaining the local weight distribution of C from that of C_{ex}. We need to know the number of zero neighbors with parity bit one. In the next section, we will show a method to obtain the number of zero neighbors with parity bit one for a class of transitive invariant codes.

5.2 Determination of the local weight distribution of a code from that of its transitive invariant extended code

A Transitive invariant code is the code which is invariant under a transitive group of permutations. A group of permutations is said to be transitive if for any two symbols in a

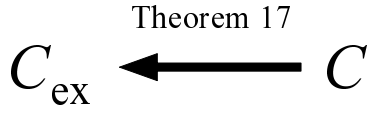


Figure 5.1: Computability of the local weight distribution of C_{ex} from that of C using Theorem 17 for the case that C contains no only-odd-decomposable codeword.

codeword there exists a permutation that interchanges them [16]. The extended primitive BCH codes and Reed-Muller codes are transitive invariant codes. For a transitive invariant C_{ex}, a relation between the (global) weight distributions of C and C_{ex} is presented in Theorem 8.15 in [16]. A similar relation holds for local weight distribution. The following lemma can be proved in a similar way as the proof of Theorem 8.15.

Lemma 7. If C_{ex} is a transitive invariant code of length $n+1$, the number of zero neighbors with parity bit one is $\frac{w}{n+1}L_w(C_{\text{ex}})$.

Proof. Arrange all zero neighbors with weight w in a column. Next, interchange the j-th column and the last column, which is the parity bit column, for all these codewords with the permutation. All the resulting codewords have weight w and must be the same as the original set of codewords. Thus, the number of ones in the j-th column and that in the last column are the same. Denote this number l_w, which is the same as the number of zero neighbors of weight w with parity bit one. Then, the total ones in the original set of codewords is $(n+1)\,l_w$, or $L_w(C_{\text{ex}})$ times the weight w. Thus, $(n+1)\,l_w = wL_w(C_{\text{ex}})$, and $l_w = \frac{w}{n+1}L_w(C_{\text{ex}})$. □

It is clear that there are $\frac{n+1-w}{n+1}L_w(C_{\text{ex}})$ zero neighbors with weight w whose parity bit is zero from this lemma. The following theorem is obtained from Theorem 16 and Lemma 7.

Theorem 19. If C_{ex} is a transitive invariant code of length n+1,

$$L_i(C) = \frac{i+1}{n+1}L_{i+1}(C_{\text{ex}}), \quad \text{for odd } i, \tag{5.2}$$

$$L_i(C) \leq \frac{n+1-i}{n+1}L_i(C_{\text{ex}}), \quad \text{for even } i. \tag{5.3}$$

If there is no only-odd-decomposable codeword in a transitive invariant code C_{ex}, the equality of (5.3) holds. That is, in this case, we have that

$$L_i(C) = \frac{n+1-i}{n+1}L_i(C_{\text{ex}}), \quad \text{for even } i. \tag{5.4}$$

Therefore, for a transitive invariant code C_{ex} having no only-odd-decomposable codeword in C, the local weight distributions of C can be obtained from that of C_{ex} by using (5.2) and (5.4) in Theorem 19 (refer to Figure 5.2).

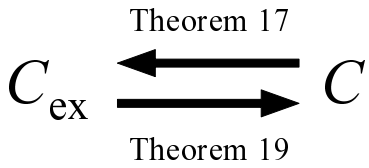


Figure 5.2: Computability of the local weight distributions of C and C_{ex} using Theorems 17 and 21 for the case that C contains no only-odd-decomposable codeword and C_{ex} is transitive invariant.

As discussed in this chapter, the local weight distributions of the $(127, k)$ primitive BCH codes for $k \leq 57$ and the punctured third-order Reed-Muller codes of length 127, 255, and 511 are obtained from those of the corresponding extended codes by using Theorem 19.

5.3 Determination of the local weight distribution of even weight codes

A similar relation as that between C and C_{ex} holds between C and C_{even}. This relation is given in Theorem 20 without proof (see Table 5.1).

Theorem 20. 1. For an even weight zero neighbor $\boldsymbol{v}$ in C, $\boldsymbol{v}$ is a zero neighbor in C_{even}.

2. For an even weight codeword $\boldsymbol{v}$ which is not a zero neighbor in C, $\boldsymbol{v}$ is a zero neighbor if and only if $\boldsymbol{v}$ is only-odd-decomposable in C.

From Theorem 20, we have Theorem 21.

Theorem 21. If there is no only-odd-decomposable codeword in C,

$$L_{2i}(C_{\text{even}}) \;=\; L_{2i}(C), \qquad\qquad 0 \leq i \leq n/2. \tag{5.5}$$

Therefore, after obtaining the local weight distribution of the $(127, k)$ primitive BCH codes for $k \leq 57$ and the punctured third-order Reed-Muller codes of length 127, 255, and 511 from those of their extended code using Theorem 17, the local weight distributions of their even weight subcodes can be obtained using Theorem 21. (see Figure 5.3).

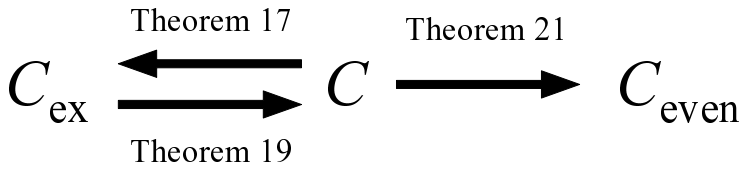


Figure 5.3: Computability of the local weight distributions of C, C_{ex}, and C_{even} from that of using Theorems 17 and 21 for the case that C contains no only-odd-decomposable codeword and C_{ex} is transitive invariant.

6 Obtained Local Weight Distributions

Extended primitive BCH codes

To determine the local weight distributions of extended primitive BCH codes, we used the algorighm described in Chapters 3 and 4. The local weight distributions of the $(128, k)$ extended primitive BCH codes for $k \leq 50$ are obtained and shown in Table 6.1. It took about 440 hours (CPU time) to compute the distribution of the $(128, 50)$ code with 1.6 GHz Opteron processor. In this case, the $(128, 29)$ code is used as the subcode, and it took only one minute to partition cosets into equivalence classes.

Comparison between the algorithm proposed by Mohri et al. [14, 15] and the proposed algorithm for extended primitive BCH codes with length 128 is presented in Table 6.2. Although the algorithm in [14, 15] is for computing the local weight distribution of cyclic codes, it can be applied to extended cyclic codes as described in Corollary 2. In the table, CT is the observed average time to check a codeword for zero neighborship without using the trellis structure. This is estimated by observing the CPU time for 2^{20} codewords. N_{MHM} and N_{new} are the number of codewords that need to be checked in the Mohri et al.'s algorithm and the proposed algorithm, respectively. PT is the time to partition the cosets into the equivalence classes. T_{MHM} (or T_{new}) are the estimated total time to obtain the local weight distributions from CT, PT, and N_{MHM} (or N_{new}). That is, $T_{\text{MHM}} = CT \times N_{\text{MHM}}$ and $T_{\text{new}} = PT + CT \times N_{\text{MHM}}$. k' is the dimension of the chosen subcode, such that N_{new} is enough small and PT is smaller than T_{new}. The table shows that the time complexity for partitioning the cosets into the equivalence classes is actually much smaller than that for checking each codewords for zero neighborship.

In the Mohri et al.'s algorithm, the computational complexity (or the number of codewords that need to be checked) for the extended BCH codes of length n is as much as that for BCH codes, which is about $1/n$ as much as that of the brute force method. The number of the affine permutations in (n, k) extended primitive BCH codes is $n(n-1)$. When the proposed algorithm is applied to the codes, the complexity is at most about $1/n^2$ as much as that of the brute force method. In fact, as shown in Table 6.2, the complexity compared with the Mohri et al.'s algorithm is $1/64$ in the case of $n = 128$.

Primitive BCH codes

The local weight distributions of the $(127, k)$ primitive BCH codes for $k \leq 57$ are obtained from those of the corresponding extended codes by using Theorem 19 in Chapter 5. The obtained local weight distributions are presented in Tables 6.3. Since the local weight distribution of the $(128, 57)$ extended primitive BCH code is unknown and the local

Table 6.1: Local weight distributions of extended primitive BCH codes of length 128.

(128,8) ex. BCH code

w	L_w
64	254

(128,15) ex. BCH code

w	L_w
56	8,192
64	16,638
72	8,192

(128,22) ex. BCH code

w	L_w
48	42,672
56	877,824
64	2,353,310
72	877,824
80	42,672

(128,29) ex. BCH code

w	L_w
44	373,888
48	2,546,096
52	16,044,672
56	56,408,320
60	116,750,592
64	152,623,774
68	116,750,592
72	56,408,320
76	16,044,672
80	2,546,096
84	373,888

(128,36) ex. BCH code

w	L_w
32	10,668
36	16,256
40	2,048,256
44	35,551,872
48	353,494,848
52	2,028,114,816
56	7,216,135,936
60	14,981,968,512
64	19,484,132,736
68	14,981,968,512
72	7,216,127,808
76	2,028,114,816
80	348,203,520
84	35,551,872
88	2,048,256

(128,43) ex. BCH code

w	L_w
32	124,460
36	8,810,752
40	263,542,272
44	4,521,151,232
48	44,899,876,672
52	262,118,734,080
56	915,924,097,536
60	1,931,974,003,456
64	2,476,669,858,944
68	1,931,944,645,120
72	915,728,180,224
76	261,375,217,152
80	43,168,588,288
84	2,464,897,280

(128,50) ex. BCH code

w	L_w
28	186,944
32	19,412,204
36	113,839,296
40	33,723,852,288
44	579,267,441,920
48	5,744,521,082,944
52	33,558,415,333,632
56	117,224,645,074,752
60	247,311,270,037,888
64	316,973,812,770,944
68	247,074,613,401,728
72	115,408,474,548,096
76	25,844,517,328,896

Table 6.2: Comparison of the prospective time complexity between the MHM algorithm and the proposed algorithm for extended primitive BCH codes of length 128.

k	CT (sec)	N_{MHM}	T_{MHM} (hour)	k'	PT (sec)	N_{new}	T_{new}* (hour)	$T_{\text{new}}/T_{\text{MHM}}$
36	1.44×10^{-5}	5.41×10^{8}	2.16×10^{0}	15	86	1.26×10^{7}	7.46×10^{-2}	1/29
43	2.02×10^{-5}	6.93×10^{10}	3.89×10^{2}	22	65	1.08×10^{9}	6.10×10^{0}	1/64
50	2.40×10^{-5}	8.87×10^{12}	5.92×10^{4}	29	66	1.39×10^{11}	9.25×10^{2}	1/64
57	3.02×10^{-5}	1.13×10^{15}	9.53×10^{6}	29	11138	1.77×10^{13}	1.48×10^{5}	1/64
64	3.31×10^{-5}	1.45×10^{17}	1.34×10^{9}	43	65	2.27×10^{15}	2.09×10^{7}	1/64
71	4.00×10^{-5}	1.86×10^{19}	2.06×10^{11}	43	7032	1.48×10^{17}	1.64×10^{9}	1/126
78	4.52×10^{-5}	2.38×10^{21}	2.99×10^{13}	50	11126	3.70×10^{19}	4.65×10^{11}	1/64
85	4.96×10^{-5}	3.05×10^{23}	4.20×10^{15}	64	65	4.76×10^{21}	6.56×10^{13}	1/64
92	5.68×10^{-5}	3.90×10^{25}	6.16×10^{17}	71	65	6.09×10^{23}	9.62×10^{15}	1/64
99	6.37×10^{-5}	4.99×10^{27}	8.83×10^{19}	71	7641	3.93×10^{25}	6.95×10^{17}	1/127
106	7.03×10^{-5}	6.39×10^{29}	1.25×10^{22}	85	65	9.98×10^{27}	1.95×10^{20}	1/64
113	7.72×10^{-5}	8.18×10^{31}	1.75×10^{24}	92	7667	6.44×10^{29}	1.38×10^{22}	1/127
120	8.48×10^{-5}	1.05×10^{34}	2.46×10^{26}	99	65	1.64×10^{32}	3.85×10^{24}	1/64

* This estimate is for computing time without using the technique in Section 4.1

weight distributions of the $(127, k)$ primitive BCH codes for $k \leq 29$ are obtained straightforwardly from their global weight distributions, only the local weight distributions of the $(127, k)$ primitive BCH codes for $k = 36, 43, 50$ are given in the table.

Reed-Muller codes

Using the algorithm described in Chapter 3, the local weight distribution of the third-order Reed-Muller code of length 128 is obtained and shown in Table 6.4. It took about 13 hours (CPU time) with 1.6 GHz Opteron processor. The details are described in section 3.6.2.

Punctured Reed-Muller codes

Using Theorem 19 in Chapter 5, the local weight distribution of the punctured Reed-Muller code of length 127 is obtained from that of the third-order Ree-Muller code. The distribution is shown in Table 6.5.

Even weight subcodes

From the local weight distributions of the $(127, k)$ primitive BCH codes for $k \leq 50$ and the punctured third-order Reed-Muller code of length 127, the local weight distributions of their even weight subcodes are obtained using Theorem 21 in Chapter 5. The distributions are shown in Table 6.6.

Table 6.3: The local weight distributions of the $(127, k)$ primitive BCH codes for $k = 36, 43,$ and 50.

$(127, 36)$ BCH code

w	L_w
31	2,667
32	8,001
35	4,572
36	11,684
39	640,080
40	1,408,176
43	12,220,956
44	23,330,916
47	132,560,568
48	220,934,280
51	823,921,644
52	1,204,193,172
55	3,157,059,472
56	4,059,076,464
59	7,022,797,740
60	7,959,170,772
63	9,742,066,368
64	9,742,066,368
67	7,959,170,772
68	7,022,797,740
71	4,059,071,892
72	3,157,055,916
75	1,204,193,172
76	823,921,644
79	217,627,200
80	130,576,320
83	23,330,916
84	12,220,956
87	1,408,176
88	640,080

$(127, 43)$ BCH code

w	L_w
31	31,115
32	93,345
35	2,478,024
36	6,332,728
39	82,356,960
40	181,185,312
43	1,554,145,736
44	2,967,005,496
47	16,837,453,752
48	28,062,422,920
51	106,485,735,720
52	155,632,998,360
55	400,716,792,672
56	515,207,304,864
59	905,612,814,120
60	1,026,361,189,336
63	1,238,334,929,472
64	1,238,334,929,472
67	1,026,345,592,720
68	905,599,052,400
71	515,097,101,376
72	400,631,078,848
75	155,191,535,184
76	106,183,681,968
79	26,980,367,680
80	16,188,220,608
83	1,617,588,840
84	847,308,440

$(127, 50)$ BCH code

w	L_w
27	40,894
28	146,050
31	4,853,051
32	14,559,153
35	310,454,802
36	793,384,494
39	10,538,703,840
40	23,185,148,448
43	199,123,183,160
44	380,144,258,760
47	2,154,195,406,104
48	3,590,325,676,840
51	13,633,106,229,288
52	19,925,309,104,344
55	51,285,782,220,204
56	65,938,862,854,548
59	115,927,157,830,260
60	131,384,112,207,628
63	158,486,906,385,472
64	158,486,906,385,472
67	131,258,388,369,668
68	115,816,225,032,060
71	64,917,266,933,304
72	50,491,207,614,792
75	15,345,182,164,032
76	10,499,335,164,864

Table 6.4: The local weight distribution of the third-order Reed-Muller code of length 128.

w	L_w
16	94,488
24	74,078,592
28	3,128,434,688
32	311,574,557,952
36	18,125,860,315,136
40	5,519,65,599,940,608
44	9,482,818,340,782,080
48	93,680,095,610,142,720
52	538,097,941,223,571,456
56	1,752,914,038,641,131,520
60	2,787,780,190,808,309,760
64	517,329,044,342,046,720

Table 6.5: The local weight distribution of the punctured third-order Reed-Muller code of length 128.

w	L_w
15	11,811
16	82,677
23	13,889,736
24	60,188,856
27	684,345,088
28	2,444,089,600
31	77,893,639,488
32	233,680,918,464
35	5,097,898,213,632
36	13,027,962,101,504
39	172,489,249,981,440
40	379,476,349,959,168
43	3,259,718,804,643,840
44	6,223,099,536,138,240
47	35,130,035,853,803,520
48	58,550,059,756,339,200
51	218,602,288,622,075,904
52	319,495,652,601,495,552
55	766,899,891,905,495,040
56	986,014,146,735,636,480
59	1,306,771,964,441,395,200
60	1,481,008,226,366,914,560
63	258,664,522,171,023,360
64	258,664,522,171,023,360

Table 6.6: The local weight distributions of the even weight subcodes of the $(127, k)$ primitive BCH codes for $k = 36, 43,$ and 50 and the punctured third-order Reed-Muller code of length 127 .

$(127, 35)$ BCH even weight subcode

w	L_w
32	8,001
36	11,684
40	1,408,176
44	23,330,916
48	220,934,280
52	1,204,193,172
56	4,059,076,464
60	7,959,170,772
64	9,742,066,368
68	7,022,797,740
72	3,157,055,916
76	823,921,644
80	130,576,320
84	12,220,956
88	640,080

$(127, 42)$ BCH even weight subcode

w	L_w
32	93,345
36	6,332,728
40	181,185,312
44	2,967,005,496
48	28,062,422,920
52	155,632,998,360
56	515,207,304,864
60	1,026,361,189,336
64	1,238,334,929,472
68	905,599,052,400
72	400,631,078,848
76	106,183,681,968
80	16,188,220,608
84	847,308,440

$(127, 49)$ BCH even weight subcode

w	L_w
28	146,050
32	14,559,153
36	793,384,494
40	23,185,148,448
44	380,144,258,760
48	3,590,325,676,840
52	19,925,309,104,344
56	65,938,862,854,548
60	131,384,112,207,628
64	158,486,906,385,472
68	115,816,225,032,060
72	50,491,207,614,792
76	10,499,335,164,864

(127,63) punc. RM even weight subcode

w	L_w
16	82,677
24	60,188,856
28	2,444,089,600
32	233,680,918,464
36	13,027,962,101,504
40	379,476,349,959,168
44	6,223,099,536,138,240
48	58,550,059,756,339,200
52	319,495,652,601,495,552
56	986,014,146,735,636,480
60	1,481,008,226,366,914,560
64	258,664,522,171,023,360

7 Conclusions and Future Research

7.1 Conclusions

In this thesis, some methods to determine the local weight distribution of binary linear block codes are studied.

As a computational method, an algorithm for computing the local weight distribution using the automorphism group of a code is proposed. In this algorithm, a code is considered a set of cosets of a linear subcode The set of cosets are partitioned into equivalence classes with the invariance property for zero neighborship under a group of permutations. The local weight distribution is obtained by computing the local weight subdistributions for each representative cosets. The algorithm can be applied to codes closed under a group of permutations. Extended primitive BCH codes are closed under the affine group and Reed-Muller codes are closed under the general affine group. The algorithm are applied to these codes, and we determined the local weight distributions for some of these codes.

As a theoretical method, relations between the local weight distribution of a code, its extended code, and its even weight subcode are studied. Only-odd decomposable codewords are key codewords when we intend to determine the local weight distribution of an extended code from that of the original code, or vice versa. A sufficient condition is derived under which no only-odd decomposable codeword exists. The local weight distributions for some of primitive BCH codes, and punctured Reed-Muller codes, and their even weight subcodes are determined from those of extended primitive BCH codes and Reed-Muller codes.

7.2 Future research direction

As described in Section 4.2, if we could find more than 50 permutations π that satisfy $\pi \boldsymbol{v} \in \boldsymbol{v} + \mathrm{RM}(2,8)$ for each coset $\boldsymbol{v} + \mathrm{RM}(2,8)$ in $\mathrm{RM}(3,8)/\mathrm{RM}(2,8)$, the local weight distribution of $\mathrm{RM}(3,8)$, or the third-order Reed-Muller code of length 256, is obtained. We should develop a technique for finding such permutations.

In other directions, a study on an error performance analysis of a code using the local weight distribution of the code is considered. Using the local weight distribution instead of the weight distribution can give a tighter upper bound than the usual union bound.

Acknowledgement

The author would like to express my sincere gratitude to my supervisor Professor Toru Fujiwara, for his insightful guidance, helpful advice, and continuous support.

The author would like to express my appreciation to Professors Shojiro Nishio, Fumio Kishino, Norihisa Komoda, and Shinji Shimojo for their invaluable advice.

The author also wish to thank Associate Professor Yasunori Ishihara and Assistant Professor Maki Yoshida for their insightful advice. Finally, the author would like to thank all members in Information Security Engineering Laboratory for their encouragement.

References

[1] E. Agrell, "Voronoi regions for binary linear block codes," IEEE Trans. Inform. Theory, vol.42, no.1, pp.310–316, Jan. 1996.

[2] E. Agrell, "On the Voronoi neighbor ratio for binary linear block codes," IEEE Trans. Inform. Theory, vol.44, no.7, pp.3064–3072, Nov. 1998.

[3] A. Ashikhmin and A. Barg, "Minimal vectors in linear codes," IEEE Trans. Inform. Theory, vol.44, no.5, pp.2010–2017, Sept. 1998.

[4] E.R. Berlekamp, "The technology of error-correcting codes," Proc. IEEE, vol.68, no.5, pp.564-593, May 1980.

[5] Y. Borissov, N. Manev, and S. Nikova, "On the non-minimal codewords in binary Reed-Muller codes," Discrete Applied Mathematics, vol. 128, issue 1, pp.65-74, May 2003.

[6] G.D. Forney, Jr., "Geometrically uniform codes," IEEE Trans. Inform. Theory, vol.37, no.5, pp.1241–1260, Sept, 1991.

[7] T. Fujiwara and T. Kasami, "The weight distribution of $(256, k)$ extended binary primitive BCH codes with $k \leq 63$ and $k \geq 207$," IEICE Technical Report, IT97-46, Sept. 1997.

[8] T.-Y. Hwang, "Decoding linear block codes for minimizing word error rate," IEEE Trans. Inform. Theory, vol.IT-25, no.6, pp.733-737, Nov. 1979.

[9] X. Hou, "GL(m,2) acting on R(r,m)/R(r-1,m)," Discrete Mathematics, 149, pp.99–122, 1996.

[10] T. Kasami, T. Tanaka, T. Fujiwara, and S. Lin, "On complexity of trellis structure of linear block codes," IEEE Trans. Inform. Theory, vol.39, no.3, pp.1057–1064, May. 1993.

[11] F.J. MacWilliams and N.J.A. Sloane, The theory of error-correcting codes, North-Holland, 1977.

[12] J. Massey, "Minimal codewords and secret sharing," Proc. 6th Joint Swedish-Russian Workshop of Info. Theory, pp.246-249, 1993.

[13] M. Mohri and M. Morii, "An algorithm for computing the local distance profile of cyclic codes," Proc. of ISITA2000, pp.445–448, Nov. 2000.

[14] M. Mohri, and M. Morii, "On computing the local distance profile of binary cyclic codes," Proc. of ISITA2002, pp.415–418, Oct. 2002.

[15] M. Mohri, Y. Honda, and M. Morii,"A method for computing the local distance profile of binary cyclic codes," IEICE Trans.Fundamentals (Japanese Edition), vol.J86-A, no.1, pp.60–74, Jan. 2003.

[16] W.W. Peterson and E.J. Weldon, Jr., Error-correcting codes, 2nd Edition, MIT Press, 1972.

[17] T. Sugita, T. Kasami, and T. Fujiwara, "The weight distribution of the third-order Reed-Muller codes of length 512," IEEE Trans. Inform. Theory, vol.42, no.5, pp.1622–1625, Sept. 1996.